A Comprehensive Framework for Secure Outsourcing of XML Data

Barbara Carminati¹, Elena Ferrari¹ and Elisa Bertino²

¹DICOM - University of Insubria, Via Mazzini, 5 - Varese, Italy {barbara.carminati, elena.ferrari}@uninsubria.it
²CERIAS and CS Department - Purdue University, West Lafayette, USA bertino@cs.purdue.edu

Abstract: Data outsourcing is today an emerging paradigm with many benefits in terms of reduction of costs, scalability and better service. For the widespread adoption of this paradigm, it is important that both the data owners and users have strong security guarantees with respect to data management. In this paper, we present a comprehensive framework for the secure outsourcing of XML data, by mainly focusing on confidentiality and completeness requirements. In the paper, besides presenting the techniques and related data structures underlying our framework, we illustrate the prototype system we have developed and discuss storage and communication overheads implied by our solution.

Keywords: Data outsourcing, XML, completeness, confidentiality, encryption.

1. Introduction

Data outsourcing represents today an emerging paradigm for a variety of data intensive applications, from grid computing to Web services and P2P systems. Relevant application domains include large-scale federated Digital Libraries, e-commerce catalogs, e-learning, collaborative applications, and content distribution networks. The main idea of data outsourcing is that the data owner is not any longer responsible for data management; rather it outsources its data (or portions of them) to one of more publishers that provide specialized data management services and query processing functions. Such an approach is scalable, results in highly efficient query executions, and reduces the management costs of the owner.

Clearly, the widespread adoption of data outsourcing is strictly related to the security concerns that both users and owners may have when data are managed by a publisher. For instance, how can the owner be sure that publishers do not send data to users that are not authorized according to the access control policies in place at the owner site? Additionally, also users may have serious security concerns. A user receiving an answer by a publisher wants to be sure that the publisher has not maliciously modified the data (for instance, by inserting fake records), or that it has delivered to the user all the data the user is allowed to see according to the owner's policies. Obviously, requiring publishers to be trusted wrt security is not always an appropriate solution in that it is not easy to verify large Web-based systems and moreover these systems can be easily penetrated. Thus, our goal is to ensure security properties even in the presence of untrusted publishers. In particular, in this paper, we focus on two crucial security requirements: confidentiality and completeness. Authenticity is also an important requirement; it has however been addressed by previous work [3] and thus we do not address it in this paper. When dealing with outsourced data, we need to address two different confidentiality requirements. The first, which we refer to as confidentiality wrt users, deals with protecting owner’s data against unauthorized accesses by users. The second, which we refer to as confidentiality wrt publishers, deals with protecting the owner's data from accesses by publishers. By completeness we mean that the user receiving a portion of data is able to verify whether he/she has received all the information he/she is allowed to see according to the owner access control policies.

In this paper, we present a framework for securing data outsourcing that is able to meet the above-mentioned security requirements. We focus on data expressed in XML [21]; however our approach can be easily extended to other types of data models. Our solution relies on the use of selective encryption and non-conventional digital signature techniques. The methods we have devised protect confidentiality both at the document and at the schema level. The framework relies on a set of additional information sent by the owner to both users and publishers to make possible the enforcement of security properties. To this purpose, we have devised a suite of XML data structures able to convey all the needed information.

In the paper, besides presenting the data structures we discuss their storage overhead and the effort required for their updates, when documents, policies or user credentials are modified. Several approaches have been proposed to address the above security requirements, some of which are discussed in Section 2. However, to our knowledge no comprehensive framework exists able to simultaneously enforce confidentiality, authenticity and completeness requirements for XML data.

A preliminary version of this paper appears in [5]. This paper significantly extends our previous paper by presenting the prototype implementation of our system and covering issues related to data structures management (i.e., storage complexity and update management), which have not been addressed in our previous work [5].

The remainder of this paper is organized as follows. Next section introduces relevant background. Section 3 presents an
overview of the framework, whereas Section 4 introduces the reference scenario we will use throughout the paper. Section 5 describes the techniques we have devised for confidentiality enforcement. Section 6 presents the XML-based data structures we have designed to transmit information from owners to users and publishers, whereas Section 7 describes how users can formulate queries over encrypted data and verify the completeness of the results. Section 8 describes the prototype implementation of our system, whereas Section 9 discusses the storage and update management overhead of our solution. Finally, Section 10 outlines some conclusions.

2. Background

In the last few years, the increasing interest in data outsourcing has led to the development of efficient strategies for ensuring security properties even in the presence of untrusted publishers. Several approaches have been proposed addressing the different security requirements discussed in the introduction; we refer the interested reader to [6] for a survey of those approaches. In this section, we focus only on the strategies that are the building blocks of our framework. In particular, we introduce the approaches proposed by Hacigumus et al. and Song et al. for querying encrypted data, and an approach based on the Merkle trees for authenticity verification.

Hacigumus et al. The approach proposed by Hacigumus et al. [10, 11, 12] exploits binning techniques and privacy homomorphic encryption to query encrypted relational data. Binning techniques are used to perform selection queries on encrypted data confidentiality. The basic idea of the scheme is the following. Given a set of words, the proposed scheme first encrypts each word using a symmetric encryption algorithm with a single secret key. Then, it generates the XOR of each encrypted word with a pseudorandom number. The resulting encrypted words can then be outsourced to the publisher. According to this scheme, when a user needs to search for a keyword \( W \), it generates the encrypted word \( \text{Enc}(W) \) and computes \( \text{Enc}(\bar{W}) \text{ XOR } S \), where \( S \) is the corresponding pseudorandom number. This simple scheme allows the publisher to search for keyword \( W \) in the encrypted data, by simply looking for \( \text{Enc}(W) \text{ XOR } S \), thus without gaining any information on the clear text. Since occurrences of the same word are xored with different pseudorandom numbers, by analyzing the distribution of the encrypted words, no information can be inferred regarding the clear text. According to such scheme, to formulate a query, users need to know information about the pseudorandom numbers. An enhanced scheme has been proposed [19] making it possible for the users to locally compute pseudorandom numbers without any interaction with the data owner. The scheme exploits a symmetric encryption function \( \text{Enc}(\cdot) \) and two pseudorandom numbers generator functions, namely \( F \) and \( f \).

In the following, with the notation \( F(x;k) \) (\( f(x;k) \), respectively), we denote the result of applying \( F \) (\( f \), respectively) to input \( x \) with key \( k \). In general, the scheme considers as input a set of clear-text words, \( W_1, W_2, \ldots, W_p \), with the same length \( n \). Given these set of words: \( 1 \) the data owner generates a sequence of pseudorandom values: \( S_1, \ldots, S_n \); \( 2 \) for each word \( W_i \), the outsourced encrypted word \( C_j \) is generated according to the following formula: \( C_j = \text{Enc}(W_j,k) \text{ XOR } S_j \), where \( K_j = \text{FB}_i(k) \), and \( \text{FB}_i \) denotes the first \( n \)-\( m \) bits of \( \text{Enc}(W_i,k) \).

