CERIAS Tech Report 2004-14

AN APPROACH TO COOPERATIVE UPDATES OF XML DOCUMENTS IN DISTRIBUTED SYSTEMS

by Elisa Bertino, Elena Ferrari, Giovanni Mella

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

An Approach to Cooperative Updates of XML Documents in Distributed Systems*

E. Bertino[1], E. Ferrari[2], G. Mella[3]
[1]CERIAS and CS Department
Purdue University
Recitation Building
656 Oval Drive
West Lafayette, IN 47907-2086
bertino@cerias.purdue.edu
[2]Dipartimento di Scienze Chimiche, Fisiche e Matematiche
Universita' dell'Insubria
Via Valleggio, 11
22100 Como, Italy
elena.ferrari@uninsubria.it
[3]Dipartimento di Informatica e Comunicazione
Universita' degli Studi di Milano
Via Comelico, 39/41
20135 Milano, Italy
mella@dico.unimi.it
Tel. +39-0250316346, Fax +39-0250316276
corresponding author

Abstract

Protection and secure exchange of Web documents is becoming a crucial need for many internet-based applications. Securing Web documents entail addressing two main issues: confidentiality and integrity. Ensuring document confidentiality means that document contents can only be disclosed to subjects authorized according to specified security policies, whereas by document integrity we mean that the document contents are correct with respect to a given application domain and that the document contents are modified only by authorized subjects. Whereas the problem of document confidentiality has been widely investigated in the literature, the problem of how to ensure that a document, when moving among different parties, is modified only according to the stated policies still lacks comprehensive solutions. In this paper we present a solution to this problem by proposing a model for specifying update policies, and an infrastructure supporting the specification and enforcement of these policies in a distributed and cooperative environment, in which subjects in different organizational roles can modify possibly different portions of the same document. The key aspect of our proposal is that, by using a combination of hash functions and digital signature techniques, we create a distributed environment that enable subjects, in most cases, to verify, upon receiving a document, whether the update operations performed on the document till that point are correct with respect to the update policies, without interacting with the document server. Our approach is particularly suited for environments, such as mobile systems, pervasive systems, decentralized workflows, and peer-to-peer systems.

*A preliminary version of this paper appeared in *Proc. of the 16th Annual IFIP WG 11.3, Working Conference on Data and Application Security*, Cambridge, UK, July 2002, pp 211-227, with the title "A Framework for Distributed and Cooperative Updates of XML Documents".

1 Introduction

The Internet has made possible a wide spectrum of distributed cooperative applications in several areas, such as collaborative e-commerce [15], distance learning, telemedicine, e-government. A requirement common to many cooperative application environments is the need for secure document exchange. By secure exchange we mean that document confidentiality and integrity are ensured when documents flow among different parties within an organization or within different organizations. Ensuring document confidentiality means that document contents can only be disclosed to subjects authorized according to access control policies agreed upon by the various parties. Ensuring document integrity means that the document contents be correct with respect to a given application domain and that the document contents be modified only by authorized subjects. It is a common case in many application environments that not all parties be authorized to modify any document that is exchanged among these parties. Rather, different parties can be given selective update privileges to different documents, or even different components of the same document. Whereas the problem of documents confidentiality has been widely investigated [8], the problem of how to ensure that a document, when exchanged among different parties, is modified only according to the stated policies still lacks comprehensive solutions. We believe that such a comprehensive solution requires:

1. A model and a high-level language for specifying update policies - such a model and language are crucial whenever several parties need to state commonly agreed-upon policies according to which documents can be modified by the involved parties.