When a user needs to search for a keyword \( W \), he/she sends the publisher \( \text{Enc}(W_k) \) and key \( K_j \), which can be locally computed by the user. Then, for each outsourced encrypted word \( C \), the publisher: (1) calculates \( C \text{ XOR } \text{Enc}(W_k) \); (2) takes the first \( n \)-\( m \) bits \( b_i \) of the resulting

\[ 1 \text{ Enc}(W_\cdot,k) \text{ denotes the encryption of } W \text{ with key } k. \]

\[ 2 \text{ This set of words can be obtained by partitioning the input clear-text into atomic quantities (on the basis of the application domain) and by padding and splitting the shortest and longest words.} \]

\[ 3 \text{ Parameter } m \text{ can be properly adjusted to minimize the number of erroneous answers due to collision of pseudorandom numbers generators } F \text{ and } f. \]
value, and computes $F(b(S_j,K_j))$; (5) if the result of $F(b(S_j,K_j))$ is equal to the $n-m+1$ remaining bits, then the encrypted word $C$ is returned. Indeed, if $C$ contains the searched encrypted word, then $Enc(W_k)\ XOR \ F(S_j,K_j)\ XOR \ Enc(W_k) = <S_j,P(S_j,K_j)>$, for the properties of the XOR operator. When a user receives $C$ as answer to a query, he/she is able to extract the value $Enc(W_k)$ from $C$ and thus to decrypt it, by simply using the decryption key $k$. Therefore, the scheme proposed in [19] assumes that users know $S_j$. In such a way, the user is able to recover $FB_j$, by xoring $S_j$ with the first $n-m$ bits of $C$. Having $FB_j$, the user can generate $K_j$ (i.e., $K_j = f(FB_j,k)$), which can be used to compute $F(S_j,K_j)$. Finally, having $<S_j,P(S_j,K_j)>$, the user is able to extract from $C$ the encrypted word $Enc(W_k)$, and to decrypt it.

**Merkle Trees.** Merkle [14] proposed a method to sign multiple messages, by producing a unique digital signature. The method exploits a binary hash tree generated by means of the following bottom-up recursive construction: at the beginning, for each different message $m$, a different leaf containing the hash value of $m$ is inserted in the tree; then, for each internal node, the value associated with it is equal to $h(l_1||h_l_2)$, where $h$ denotes the concatenation of the hash values corresponding to the left and right children nodes, and $H()$ is an hash function. The root node of the resulting binary hash tree can be considered as the digest of all messages, and thus it can be digitally signed by using a standard signature technique, and distributed. The main benefit of this method is that a user is able to validate the signature based only on a subset of messages, providing him/her the hash values of the missing messages.

### 3. System Overview

The architecture of our secure data outsourcing framework (cfr. Figure 1) is a standard third party architecture, which consists of three main entities: users, publishers, and data owners.

*Figure 1. Overall architecture*

The main task of the data owner is sending its documents to publishers and managing the user subscriptions. To enforce confidentiality wrt publishers, the owner encrypts its data before delivering them to publishers, and it does not send publishers any decryption keys. Confidentiality requirements wrt users are instead specified through a set of access control policies stating which users have the right to access which portions of the owner's documents. Access control policies are specified according to a credential-based access control model for XML data proposed by Bertino and Ferrari [4]. To ensure confidentiality wrt users, instead of encrypting data to be managed by publishers with a unique key, the owner encrypts different data portions of the same document with different keys, according to the specified access control policies. In particular, all data portions to which the same access control policies apply, hereafter-called *policy configuration*, are encrypted with the same key since they can be accessed by the same users. Keys are distributed to users by evaluating user credentials against the specified access control policies, when users subscribe to the owner.

Besides keys, the proposed framework requires other information to be transmitted by the owner to both publishers and users for security properties verification and enforcement. To limit the overhead incurred by these operations, all such information is stored into a Directory server managed by the owner. The Directory server contains three kinds of entries: the *publishers* entry, which is shared by all publishers and contains all encrypted documents they are entitled to manage, plus additional information they need for the correct functioning of the system; the *users* entry, shared by all the subscribed users, storing common user information; and a distinct *user ID* entry for each single user, containing needed information to verify security properties. After a mandatory subscription phase, each user/publisher receives by the owner the keys to access the corresponding entries in the directory. The keys corresponding to the policies satisfied by each user are stored by the owner in the corresponding *user ID* entry of the owner's Directory server. Additionally, the entry stores the *user policy configuration*, a certificate signed by the owner maintaining information on the access control policies the user satisfies. Such information is needed by users to verify the completeness of the results returned by publishers (see Section 7.2 for details).

Applying a cryptographic-based solution in a third party scenario requires addressing two main issues. The first is related to the fact that publishers operate on encrypted data. Therefore, we need some mechanisms to make them able to perform queries over them. To this purpose we use the approaches proposed by Hacigumus et al. and Song et al. (cfr. Section 2) and we adapt it to XML documents encrypted with multiple keys. Therefore, the owner provides the publisher, besides the encrypted nodes, also the ids of the partitions or the encrypted keywords associated with their values. Similarly, users find in the common *users* entry information on the partitioning and encrypting techniques adopted by the owner and use them to rewrite their queries before their submission to publishers. Users submit queries to publishers through a *client*, which can be downloaded from the owner site. Queries are formulated in XPath [21]. The client translates the queries submitted by users into a set of

---

1. Users are able to generate it using the pseudorandom number generator and knowing the seed.
2. Here and in the following we refer to symmetric encryption.
encrypted queries that publishers can evaluate directly on the managed documents. The second issue is the definition of encryption strategies able to mitigate as much as possible the security attacks that can be perpetrated against the system. In particular, we have addressed information inference attacks that a publisher (user) can perpetrate by analyzing the distribution of encrypted nodes. To overcome this drawback, in Section 6.1, we show how the scheme by Song et al. can be used to encrypt tagnames and attribute names and values, thus preventing inferences.

Authenticity is instead assured by the so-called Merkle signatures. A Merkle signature is a signature generated by the owner using a bottom-up computation on a whole XML document, based on Merkle Trees (cfr. Section 2). Such signature is generated before encrypting a document and it is provided to the publisher along with the corresponding document. The publisher will then forward it to the user querying the document to which it refers to. The nice property of Merkle signatures wrt standard digital signatures is that a user may validate them even if he/she does not receive by publishers the whole document over which the signature has been computed, provided that he/she receives a set of additional hash values referring to the missing portions of the document. This set of hash values is called Merkle hash path. In general, given two nodes \( v, w \) such that \( v \) belongs to Path\( (w) \), the Merkle hash path between \( w \) and \( v \), denoted as \( \text{MPath}(w,v) \), is the set of hash values needed to compute the Merkle hash value of \( v \) having the Merkle hash value of \( w \). More precisely, the Merkle hash path between \( w \) and \( v \) consists of all the Merkle hash values of \( w \)'s siblings, together with the Merkle hash values of all the siblings of the nodes belonging to the path connecting \( w \) to \( v \). It is important to note that since the publisher operates on encrypted data, it is not able to compute the Merkle hash values, and, as a consequence, it cannot generate the appropriate Merkle hash paths to be returned to the user submitting the query. For this reason, the owner gives the publisher a set of additional hash values, one for each document node that makes it able to generate the Merkle hash paths. Since the focus of the paper is on confidentiality and completeness enforcement we do not elaborate more on authenticity enforcement; details can be found in [3].

Our framework relies on a set of XML-based data structures to convey the needed information to both publishers and users. In particular, all additional information needed by publishers for query processing and authenticity verification is encoded in XML and attached to the encrypted XML document, forming the so called Security Enhanced Encryption (SE-ENC) of the original document. All SE-ENC documents are stored by the owner in the publisher’s directory entry. Similarly, all information needed by users for authenticity/integrity verification is encoded by the publisher in XML and attached to the query answer, resulting in the so called reply document. Finally, to make a user able to verify the completeness of a query result, the owner generates an additional XML-based data structure, called query template, containing the encrypted structure of the original document.