2. An infrastructure supporting the specification and enforcement of such policies in a distributed environment.

In this paper, we present such a comprehensive solution. We assume that documents to be protected are encoded in XML [16]. We have chosen to cast our approach in the framework of XML documents because of the widespread adoption of such a document standard in a large variety of application environments. However, we believe that our approach can be easily extended to other document exchange formats.
The key ingredients of our approach can be summarized as follows. We provide an access control model supporting, besides several document browsing privileges, various authoring privileges, such as deleting and modifying document elements and attributes, or inserting new elements and attributes into documents. These authorization privileges support a fine granularity level of control on document modifications. An important aspect of our access control model is the use of subject credentials. A credential is a set of properties concerning a subject that are relevant for security purposes (for example, the position of the subject within the organization, projects a subject is working on, etc.). Authorizations are then expressed by specifying the subjects receiving the authorizations in terms of conditions against the subject credentials. Subject credentials thus represent a way to support access control based on subject qualifications and profiles. In our model, both credentials and update policies are encoded in XML. Therefore, not only we provide a high-level language for policy representations, but we can also apply the protection mechanisms we provide for regular XML documents to credentials and access control policies. Such a capability is crucial in an environment where credentials and access control policies themselves need to be exchanged among the various parties, for purposes such as access control policy negotiations.
Our access control model is complemented by an infrastructure supporting secure cooperative document updates. The basic idea underlying our approach is that the server sends the document to be modified to a given subject; this subject operates on the document and then forwards the document to a second subject and so forth. Each subject[1] upon receiving the document from the server or from the previous

[1]By subject we mean either a human user or a software application.

subject along the path must be able to modify all and only those portions of the document for which it has a proper authorization according to the specified security policies. The main goal of our approach is to enable a subject, upon receiving a document, to verify whether the updates performed on the document till that point are correct with respect to the stated policies. Our approach is based on the use of hash functions and digital signatures. The proposed document infrastructure is particularly suited for decentralized environments, such as decentralized workflows, mobile systems, agent systems and e-commerce. In such environments, it is not always practical or possible requiring frequent client connections with document servers.

The work presented in this paper has been developed in the framework of the Author-$\mathcal{X}$ project [2]. Author-$\mathcal{X}$ is a Java-based system for access control and security policy design for XML documents. For access control, Author-$\mathcal{X}$ supports credential-based policy specifications at varying granularity levels. Additionally, Author-$\mathcal{X}$ supports push and pull distribution policies for document release. A number of administration tools are also provided, to facilitate security administration according to the underlying security policies. What we describe in this paper are the techniques and protocols provided by Author-$\mathcal{X}$ to enforce distributed document updates. This is a major extension since it requires, besides an extension to the policy specification language, the development of an infrastructure and related algorithms for supporting correct update operations in a distributed environment in which subjects can autonomously verify the correctness of update operations without interacting, in most cases, with the document server. These features were not supported by the previous versions of Author-$\mathcal{X}$ and, to the best of our knowledge, they have not been proposed before.

The remainder of this paper is organized as follows. Section 2 compares our work with other proposals. Section 3 briefly summarizes basic concepts of XML and the access control model on which our infrastructure for distributed update relies. Section 4 introduces the architecture for the management of distributed updates, whereas Section 5 presents document encryption as a way to enforce access control. Data structures required to support distributed updates and document dispatching are covered by Section 6, whereas Section 7 describes the protocols used by the subjects and the server to check document integrity, and gives an illustrative example of our approach. Section 8 gives details about the implementation of the document integrity verification protocols. Section 9 reports a complexity analysis of the most relevant operations executed by the proposed protocols, whereas Section 10 discusses possible extensions to the proposed protocols. Section 11 concludes the paper and outlines future research directions. Finally, Appendix A presents correctness results for the subject protocol.

2 Related work

Several research groups from both academia and industry are currently investigating problems related to security and XML. Work in this field has mainly focused on the development of access control models and encryption techniques for XML documents (an overview of research work and commercial products related to XML security can be found in [8]). To the best of our knowledge, the work reported in this paper is the first to address the problem of XML document distributed updates. Even though we are not aware of other proposals to which our model can be directly compared, the access control model on which our infrastructure relies has some relationships with access control models and mechanisms developed for object-oriented DBMSs [7, 9], HTML documents [11] and, recently, XML documents [4]. Additional related work includes the XACML and SAML standards proposed by OASIS [6, 12]. For this reason, in what follows we briefly review these proposals and compare them with our work.

The models proposed in [7, 9] are specifically tailored to an object-oriented DBMS storing conventional, structured data. As such, great attention has been devoted to concepts such as versions and composite objects, which are typical of an object-oriented context. Like our model, those models support the concept of authorization propagation, even if our model has a larger variety of authorization propagation options in that it supports three different options by which the Security Administrator (SA) can specify:

i) that an authorization defined at a given level in the XML document hierarchy propagates to all lower levels; *ii)* that the propagation stops at the first level down in the hierarchy; or *iii)* that no propagation has to be enforced. By contrast, ORION [9] has only one propagation policy, which is equivalent to the first option.

An access control model for WWW documents has been proposed in [11]. In such a model, HTML documents are organized as unstructured pages connected by links. Authorizations can be granted either on the whole document or on selected document portions. Although we borrow from [11] the idea of selectively granting access to a document (by authorizing a subject to see only some portions and/or links in the document), our work substantially differs from this proposal. Differences are due to the richer structure of XML documents with respect to HTML documents and to the possibility of attaching a DTD/XMLSchema to an XML document, describing its structure. Such features require the definition and enforcement of more sophisticated access control policies, than the ones devised for HTML documents. The access control model proposed in [11] has great limitations deriving from the fact that it is not based on a language able to semantically structuring the data, as in our model for XML. As such, authorization administration is very difficult. In particular, if one wants to give access to portions of a document, it has to manually split the page into different *slots* on which different authorizations are given. This problem is completely overcome by our model because XML provides semantic information for various document components. Authorizations can thus be based on this semantic information.

An access control model for XML documents has been recently proposed [4]. Such model is very similar to previous models for object-oriented databases and does not actually take into account some peculiarities of XML. In particular, this model has two main shortcomings. The first is that it does not consider the problem of a secure massive distribution of XML documents and thus considers only the information pull mode for document distribution. Second, the model proposed in [4] does not provide access control modes specific to XML documents. It only provides the read access mode. By contrast, we provide a number of specialized access modes for browsing and authoring, which allow the SA to authorize a subject to read the information in an element and/or to navigate through its links, or to modify/delete the content of an element/attribute.

An extensible access control markup language (XACML) has been recently proposed as OASIS standard [6]. There are two main differences between this language and the one on which our model relies. XACML supports the concept of authorization propagation only for the request specification, but it does not support this feature for policy specification. By contrast, our language supports several authorization propagation options for policy specification as already mentioned. Moreover, XACML supports the concept of subject's role [10], whereas our model is based on the more general concept of subject's credentials.

Finally a security assertion markup language (SAML) has also been recently proposed as OASIS standard [12]. This language supports the specification of authorization requests and responses. SAML has a very general notion of protection object in that a protection object is generically a resource, whereas our model is specifically tailored for XML documents. As such SAML is not able to support several features that are relevant to the protection of XML documents. Moreover SAML supports the following types of actions to be exercised on a resource: Read, Write, Execute, Delete and Control; whereas our model supports a greater and more specific set of privileges to be exercised on XML documents, allowing a subject to modify both elements and attributes. Finally, our work includes not only the definition of a language but also the development of a system, able among other things to support certified distributed updates. Note, however, that our approach to enforce secure distributed updates to XML documents can be used also when different document models and update authorization languages are adopted.