The query template has the twofold goal of making a user able to verify the completeness of the answers received by publishers, as well as to make easier the task of query submission, in that by inspecting the query template a user can obtain information on the structure of the documents (or portions) he/she is allowed to access. To avoid information leakage at the schema level, query templates are encrypted by the owner using the same strategy employed for XML documents. This means that a user can see only the portions of a query template corresponding to document portions he/she is allowed to access, according to the specified access control policies. Query templates are digitally signed by the owner, through a Merkle signature, to prevent alterations. All query templates are stored by the owner in the Users directory entry. More details on completeness verification are given in Section 7.2.

4. Running Example

In this section, we introduce the reference scenario we will use throughout the paper. This example has been inspired by the Reuters press agency, which exploits an XML-based language, called NewsML [8], to structure and publish news [17].

Figure 2. An example of XML encoded news

![Figure 2](https://example.com/figure2.png)

Throughout the paper we consider as data owner a press agency, called XMLPress, which produces news in an XML-based format. We assume that XMLPress makes its news available to customers (i.e., external providers and/or users), on the basis of different subscription fees. Furthermore, we suppose that XMLPress does not manage user interactions on its own, rather it outsources its news to one or more publishers, which are in charge of answering customer queries. To keep the reference scenario simple, we do not consider all the features of the NewsML language; rather we exploit a simplified version of this language, which encodes several textual news into a unique XML document. Figure 2 presents an example of XML document encoding textual news. Each outsourced news is represented by means of a different NewsItem subelement, which stores the news content into the DataContent subelement. Moreover, the
news is complemented with additional information helpful for searching news (e.g., creator, topic of the news and date of issue).

Furthermore, we assume that XMLPress offers two kinds of subscription: topic subscriptions, which allow customers to access only news related to selected topics (specified by customers during the registration), and temporal subscriptions, authorizing customers to access all the news published in a specified temporal interval.

In what follows, we show how the framework proposed in this paper allows XMLPress to outsource its data to external publishers by, at the same time, being assured that its data are not accessed by publishers and are accessed by customers according to the access constraints imposed by subscription fees in place in the press agency. Moreover, we show how users can verify the completeness of the news received by publishers.

5. Confidentiality Enforcement

In this section we explain the techniques we use to enforce confidentiality. Then, in the next section we describe the XML-based data structures we have designed for supporting the verification process. Since completeness verification can be explained only after such data structures have been described, we postpone its discussion to Section 7.2.

In traditional client-server architectures confidentiality is usually enforced by means of an access control mechanism, i.e., a trusted software module that mediates user access requests by authorizing only those in accordance with the specified access control policies. This solution is no longer applicable to third party architectures; the main reason is that publishers cannot be responsible for hosting the access control mechanism since we cannot rely on their trustworthiness. Therefore, as introduced in Section 3, to ensure confidentiality both wrt publishers and users, we propose a solution based on encryption techniques. Data are encrypted by owners, before their delivery to publishers, according to the specified access control policies. In particular, all data portions to which the same access control policies apply are encrypted with the same key. We refer to this document encryption driven by the specified access control policies as well-formed encryption. Publishers do not receive any key. In contrast, each user receives by the owner the keys corresponding to the policies he/she satisfies.

According to this strategy, confidentiality wrt publishers is ensured since publishers do not manage clear-text data. Moreover, generation of the well-formed encryption ensures confidentiality wrt users in that each user receives from the owner only the keys corresponding to the satisfied policies. This means that even if a publisher maliciously sends a user more information than the one he/she is allowed to see, the user cannot access it since he/she does not have the proper decryption keys.

Clearly, one important issue is related to key management in that well-formed encryptions may require to generate a large number of keys (in the worst case a number of keys equal to the number of possible policy configurations that can be built from the policy base). To solve this problem we use a hierarchical key management scheme that requires, in the worst case, to manage a number of keys linear in the cardinality of the policy base [1]. According to the adopted key management, by simply providing users with the encryption key associated with a policy, say $acp_j$, they are then able to locally derive all the encryption keys corresponding to those policy configurations containing $acp_j$.

5.1. Encryption

The second issue to be considered is how to make publishers able to evaluate queries over encrypted documents. Devising a solution to query encrypted data mainly depends on the nature of the data, that is, the data domain and the underlying data model. Usually, an XML document contains data to be modeled into elements, which in turn could contain other elements. By contrast, attributes are used to better describe data contained into the corresponding elements. Consider, for instance, the XML document shown in Figure 2.

```
<root>
  <content><title>Main news</title><body>
    <article>
      <element>
        <2nd>
        <topic>Big News</topic>
      </element>
      <element>
        <2nd>
        <topic>Interesting News</topic>
      </element>
    </article>
  </content>
</root>
```

DataContent elements contain the news itself, whereas attributes Topic and Date store additional information about news. Therefore, in devising methods to query XML encrypted data, we need to consider that elements could be searched based on their attributes values and/or their contents. Thus, we need a method that makes the publisher able to perform logical comparisons on encrypted attribute values, as well as keyword-based searches on encrypted element contents and textual attributes.

5.1.1. Textual data

To address this requirement, we have adopted two different strategies to query XML encrypted data. In particular, we use an approach similar to the one by Song et al. [19] to query elements and attributes with textual domain. By contrast, for non textual elements and attributes, we exploit the method by Hacigumus et al. [9]. Thus, we assume that before encrypting a node $n$, the encryption process determines $n$'s data domain. We now discuss in more details the strategies for textual and non-textual encrypted data.

Textual data. As introduced in Section 2, Song et al.'s approach [19] makes a publisher able to search for a specific keyword on encrypted textual data without loss of data confidentiality. Applying such an approach to our scenario requires two adjustments wrt the original formulation. The first is because the scheme by Song et al. [19] works for words of same length, whereas we need to consider words of variable length. Therefore, we extend the scheme proposed in [19] to the management of variable length words, as follows. Let $W$ be the longest word in the owner's dictionary, and let $L_W$ be the length of $W$. For each sequence of clear-text words: $W_1$, $W_2$, ..., $W_i$, with length $L_i \leq L_W$, the steps for encrypted words generation are the following: (1) for each word $W_i$, the data owner pads it with a sequence $pbdts$ of $L_W - L_i$ bits. (2) it generates a sequence of pseudorandom values $S_1$, ..., $S_m$, of length $L_m$, where $m$ is a parameter (see Section 2); (3) for each keyword $W_i$, the outsourced encrypted word $C_i$ is generated by the following formula: $W_i = Enc(W_i)pbdts.k ) \ XOR <S_i,F(S_i,K_j)>, where K_j = KBj(k)$ and $FBj$ denotes the first $n-m$ bits of $Enc(W_i)pbdts.k$.

According to this scheme, users have to know $L_W$, that is, the length of the longest word managed by the data owner, to be able to pad the searched keyword $W_i$ before submitting the query. We need also to define a method for keywords
selection. In particular, given a node \( n \), we need to state how to select keywords from \( n \)'s content. A naïve solution is to split the content of \( n \) into separate \( L_n \) blocks, and to treat them as distinguished keywords. This solution however is useless if we consider that the resulting encrypted words should be exploited for the search. Thus, it is obvious that some content analysis should be performed over the node's content before keywords selection. In particular, the proposed solution requires a first phase during which the owner preprocesses the textual data contained into an XML node and extracts a set of keywords\(^7\). Then, for each keyword \( W_i \), the system retrieves the encryption key \( k \) corresponding to the policy configuration applied to the XML node to which \( W_i \) belongs to and encrypts \( W_i \) according to the scheme introduced above. Hereafter, we denote with \( Cipt(W_i,k) \) instead of with \( Enc(W_i,k) \), the encrypted keyword obtained through the abovementioned scheme to highlight the fact that, for textual data, we do not adopt a standard encryption technique.