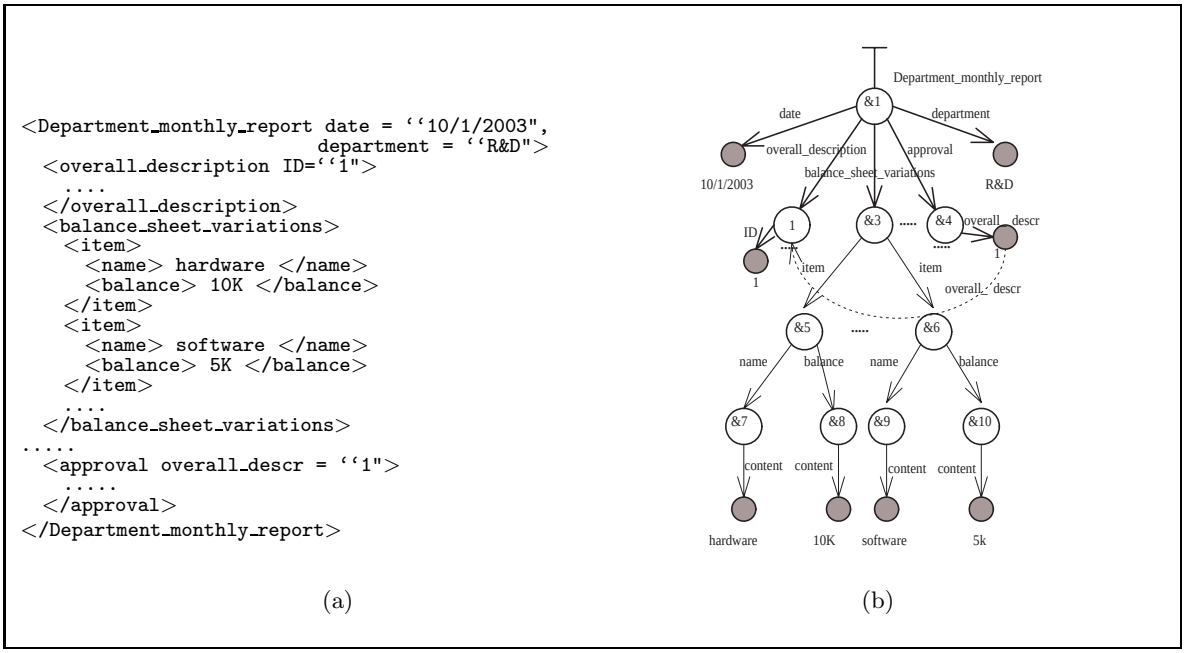

Figure 1: (a) An example of XML document and (b) its graph representation

3 Preliminaries

In this section we first review the basic concepts of XML. We then summarize the basic characteristics of the access control model that our system supports. More details on the model can be found in [2].

3.1 Basic Concepts of XML

Building blocks of XML documents [16] are nested, tagged *elements*. Each tagged element has zero or more subelements, zero or more attributes, and may contain textual information (data content). Elements can nested at any depth in the document structure. Attributes are of the form *name* = *attvalue*, where *name* is a label and *attvalue* is a string delimited by quotes. Attributes can have different types allowing one to specify the element identifier (attributes of type `ID` often called *id*), additional information about the element (e.g., attributes of type `CDATA` containing textual information), or links to other elements of the document (attributes of type `IDREF/URI` referring to a single target or `IDREF(s)/URI(s)` referring to multiple targets). An example of XML document is given in Figure 1(a). This document is a monthly report produced by a department, containing an overall description portion that gives some general information, a balance sheet variation that specifies new values concerning some items, and finally an approval portion containing some notes.
Based on this nested structure, an XML document can be represented as a graph, as illustrated in Figure 1(b). In the graph representation, white nodes represent elements, whereas gray nodes represent attributes. A node representing an element contains the element identifier (*id*). An element identifier can be the `ID` attribute value associated with the element, or can be automatically generated by the system, if no attribute of type `ID` is defined (system defined *id* are represented as $\&n$, where n is a natural number). A node representing an attribute contains its associated value. For simplicity, the data content of an element is represented as a particular attribute whose name is `content` and whose value is the element data content itself. The graph can contain edges representing the element-attribute and the element-subelement relationships, and *link edges*, representing links between elements introduced by `IDREF/URI` attributes. Edges are labeled with the tag of the destination node (i.e., an element or

```
<manager cid="154">
  <name>
    <Fname> Bob </Fname>
    <lname > Watson </lname>
  </name>
  <age> 39 </age>
  <department> R&D </department>
  <salary> 8,000 </salary>
  <category > Top Executive </category>
</manager>
```

```
<secretary cid="104", manager="154">
  <name>
    <Fname> Tom </Fname>
    <lname > Moore </lname>
  </name>
  <age> 25 </age>
  <department> R&D </department>
  <salary> 2,000 </salary>
  <level > third </level>
  <duty > manager secretary </duty>
</secretary>
```

Figure 2: Examples of $\mathcal{X}$-sec credentials

an attribute) and are represented by solid lines, whereas link edges are labeled with the name of the corresponding `IDREF/URI` attribute and are represented by dashed lines.
A document type declaration can be attached to XML documents, specifying the syntactic rules that XML documents must follow. These rules are collectively known as the *Document Type Definition* (DTD) or XML Schema. An XML source is a set of XML documents and associated DTDs/XML Schemas. Throughout the paper, we assume that an XML source $\mathcal{S}$ is given.

3.2 An Access Control Model for XML Documents

In this section we briefly review the access control model on which the proposed infrastructure relies. We first characterize how subjects are qualified in access control policies. Then, we introduce the concept of *protection object*, and the access privileges supported by the model. Finally, we introduce propagation options and we show how all the above-mentioned components are used in the specification of access control policies.

Subject. To better take into account subject profiles in the formulation of access control policies, subjects are qualified by means of *credentials*. A credential is a set of attributes concerning a subject that are relevant for security purposes. The use of credentials allows the SA to directly express relevant access control policies in terms that are closer to the organizational structure of the enterprise. For instance, by using credentials, one can simply formulate policies such as "Only programmers that are permanent staff can access documents related to the internals of the system". Each subject has one or more associated credentials that are assigned when a subject subscribes to the system. To make the task of credential specification easier, credentials with similar structures are grouped into *credential types*. Both credentials and credential types are encoded in an XML-based language called $\mathcal{X}$-sec [1]. Figures 2 gives examples of $\mathcal{X}$-sec credentials for the document in Figure 1.
Access control policies specify conditions on credentials and credential properties. These conditions (which are expressed by means of an XPath-based language [18]) implicitly identify the set of subjects to which a policy applies. Examples of conditions are: *All top executive managers*, or *All secretaries working at the R&D Department*.

Protection objects. By protection object we mean the entities to which an access control policy applies. The model provides a wide range of protection objects, in that it is possible to specify policies that apply to: *i)* all the instances of a DTD/XML Schema; *ii)* collections of documents; and *iii)* selected portions within a document(s) (i.e., an element (or a set of elements), an attribute (or a set of attributes), a link (or a set of links)). This wide range of protection objects is complemented by content-dependent access control, that is, the possibility of specifying access control policies based on document content in addition to document structure.

Privileges. Access control policies can be categorized into two groups: *authoring policies*, that allow

Table 1: Access privileges and their semantics

Type	Privilege	Meaning
Browsing	`view`	To read the values of all the attributes in a protection object, apart from attributes of type `IDREF`(s)/URI(s). The `view` privilege can also be given on selected attributes within an element
	`navigate`	To see all the links implied by attributes of type `IDREF`(s)/URI(s) contained in a protection object. The `navigate` privilege can also be given on selected attributes within an element. The view the subject has on the referred elements depends on the authorizations the subject has on them
Authoring	`delete_attr`	To remove an attribute from an element
	`insert_attr`	To add an attribute to an element
	`update_attr`	To modify an attribute value
	`insert_elemt`	To insert new elements that are direct subelements of the element on which the `insert_elemt` privilege is specified
	`delete_elemt`	To remove the subtree rooted at the element on which the `delete_elemt` privilege is specified

a subject to modify a protection object, and *browsing policies*, that allow a subject to access the information contained into a protection object. Two browsing privileges are supported: `view` and `navigate`, that allow subjects to read the information in a protection object and/or to see the relations occurring among protection objects (defined through `IDREF`(s)/`URI`(s) attributes).
Authoring privileges allow subjects to modify/delete or insert protection objects. We support five authoring privileges: three at the attribute level – `delete_attr`, `insert_attr`, and `update_attr` and two at the document/element level – `insert_elemt` and `delete_elemt`. The semantics of access privileges is given in Table 1.

Propagation options. A further distinguishing feature of our access control model is that a set of propagation options can be exploited in the specification of access control policies. Propagation options specify whether and how a policy specified on a given protection object o propagates to protection objects that are related to o by some sort of relationship. Propagation options are therefore a means to concisely express a set of security requirements. Two different types of propagation are provided: *implicit* and *explicit propagation*. Implicit propagation is always applied by default and is based on the following principles: 1) policies specified on a DTD/XML Schema automatically propagate to all DTD/XML Schema instances; 2) policies specified on a given element automatically propagate to all the attributes of the element.
In addition to implicit propagation, the SA can state, whether and how a policy specified on a given protection object propagates to lower level protection objects (wrt the document/DTD/XML Schema hierarchy). Three different options are provided for explicit propagation by means of which the SA can specify that: *i)* no propagation is enacted (NO_PROP option), that is, the policy only applies to the protection objects which appear in its specification; *ii)* the policy propagates to all direct subelements of the elements in the specification (FIRST_LEVEL option); *iii)* the policy propagates to all the direct and indirect subelements of the elements in the policy specification (CASCADE option).
Like credentials, access control policies are encoded using $\mathcal{X}$-sec. We denote with the term *Policy Base* ($\mathcal{PB}$) the XML file encoding access control policies of the source $\mathcal{S}$.[2]

Example 1 *Figure 3 shows a policy base referring to the XML document in Figure 1. According to the policies in Figure 3 secretaries, managers and accountants working in the R&D department are entitled to see, respectively, the information contained in the monthly report of their department, apart*