**Non-textual data.** To make publishers able to evaluate queries on encrypted non-textual data, we adapt the approach by Hacigumus et al. [9]. As far as partition generation is concerned, the choice of the most appropriate partitioning technique mainly depends on the node domain. For instance, for numeric data (such as integer, real, etc.), a strategy based on an equi-partitioning of the domain could be appropriate. In contrast, for temporal data, a partitioning based on time intervals could be more appropriate. Therefore, in our system we associate a different partitioning function with each possible data domain, with the exception of textual domain. Thus, given a node \( n \) of a document \( d \), by analyzing the XML schema defining \( d \) it is possible to select the appropriate partitioning function.

### 6. Data Structures

Our framework relies on the transmission of additional information from the owner to both publishers and users to make it possible the enforcement of security properties. Publishers should receive the partition ids and encrypted keywords corresponding to the data they manage, whereas users should receive information to authenticate portions of a requested document. To properly convey all this additional information, referred in what follows as *security information*, we have designed a set of XML-based data structures that will be described in the following sections.

#### 6.1 Security Enhanced Encryption

**Security Enhanced Encryption (SE-ENC) document** is transmitted by the owner to publishers and consists of the well-formed encryption of a document plus additional information needed for confidentiality and authenticity enforcement. Generation of SE-ENC documents consists of two main steps: (1) generation of the well-formed encryption, and (2) generation of the related security information.

The generation of the well-formed encryption is performed by the owner by first marking the nodes of the input document with the applicable access control policies. Then, all nodes with the same marking are encrypted with the same key. A further key is generated to encrypt document portions that are not covered by any policy (and therefore cannot be accessed by anyone). Different approaches for the encryption of XML documents have been proposed (see for instance [13, 20]). However, we prefer to adopt a slightly different approach that preserves as much as possible the structure of the original XML document in the document encryption. In particular, the well-formed encryption of a document \( d \) is an XML document \( d' \) with the same element-subelement and element-attribute relations of the original document but with the names and contents of all the nodes (i.e., elements and attributes) encrypted. Clearly, this choice makes query formulation and processing easier, but it could introduce some security threat. An untrusted publisher (or user) could infer information by analyzing the distribution of encrypted nodes. This threat could be easily perpetrated against tag names, attribute names and values, since an XML document may often contain repeated elements and several attributes with the same names (or values). To overcome this drawback, rather than symmetric encryption we apply the scheme proposed by Song et al. (see Section 2) to encrypt tag names and attribute names and values. In contrast, for element content we adopt traditional symmetric encryption (e.g., TDES, AES), that requires less computational resources. This choice is motivated by the fact that the probability of having a number of occurrences of the same encrypted element content sufficient for data dictionary attacks is small. This probability is further reduced in our context, where elements with the same content may be encrypted with different keys, if they are protected by different access control policies.

In formally defining the well-formed encryption and throughout the paper, we make use of a graph-based representation of an XML document. More precisely, we define an XML document as a tuple \( d=(V_d,E_d,\phi_d) \), where: \( V_d=V_d^\varepsilon \cup V_d^\nu \) is a set of nodes representing elements (\( V_d^\varepsilon \)), and attributes (\( V_d^\nu \)); \( \nu_d \) is a node representing the document element (called *document root*); \( E_d \) is the set of edges representing element-subelements, element-attributes relationships, or links between elements; and \( \phi_d \) is the edge labeling function. Moreover, hereafter given an element \( \nu \) (an attribute \( a \)) we denote with \( \nu.tagname \), and \( \nu.content \) (\( a.name \) and \( a.value \)) the element's tag name and content, respectively (the attribute name and value, respectively).

We are now ready to formally define a well-formed encryption.

**Definition 1. (Well-formed encryption).** Let \( d=(V_d,\nu_d,E_d,\phi_d) \) be an XML document. Let \( PC_{\nu}(d) \) be the set of policy configurations which apply to nodes in \( d \). Let \( Key(pc) \) be the encryption key associated with policy configuration \( pc \) and let \( V(pc) \) be the set of nodes to which \( pc \) applies. The well-formed encryption of \( d \) is an XML document \( d'=(V_d,\nu_d,E_d,\phi_d) \), such that:

- \( d' \) preserves the relationships in \( d \);
- \( \forall pc \in PC_{\nu}(d), \forall \nu \in V_d(pc), \exists v' \in V_d \) such that:

\(^7\) Several techniques developed in the Information Retrieval field can be used to this purpose.

Received December 19, 2008
1554-1010 $03.50 © Dynamic Publishers, Inc
Confidentiality information is used to make publishers able to process queries over encrypted data. In particular, it consists of the partition ids or encrypted keywords associated with the considered node. All these data are contained into a unique element, called Query-Info element, child of Sec-Info. In contrast, authenticity information makes publishers able to generate the Merkle hash paths needed by clients to verify authenticity of query answers. The Auth-Info element contains the hash values of the name and content of the corresponding node. The confidentiality and authenticity information corresponding to attributes are instead stored into a different element, called Attributes.

**Definition 2 (Security Information element).** Let $d'=(V',\lambda',\rho',E',\mathcal{F}_d')$ be the well-formed encryption of an XML document $d$. Let $v\in V'\setminus\lambda'$ be the encrypted element corresponding to $v\in V_d$. The Security Information element associated with $v$ is an XML element $s$ such that:

- $s\text{.tagName}=\text{Sec-Info}$;
- $s$ contains two subelements: Node-Info and Attributes, where:

- $v\text{.tagName} = \text{Ciph}(v\text{.tagName}, Key(pc))$,
  \[v\text{.content} = \text{Enc}(v\text{.content}, Key(pc)), \text{if } v\in V_d;\]
- $v\text{.name} = \text{Ciph}(v\text{.name}, Key(pc))$,
  \[v\text{.val} = \text{Ciph}(v\text{.val}, Key(pc)), \text{if } v\in V_d;\]

Once the well-formed encryption has been generated, it undergoes a second phase, during which it is complemented with security information. Security information is associated with each element by means of a particular element, called Security Information element (Sec-Info) element, which is added as an additional child of the corresponding element. The Sec-Info element contains security information (i.e., authenticity and confidentiality information) for both the element itself and all its attributes.

Occasionally, the number of attributes in $v$, with the same structure described above.

When all the Sec-Info elements have been added to a well-formed encryption, the last step to generate the SE-ENC document is adding a Sign element, storing the Merkle signature of the original document (cf. Section 3).

**Example 1.** For simplicity, assume that the owner has specified only two access control policies: a first policy, $acp_1$, representing a subscription to all the news related to security issues, and a second policy, $acp_2$, modeling a temporal subscription for October 2008. Moreover, suppose that $K_1$ is the encryption key used to encrypt nodes to which only policy $acp_1$ applies, whereas $K_2$ is the encryption key used to encrypt nodes covered by policy $acp_2$. Finally, suppose that $K_3$ is the encryption key used for encrypting nodes to which both $acp_1$ and $acp_2$ apply. As example of Sec-Info element, consider the one associated with the first NewsItem element of the XML document in Figure 2 and generated by considering the above-mentioned access control policies (see Figure 3). According to Definition 2, the corresponding Sec-Info element contains the security information related to NewsItem element as well as to all its attributes (i.e., Topic and Date). In particular, this information is modeled into distinguished Node-Info elements, whereas this information associated with NewsItem’s attributes is recorded in the Attributes subelement. Note that since the Topic attribute is equal to ‘Security’ and the Date attribute is equal to ‘09/29/2008’, only $acp_2$ applies to the NewsItem element and its attributes.