[2]We assume that each policy is uniquely identified by an identifier, generated by the system when the policy is specified.

```
<policy_base>
  <policy_spec pid="P1" cred_expr="//manager[@department="R&D"]" target="Department_montly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]" priv="view" prop="CASCADE"/ >
  <policy_spec pid="P2" cred_expr="//secretary[@department= "R&D"]" target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]" priv="view" prop="NO_PROP"/ >
  <policy_spec pid="P3" cred_expr="//secretary[@department="R&D"]" target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]/overall_description"
    priv="view" prop="CASCADE"/ >
  <policy_spec pid="P4" cred_expr="//secretary[@department="R&D"]" target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]/approval"
    priv="view" prop="CASCADE"/ >
  <policy_spec pid="P5" cred_expr="//accountant[@department="R&D"]" target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]/balance_sheet_variations"
    priv="view" prop="CASCADE"/ >
  <policy_spec pid="P6" cred_expr="//secretary[@department="R&D"]" target="Department_monthly_report.xml"
    path="//Department_monthly_report[department="R&D"]/overall_description"
    priv="update_attr" prop="NO_PROP"/ >
  <policy_spec pid="P7" cred_expr="//accountant[@department="R&D"]" target="Department_monthly_report.xml"
    path="Department_monthly_report[@Department="R&D"]/balance_sheet_variations"
    priv="update_attr" prop="CASCADE"/ >
  <policy_spec pid="P8" cred_expr="//accountant[@department="R&D"]" target="Department_monthly_report.xml"
    path="Department_monthly_report[@Department="R&D"]/balance_sheet_variations"
    priv="insert_elemt" prop="NO_PROP"/ >
  <policy_spec pid="P9" cred_expr="//accountant[@department="R&D"]" target="Department_monthly_report.xml"
    path="Department_monthly_report[@Department="R&D"]/balance_sheet_variations"
    priv="delete_elemt" prop="FIRST_LEVEL"/ >
  <policy_spec pid="P10" cred_expr="//company_management_director" target="Department_monthly_report.dtd"
    path="" priv="view" prop="CASCADE"/ >
  <policy_spec pid="P11" cred_expr="//manager[@department="R&D"]" target="Department_monthly_report.xml"
    path="Department_monthly_report[@Department="R&D"]/approval"
    priv="update_attr" prop="NO_PROP"/ >
</policy_base>
```

Figure 3: An example of *Policy Base*

from balance sheet variations, all information in the monthly report of their department, and only the balance sheet variations. Moreover, secretaries can also modify the overall description part, managers are entitled to update the approval part, and accountants can modify, insert new sub-elements and delete the balance sheet variations element, and delete one of its items. Finally, the company management director can see the monthly reports of all the company departments. ○

4 Distributed Updates of XML Documents

Updates to XML documents can be made according to two different modes. Under the first, which is more traditional, a subject wishing to modify an XML document sends a request to the XML document server which, on the basis of the specified authoring policies, decides whether the operation can be authorized (partially or totally) or should be denied. However, there can be cases in which this traditional approach is not adequate or it can be inefficient (since it requires an interaction with the server for each document modification). For these reasons, in this paper we propose an alternative approach to document updates which relies on encryption and digital signature techniques and supports a distributed approach to document updates. The idea is motivated by the fact that often, within an organization, XML documents are subject to pre-defined cooperative update processes (which usually take place at specific periods of times) according to which different organizational roles must modify possibly different portions of the same document. Each subject receiving the document must be able to modify all and only those portions of the document for which it has a proper authorization and then it has to pass the document to another subject for additional operations. The idea is to develop a framework supporting this update mode, able to minimize the interactions with the document server and, at the same time, guaranteeing the correct enforcement of access and authoring privileges on the document. In the following, we first give an overview of the proposed framework. Then we discuss the