This implies that to encrypt the tag name of this element, as well as the name and values of its attributes, $K_2$ must be used. The same key is used to encrypt its data content. Moreover, according to Definition 2, the Node-Info subelement of a node (i.e., NewsItem, Topic and Date nodes) contains confidentiality information (i.e., the partition ids/encrypted keywords associated with the corresponding content) and the authenticity information. Thus, for instance, the authenticity information associated with NewsItem element consists of the hash value of its tag name and content. Note that, since the NewsItem element contains only subelements (see Figure 2), its data content is empty. Thus, the hash value of NewsItem content is empty (see the first H-content element in Figure 3). For the same reason, the confidentiality information associated with NewsItem element is empty.

In contrast, since the Topic attribute contains textual data the confidential information associated with it consists of the encrypted keywords extracted by its value, that is, the encryption of ‘Security’ (see the second Query-info element in Figure 3).
in Figure 3). Whereas, the confidential information associated with the Date attribute is the id of the partition to which the Date value belongs to (see the third Query-info element in Figure 3).

6.2 Query Template

The so called query template of an XML document $d$ has a twofold goal: (1) to make users able to formulate XPath queries against the documents managed by publishers and translate them in terms of partition ids/encrypted keywords, and (2) to make users able to verify the completeness of publishers’ answers. A query template is generated starting from an SE-ENC document, by pruning from it the encrypted data contents, attribute values and authenticity information. Additionally, the query template contains information on which access control policies apply to each node of the original document. Such information is crucial to correctly encrypt queries to be submitted to publishers. To minimize the required storage space we encode policy information into clear-text document $d$, where, starting from the left side, the value of the $i$-th bit is: 1, if the $i$-th policy applies to $n$; 0, otherwise. Then, we translate each 4-bits block of the resulting binary string into the corresponding hexadecimal representation.

This information is then stored as an additional attribute of the Sec-Info element, named PC. Finally, to make policy configurations understandable by clients, it is necessary to insert an additional element into the query template. This element, called Policy, contains the identifiers of the policies which apply to $d$.

Example 2. Consider the first DataContent element of the XML document in Figure 2, where both the access control policies of Example 1 apply. According to the proposed encoding strategy, the binary encoding of the policy configuration associated with this node is '0011', to which the hexadecimal value 'c'.

To prevent alterations to the query template, the owner signs it with a Merkle signature, which is stored into an additional Sign element. It is important to note that, since the query template is generated from the SE-ENC document, all its nodes are selectively encrypted according to the specified access control policies. Therefore, as it will be explained in Section 7, users can access only the portions of the query template for which they have a corresponding authorization. We omit the formal definition of the query template since it is very similar to the SE-ENC document.

6.3 Reply Document

The last data structure is used to complement the encrypted nodes returned by a publisher with additional information needed for authenticity verification. The resulting XML document, called reply document, is formally defined as follows.

**Definition 3. (Reply document)** Let $g=(V_q,\varphi,q_{\phi_g})$, be the SE-ENC document corresponding to an XML document $d$, let $u$ be a user, and $q$ be a query on $d$ submitted by $u$ to a publisher. Let $View(q,u)=(V_r,\varphi,q_{\phi_v})$, be the XML document answering $q$. The reply document of query $q$ with respect to $u$ is an XML document $r=(V_r,\varphi,q_{\phi_r})$, such that:

- $V_r=V_q \cup V_{ATT} \cup Sign$, where:
  - $V_{ATT}$ contains a node, called AttributeElement, for each attribute $a \in V_q$. This node represents an element whose data content is the value of $a$. The name of $a$ is stored into an additional attribute of AttributeElement, called AttrName. The node is a direct child of the node in $V_q$ corresponding to the element in $View(q,u)$ to which $a$ belongs to;
  - $Sign$ is an element, direct child of $\varphi_q$, containing the content of Sign in $g$;
  - each node $e \in V_r$ contains an attribute, called MbPath, containing the Merkle hash path between $e$ and its father.

Figure 4. An example of reply document

Figure 4 shows the reply document returned by a publisher to a user, who has requested the second DataContent element of the document in Figure 2.

7. Client-side Processing

In this section, we explain how users are able to submit encrypted queries to publishers and verify the completeness of query answers.

7.1 Query Submission

We assume that users submit queries through XPath [22]. XPath allows one to traverse the graph structure of an XML document and to select specific portions of the document according to some properties, such as the type of the elements, or specific content-based conditions. In general, an XPath expression consists of a location path, which allows one to select a set of nodes from target documents, which in turn consists of one or more location steps, separated among each other by a slash. In this paper, we consider a fragment of XPath in which conditions are specified by means of equality or comparison operators. Moreover, among the
functions supported by XPath, we consider the contains() function, which allows keyword based searches on textual data. To query an XML document through XPath, it is necessary to know the corresponding schema. For this reason, the user retrieves from the directory server the query templates of the interested documents. This operation is required only the first time user inquires a document or when its document schema is modified.

Let us now see how the user can exploit the query template to formulate queries that can be processed against encrypted documents. For simplicity, in the following we refer to queries over single documents. A user can access only selected portions of query templates, that is, only those nodes for which he/she has the appropriate decryption keys. In that way we prevent information leakage that may arise by accessing document schemas. Therefore, the client first generates the authorized view of the query template, where by authorized view we mean an XML document containing only the authorized portions of the query template. To decrypt a node, the client has to know which decryption key has to be used. This information can be derived from the PC attribute and the Policy element of the query template. The view of the query template is built by a function, called View(), which takes as input the policy configuration of a user \( u \) that, we recall, contains the ids of access control policies applying to nodes and the query template, and turns the set of decrypted nodes, into a well-formed XML document. The resulting view is then displayed to the user, making him/her able to formulate XPath queries on it.

Let us see how Ann is able to detect the malicious behavior of the untrusted publisher. Consider the query template in Figure 5(a) and the access control policies of Example 1. Suppose that Ann is interested in receiving all the news, and therefore she submits to the publisher the encrypted version of the query: ‘News/*’. Moreover, suppose that an untrusted publisher sends Ann only the first NewsItem element and not the third. Let us now see how Ann is able to detect the malicious behavior of the publisher. The completeness verification considers the query submitted to the publisher, that is, XPath expressions returned by the query rewriting algorithm in [6] are evaluated against the SE-ENC structure and rewrite them in terms of partition ids/encrypted keywords. An Algorithm doing all the above mentioned operations is presented in [6].

### 7.2 Completeness Verification

Let us now see how a user can verify the completeness of a query answer by using the query template. The idea is quite simple. He/she performs the translated queries against the corresponding query template and compares the resulting nodes against the nodes returned by the publisher. However, the nodes returned by evaluating one of these queries on the query template could be a superset of the nodes the user is entitled to see according to the owner access control policies. Thus, in order to verify the completeness of query answers, the client must also consider the access control policies specified on the requested document. Policy information are contained in the Policy element and PC attributes of the query template.

**Example 4.** Consider the query template in Figure 5(a) and the access control policies of Example 1. Suppose that Ann is interested in receiving all the news, and therefore she submits to the publisher the encrypted version of the query: ‘News/*’. Moreover, suppose that an untrusted publisher sends Ann only the first NewsItem element and not the third. Let us now see how Ann is able to detect the malicious behavior of the publisher. The completeness verification considers the query submitted to the publisher, that is, XPath expressions returned by the query rewriting algorithm in [6]. More precisely, the user XPath expression ‘News/*’ is transformed by the algorithm into: ‘Ciph(News,K_0)*’. The evaluation of this query on the query template returns three elements with tag name NewsItem. The client must then prune from these nodes those for which Ann has no authorization. To do that, the client verifies the policy configuration of each of these nodes by checking the PC attribute stored into the corresponding Node-Info element. More precisely, considering the value of the Policy element, the nodes for which Ann has an authorization are those whose policy configuration has the 1-st bit set equal to 1, that is, 0001, 0011, to which characters ‘a’ and ‘c’ correspond. Thus, the client prunes the second NewsItem element, since its policy configuration does not match Ann’s policy configuration. As a result of the completeness verification, the client returns Ann the first and the third NewsItem elements. Therefore,
she verifies that in the answer received by the publisher an element is omitted.

A formal proof of the correctness of the completeness verification process is reported in [6].

8. Prototype Implementation

In this section, we describe the developed Java-based prototype implementing the proposed framework. The prototype consists of three main components, i.e., owner, publisher and client. The owner has been designed as a Web-based application, where by means of a JSP interface users subscribe to the service and administrators manage the owner functionalities. In contrast, the publisher has been divided as a socket, whose main goal is to listen and answer client queries. The client component is a Java tool that makes users able to formulate queries, to decrypt and view the results and verify their security properties.

Each single component exploits a set of XML repositories in order to manage the needed data, and several Java libraries for the needed functionalities, like BouncyCastle as encryption engine [8], JDOM and Xerces [15, 16] to handle XML documents and XPath queries. In what follows, for each component, we introduce its functionalities and main modules.

8.1 Owner

The owner is in charge of generating and managing the data to be outsourced and the specification and update of policies.

![Figure 6. Owner architecture](image)

As depicted in Figure 6, in order to implement these functionalities, the owner architecture includes three main components, namely, Policy manager, Document manager and Subscription manager. The Policy manager provides a GUI to specify, update and delete access control policies. In what follows we focus on the Document manager and Subscription manager, since they represent the more innovative components being strictly related to secure data outsourcing.

**Subscription manager.** The first step of the subscription phase is the selection of the access control policies that apply to the user, among those defined by the owner and stored into the Policy Base (i.e., PB repository). Since our framework exploits a credential-based access control model to specify access control policies, the owner first assigns the user one or more credentials (by activating the Credential manager module, see Figure 5), which are then stored in the Credential repository. Given the assigned credentials, the owner is able to state which access control policies apply to the subscribing user. The matching between credentials and access control policies is performed by the Policy matching module, which returns the ids of the matching policies to the Policy configuration issuer. These ids are then recorded into the user policy configuration, which is signed with the private key of the owner.

**Document manager.** This component receives as input an XML document and creates the corresponding SE-ENC and QT (query template) document. First, the Document manager computes the Merkle signature of the original XML document as well as the hash values of each single node of the original document. In case of a node with textual domain, this module extracts the set of meaningful keywords from the node, and generates the corresponding encrypted words. In contrast, for a node with non-textual domain, the Query-Info generator exploits a library of partitioning functions, which associates with each different non-textual data domain a unique partitioning function. The XML document is now ready to be encrypted. Encryption is executed by first marking the nodes of the input document with the policies that apply to them. This task is performed by the Policy marker, which queries the Policy matching module to determine which access control policies apply to a node. Then, it groups together all the nodes marked by the same access control policies. After that, the Encryption generator module differently encrypts each group with the key corresponding to that policy configuration, on the basis of the nature of the data it contains (textual vs. non textual). The appropriate key is retrieved by means of the Key manager module. This module is in charge of the generation and management of the encryption keys according to the hierarchical key assignment scheme we use. More precisely, the Key repository stores only the first level encryption keys, i.e., the encryption keys associated with a single access control policy, whereas all the other encryption keys are generated by the Key manager only when needed. Once the well-formed encryption has been generated, the resulting XML document undergoes a final phase, during which it is complemented with the Merkle signature, the hash values and the partition ids associated with each node and the associated encrypted keywords, if any. This task is performed by the SE-ENC generator, which retrieves all the needed information from XHash and Partition ids repositories.

The generation of query template documents is easier than the generation of SE-ENC documents. Indeed, as introduced in Section 6.2, the query template document is basically the SE-ENC document without encrypted contents and...
authenticity information, and with additional information about the access control policies that apply to each node. Thus, given an XML document, the QT generator first retrieves the corresponding SE-ENC document, and applies to it an XSLT transformation [21] pruning from the SE-ENC document the encrypted data contents, attribute values and authenticity information. Then, for each node, the QT generator queries the Policy matching module to determine which access control policies apply to the node. Then, it encodes information about the applied access control policies according to the strategies illustrated in Section 6.2 and stores the results into the corresponding PC attribute.

8.2 Publisher

The main goal of the publisher is to answer client queries and generate reply documents containing, in addition to the query answer, information needed by Clients to validate the authenticity of the answer (i.e., the Merkle hash paths).

The first task of the publisher, that is, answering the submitted queries, is very simple. Indeed, the publisher, upon receiving an encrypted XPath query, retrieves the requested SE-ENC document and evaluates the query directly on it.

Figure 7. Publisher architecture

As depicted in Figure 7, this is done by the XPathAnswer module, which exploits the Xerces library as XPath parser, and passes the returned view to the Reply generator module, which is the core component of the publisher. Given the view answering the submitted query, the process performed by this module to generate the corresponding reply document requires several steps. A reply document basically consists of a view of the SE-ENC document containing only the encrypted nodes answering the query, without all the additional information (i.e., Query-info and Auth-Info elements). According to Definition 3, all attributes contained into the resulting nodes are encoded in the reply document by means of additional elements (i.e., AttributeElement), which contain the MhPath attribute storing the corresponding Merkle hash paths. Thus, as first task, the view of the SE-ENC document returned by the XPathAnswer module is passed to the Attribute generator module which replaces each of its attributes with the new element. After that, all the elements are complemented with the corresponding Merkle hash path. More precisely, since the SE-ENC data content is encrypted, the MhPath generator module exploits the hash values inserted by the owner into the SE-ENC document. Moreover, to limit the storage overhead, before inserting an hash value into a MhPath attribute, the MhPath generator module verifies whether this value is already stored into another previously computed MhPath attribute. If this is the case, it inserts in the new MhPath attribute only a reference to such an existing value is inserted, instead of the hash value itself. Finally, the reply document is complemented with the Merkle signature stored into the Sign element of the SE-ENC document corresponding to the view.

8.3 Client

The client component has been devised to make the user able to submit queries to publishers, decrypt the results and verify their security properties. To perform these tasks, the Client architecture includes three main modules, namely Query submitter, Answer viewer and Security evaluator (cf. Figure 8). In what follows, we briefly describe these components.

Figure 8. Client architecture

Once the user has selected the document to be queried, the Query specification module retrieves the corresponding query template document. Obviously, the user is not able to define the query on the query template document, since it is encrypted. Thus, the query template document is passed to the Answer viewer module, which extracts from it the authorized view and shows to the user its decryption. On the decrypted view of the query template document, the user is thus able to define his/her XPath queries. As introduced in Section 7, in order to make the publisher able to evaluate user XPath queries on SE-ENC documents, they have to be translated (see [6] for more details). This translation is performed by the Query translator module, which passes the obtained encrypted XPath queries to the QuerySocket module. This module is in charge of submitting encrypted XPath queries to publishers and waiting for the corresponding reply documents. Once the QuerySocket module receives a reply document, it forwards it to the Answer viewer module. Here the reply document undergoes a process similar to the one executed over the query template document to extract the authorized view. The final task performed by the client is security validation. This task implies both authenticity and completeness validation. In the client implementation, these tasks are performed by two separate modules, namely Authenticity and Completeness module, which, by retrieving the additional information stored into the reply and query template documents, are able to determine whether the view is authentic and complete.