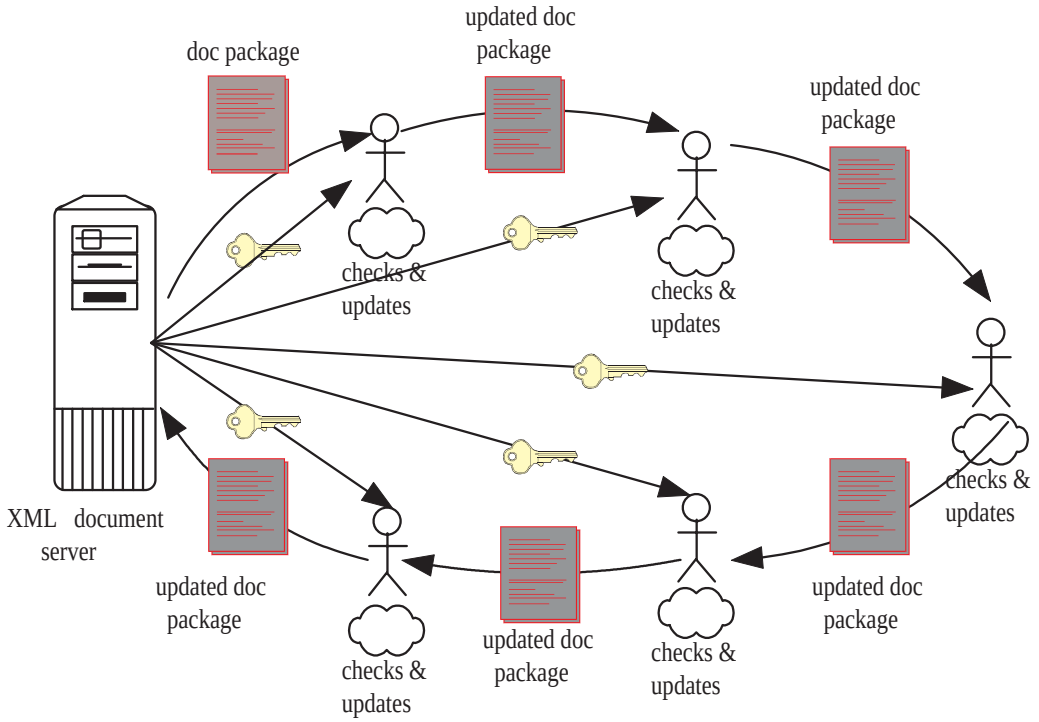

Figure 4: Overview of the update approach

assumptions on which it relies.

4.1 Overview of the update approach

The framework we have developed relies on the use of encryption techniques and consists of using different keys for encrypting different portions of the same document according to the specified access control policies. Each portion is encrypted with one and only one key. The same (encrypted) copy of the document is then sent to a subject belonging to a *collaborative group*, where by collaborative group we mean a set of subjects that may receive the document for updating or reading it. The document before returning to the server must be seen and/or modified by subjects in the collaborative group, according to a specified set of conditions, called here and in what follows *path conditions* - for example a manager must be the last subject that receives the document. Each subject in the collaborative group only receives the key(s) for the portion(s) it is enabled to see and/or modify (see Figure 4 for a general overview of the approach) and the path conditions. Such conditions together with other criteria are used by the subject in order to determine the next receiver of the document from the set of subjects in the collaborative group. The approach we propose is distributed in the sense that each subject, under specific assumptions, once receiving the encrypted document, is able to verify, without interacting with the server, whether the operations performed till that point on the document are correct (that is, they do not violate the access control policies of the source). This goal is obtained by attaching to the encrypted document additional *control information*, with the purpose of making a subject able to verify the correctness of the updates performed so far on the document, without the need of interacting with the document server. The encrypted document and the control information form the *document package*.

To support this update schema, we propose the architecture shown in Figure 5 which consists of five main components. The document is first processed by a *Parser* which, on the basis of the specified policies, analyses the document structure and groups document portions according to the policies that