9. Data Structure Management

In this section, we discuss two important issues related to
the proposed system. The first, is the storage overhead of the defined data structures, whereas the second relates to their management upon policies, documents or users updates.

9.1 Storage Complexity

The storage overhead of the proposed infrastructure is related to the SE-ENC, query template and reply documents. Let us first consider SE-ENC documents. According to our approach, to generate an SE-ENC document, the owner complements each element of the well-formed encryption with a Sec-Info element. The Sec-Info element contains the confidentiality and authenticity information associated with the corresponding element, plus the confidentiality and authenticity information associated with each attribute of the corresponding element. Thus, let $n$ be number of nodes (i.e., attributes and elements) of an XML document. A first estimation of the storage overhead of SE-ENC is $n*(|\text{Auth-Info}| + |\text{Query-Info}|)$, where $|\text{Auth-Info}|$ (resp. $|\text{Query-Info}|$) is the overhead imposed by authenticity (resp. confidentiality) information.

We start by first analyzing the storage overhead imposed by the authenticity information. This information consists of two hash values, that is, the hash value of the data content of the corresponding element (value of an attribute, respectively), and the hash value of the tag name (name of the attribute, respectively). Therefore, the $|\text{Auth-Info}|$ overhead is $2*|\text{HashSize}|$, where $|\text{HashSize}|$ denotes the size of the value returned by the adopted hash function.\(^9\) The storage overhead of confidentiality information is determined by the query processing information, which makes the publisher able to query encrypted data. Query processing information consists of the encrypted words and partition ids associated with the node. Therefore, the overhead mainly depends on the data type of the node. If the element (attribute, respectively) contains non-textual data, the corresponding query-processing information consists of a unique id representing the partition to which that value belongs to. In contrast, if the node contains textual data, the corresponding query processing information consists of the encrypted keywords extracted from the text. In order to estimate this overhead, in the worst case, we assume a situation where each word of the text is selected as a keyword. Thus, let $ke$ be the number of words in the text, the storage overhead of query processing information is $ke*|\text{Lw}|$, where $|\text{Lw}|$ is the length of the longest word in the dictionary of the owner (see Section 5). The resulting storage overhead is thus $n*(2*|\text{HashSize}| + ke*|\text{Lw}|)$. To complete the estimation of the storage overhead of an SE-ENC document, we have to consider also the additional Sign attribute containing the Merkle signature. However, we can assume this overhead irrelevant for the final estimation, therefore the total storage overhead of the SE-ENC document is $O(n*(2*|\text{HashSize}| + ke*|\text{Lw}|))$. It is very important to note that this result is obtained by considering the worst case, that is, each node has a textual domain and every single word contained in the text represents a keyword.

We now analyze the storage overhead of query templates. A query template is computed by an XSLT transformation, which prunes from the SE-ENC document the encrypted data contents and attribute values as well as authenticity information. Additionally, the query template contains the Policy elements and PC attributes. Regarding PC attributes, the overhead of the hexadecimal representation introduced in Section 6.2 is linear to the number of access control policies. More precisely, let $Np$ be the number of access control policies that apply to document $d$. The size of the policy configuration of an element $e$ is $Np/4$ characters. The overhead of the Policy element is not relevant for the overall computation. Thus, we can estimate the storage overhead as: $O(n*kw*|\text{Lw}| + Np/4)$. Finally, we estimate the storage overhead of the reply document. According to the proposed framework, the reply document contains the nodes answering the query submitted by the user, plus additional information needed for authenticity verification. By Definition 3, this last information consists of the MhPath attributes added to each node. These attributes contain a set of hash values. More precisely, given a terminal node $e$ the maximum size of the corresponding MhPath attribute is equal to $|\text{HashSize}|*Nc + \sum_{f \in \text{path}(e)} |\text{HashSize}|*2+|\text{Lw}|(f)$, where $|\text{Lw}|(f)$ denotes the number of $f$'s siblings and $Nc$ denotes the number of children of node $e$. Note that the sets of Merkle hash values stored into different MhPath attributes could be not disjoint. In the prototype we have developed, we exploit a strategy that allows us to store each hash value only once among all the MhPath attributes. This approach allows us to reduce the total number of hash values stored in all the MhPath attributes to the number of nodes of the document. Thus, in the worst case, the storage overhead is $n*|\text{HashSize}|$.

9.2 Update Management

In this section, we discuss the impact of updates in our system. We first consider updates to the XML source and to the Policy Base, and we estimate the computation required to the owner for update management and the communication overhead required for updating publishers/users. Due to the key management scheme we adopt, such updates do not require any update to user keys. Then, we consider updates on user's credentials. However, it is important to note that according to the architecture in Figure 1, each time a modification occurs the owner has simply to store the updated data structures into the publisher's entry and/or user_ID entry of the Directory server, and notify the publishers/users about that.

XML source updates. We can classify the updates that can occur to an XML document or DTD/Schema into two groups: (1) modification of an existing node (i.e., an update to the data content of an element or the value of an attribute), and (2) insertion/deletion of a node.

(1) The modification of a node requires the regeneration of the Merkle signature of the document to which the node belongs to, the well-formed encryption, and, as a consequence, of the SE-ENC and query template documents.

\(^9\) The size of the hash value depends on the algorithm used. It can be 128 or 160 bits.
However, all these updates can be executed by incrementally updating the corresponding documents without the need of regenerating them from scratch:

(1).i. For updates of Merkle signatures, the owner can reuse the Merkle hash values of the portions that are really affected by the modification. More precisely, the Merkle hash values that need to be recomputed are, besides the Merkle hash value of the node whose content has been modified, only those of the nodes belonging to the path connecting the modified node to the document root.

(1).ii. Since we support content-based access control, an update to the content of a node can imply a modification of the policy configuration applied to it. Indeed, it could be the case of an access control policy that did not apply to the node before the update and starts to apply to it after the update occurs, since the updated content satisfies the conditions specified in the policy (or vice versa). Since document encryption is driven by policy configurations, when the above described situation occurs, managing the update also requires the re-encryption of the modified node with the encryption key corresponding to the eventually new policy configuration. However, it is important to note that even if some nodes are encrypted with a different encryption key, that is, the encryption key associated with a new policy configuration, no updates need to be sent to users. Indeed, according to the exploited hierarchical key management scheme by receiving the encryption keys associated with the satisfied access control policies a user is able to locally derive the encryption keys corresponding to all and only policy configurations containing at least one of the satisfied policies.

(1).iii. The SE-ENC document update can be executed by regenerating and sending to publishers only those portions affected by the node modification. We recall that the SE-ENC document consists of the well-formed encryption of a document plus information enabling authenticity verification and querying processing. The update of the well-formed encryption requires the replacement of the encryption of the modified portion with the new one. In contrast, the update of authenticity information requires the replacement of the hash values contained in the Auth-Info nodes corresponding to all those nodes belonging to the path connecting the modified node to the root. For what concerns the update of query processing information, we need to update the partition ids and the encrypted keywords contained into the Query-Info element of the modified node. Finally, the owner has to generate the Merkle signature of the updated SE-ENC document and store it into the publisher’s entry.

(1).iv. The update of the query template can be executed by modifying and storing into the publishers entry only those portions that are really affected by the node modification. More precisely, only those portions that need to be updated are those regarding an eventual new policy configuration and an eventual new set of partitions ids/encrypted keywords implied by the content modification (i.e., PC attribute and Query-Info element in the Sec-Info element corresponding to the modified node). Finally, the owner regenerates the Merkle signature over the updated query template, and stores it into the publisher’s entry.

(2).i. The insertion of a new node into an XML document or DTD/XML Schema entails the regeneration of the Merkle signature, well-formed encryption, and obviously of the SE-ENC and query template documents. Like the previous case, it is possible to perform an incremental update of all these data structures. More precisely, in the case of a node insertion the operations for updating the Merkle signature and well-formed encryption are the same presented above. As a consequence, the overhead required to update publishers is the same imposed by content update management.

(2).ii. The deletion of an existing node can be managed like in the previous cases, with only few differences. The owner has to regenerate the Merkle signature without considering the deleted node, and thus to recompute the Merkle hash values affected by this deletion, that is, only those of the nodes belonging to the path connecting the deleted node to the document root. By contrast, instead of regenerating the SE-ENC document the owner can simply notify the publisher that the node has been deleted, by sending also the new Merkle signature together with the new hash values. Update of the query template is similar to the one presented for content update management.

**Policy base updates.** Each time an access control policy is inserted, deleted, or modified, the well-formed encryption, SE-ENC and query template documents should be modified accordingly. We start by considering the update of the well-formed encryption. In order to efficiently manage this operation, we adopt a strategy proposed by us in [7] that incrementally maintains the well-formed encryption by changing all and only those portions of the document which are really affected by the operation, without the need of re-encrypting the document from scratch. A modification of the well-formed encryption obviously implies an update of the corresponding SE-ENC document. More precisely, each node to which the newly inserted policy applies, as well as each node to which a revoked policy does no more apply, must be re-encrypted by using the encryption key corresponding to the policy configuration formed after the modification. Thus, the SE-ENC document must be updated with these new encrypted portions. By contrast, since the query template does not contain data content, its update can be executed by inserting only the new policy configurations, and the new Merkle signature computed after the update. It is important to note that even if some nodes are encrypted with new generated encryption keys, we do not need to send keys to users, since according to hierarchical key management scheme users are able to locally generate the new encryption keys.

**User credentials updates.** An update to a user credential could change the set of access control policies that apply to the considered user. This obviously impacts key management in that key delivering is performed on the basis of access control policies satisfied by user credentials. Thus, if a user credential update makes a policy no longer satisfied by a user, the system has to ensure that the user is no longer able to use the key associated with that policy to access the corresponding data. This implies the revocation of the
encryption key and its replacement with a new one. Obviously, this requires an additional effort to distribute the new key to all the other interested users, that is, all users to which the policy still applies. However, wrt this overhead we can make some important considerations. We recall that credentials are sets of properties possessed by users. Such properties can be classified into two groups: static properties and dynamic properties. The first represents user properties that do not change during the life of the user. Examples of such properties are the date of birth or the SSN. Obviously for this kind of properties no updates are expected, and thus the system does not have to manage the problem of key revocation. By contrast, dynamic properties can change during the life of the user. An example of these properties is, for instance, the role that an employee plays in an organization. However, to minimize the required overhead a possible solution is to exploit a key management scheme able to constraint the validity time interval of the encryption keys (for instance, the one proposed [2]). Indeed, a possible solution is to associate temporal encryption keys with access control policies whose user specifications depend on dynamic credential properties. The key validity period can be set by estimating the frequency of properties updates. Thus, once the life time of the temporal key has expired, the system should generate a new one to be associated with the considered access control policy.

10. Conclusion

In this paper, we have presented a comprehensive framework for secure outsourcing of XML data, by mainly focusing on confidentiality and completeness enforcement. The framework makes use of different encryption strategies and non conventional digital signature techniques. In the paper, besides describing the underlying techniques, we have presented the XML data structures we have designed on support of the proposed framework. Additionally, we have described the prototype system we have developed and discuss the overhead implied by update management of the defined data structures. We plan to extend the work reported in this paper along several directions. First, we are currently performing an extensive performance evaluation of the proposed system. Additionally, we plan to consider further security properties, such as for instance privacy and ownership protection.

References


Author Biographies

Barbara Carminati is an assistant professor Computer Science at the University of Insubria, Italy. Her main research interests are related to security and privacy for innovative applications, like XML data sources, semantic Web, data outsourcing, Web service and data streams. On these topics she has published more than forty publications in international journals and conference proceedings. She is a member of the Editorial Board of Computer Standards & Interfaces Journal, Elsevier press. She has been the Program Co-Chair of the First International ICDE Workshop on Secure Semantic Web, and chair of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT). Barbara Carminati served as Program Committee member of several international conferences and workshops, among which the most recent are: International Conference on Data Engineering (ICDE), ACM Symposium on Access Control Models and Technologies (SACMAT), International Workshop on Privacy Aspects of Data Mining (PADM). Barbara Carminati served as reviewer for several international journals including, ACM Transactions on Internet Technology (TIST), IEEE Transactions on Knowledge and Data Engineering (TKDE), ACM Transactions on Information and System Security (TISSEC). She is a member of the ACM and IEEE.

Elena Ferrari is a full professor of Computer Science at the University of Insubria, Italy, where she heads the Database & Web Security Group. From 1998 until January 2001, she has been an assistant professor at the Department of Computer Science of the University of Milano (Italy). She received the MS degree in Computer Science from the University of Milano (Italy) in 1992. In 1998, she received a Ph.D. in Computer Science from the same university. Her research activities are related to various aspects of data management systems, including web security, access control and privacy, multimedia databases, and temporal databases. On these topics she has received December 19, 2008
published more than 120 scientific publications in international journals and conference proceedings. She gave several invited lectures and tutorials in Italian and foreign universities as well as on international conferences and workshops. Dr. Ferrari has served as Program Chair of the 4th ACM Symposium on Access Control Models and Technologies (SACMAT'04), Software Demonstration Chair of the 9th International Conference on Extending Database Technology (EDBT'04), Co-Chair of the first and second ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD, the first COMPSAC'02 Workshop on Web Security and Semantic Web, the first ECOOP Workshop on XML and Object Technology (XOT 2000), and the first ECOOP Workshop on Object-oriented Databases. She has also served as Program Committee member of several international conferences and workshops. Prof. Ferrari is in the Editorial Board of the VLDB Journal, the Transactions on Data Privacy, and the International Journal of Information Technology (IJIT). She is a member of the ACM and senior member of IEEE.

Elisa Bertino is professor of Computer Science at Purdue University and serves as Research Director of CERIAS. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Telecordia Technologies. Her main research interests include security, privacy, database systems, object-oriented technology, and multimedia systems. In those areas, Prof. Bertino has published more than 300 papers in all major refereed journals, and in proceedings of international conferences and symposia. She has served until 2006 as the coordinating co-editor in chief of the Very Large Database Systems (VLDB) Journal. She serves, or has served, on the editorial boards of several scientific journals, including IEEE Internet Computing, ACM Transactions on Information and System Security, AMC Transactions on Web, IEEE Transactions on Secure and Dependable Computing, IEEE Security&Privacy, IEEE Internet Computing. She is a Fellow member of IEEE and a Fellow member of ACM and has been been named a Golden Core Member for her service to the IEEE Computer Society. She received the 2002 IEEE Computer Society Technical Achievement Award for “For outstanding contributions to database systems and database security and advanced data management systems” and the 2005 IEEE Computer Society Tsutomu Kanai Award “For pioneering and innovative research contributions to secure distributed systems”.

Received December 19, 2008