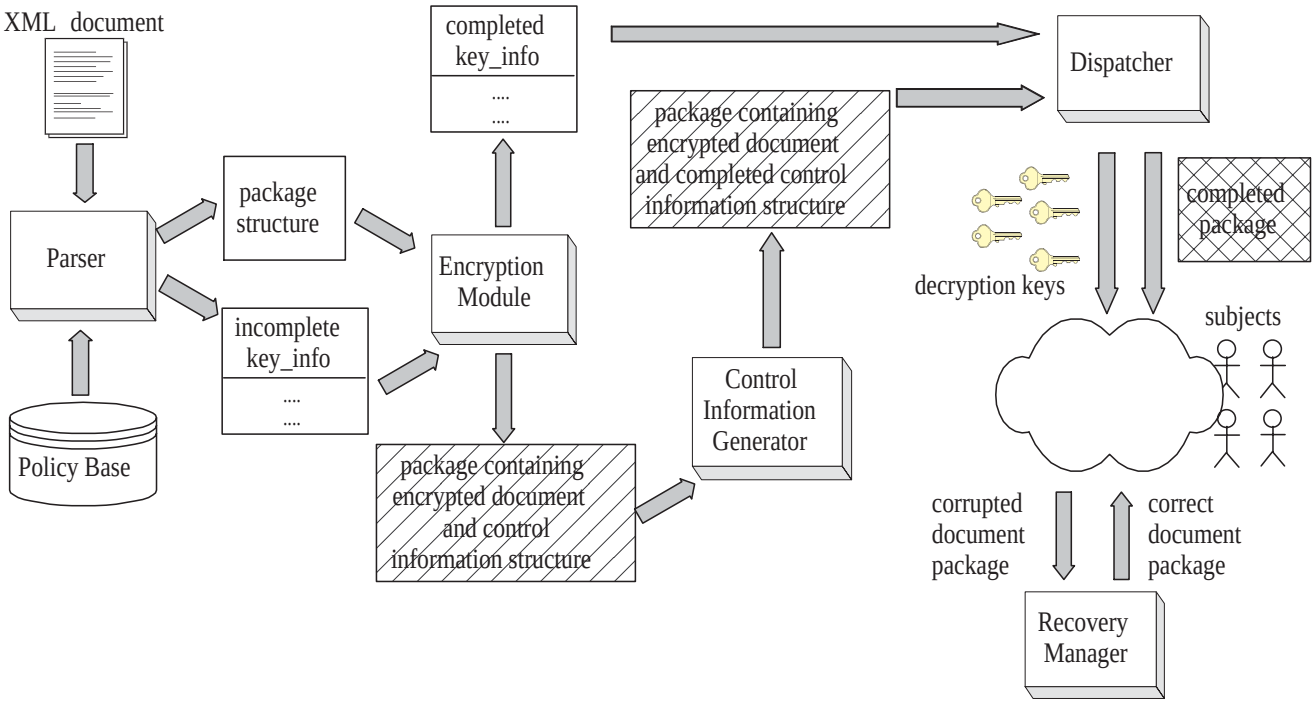

Figure 5: Distributed document updates: overall schema

apply to them. The result is the *package structure*, containing the document content already grouped according to the above-mentioned strategy together with the *control information structure*, that is incrementally updated during the package generation process. Such structure contains some *control information* , that are needed by subjects to verify update correctness. Additionally, it generates a table, named *Key_Info*, which contains information on the generated groups and their corresponding portions. Both the package structure and the *Key_Info* table are received as input by the *Encryption Module*, that generates a symmetric key for each group, and stores it in the *Key_Info* table. Then, it encrypts all the document portions with the corresponding keys. The result is the package containing both the encrypted document and the control information structure, and the updated *Key_Info* table. The package is received as input by the *Control Information Generator* that generates a set of additional control information which are stored in the control information structure. The *Dispatcher* is in charge of generating the completed document package, containing both the encrypted version of the document and the control information, and of sending it to the first chosen subject belonging to the collaborative group.
By contrast, symmetric encryption keys are separately sent to each subject. Finally, the *Recovery Manager* receives recovery requests from subjects and sends back to the subjects, at the end of a recovery procedure, the last correct version of the package. In Section 5 we describe the main components of the proposed architecture.

4.2 Assumptions

It is important to note that our approach relies on a set of assumptions that we discuss in what follows. We assume that each time a subject detects that a portion of the package is inconsistent (that is, a previous subject has operated on that portion violating the policies in $\mathcal{PB}$) the subject sends a recovery request to the document server (D_s) to obtain the last correct version of the package. This implies that we assume no collusion among the subjects and that, whenever a subject *sbj* updates a document portion

(with proper authorizations and after the execution of the document content integrity check protocol), *sbj* is sure of the integrity of that portion. Moreover, we assume that a subject sends the package to only one subject in the collaborative group, that is, we do not allow a subject to simultaneously send a package to more than one subject. Additionally, to prevent a subject from inserting old versions of document portions into a package we assume that if the receiver s_r of the package has already received the package the sender, instead of sending the package to subject s_r, sends it to D_s that updates some control information and then sends the package to subject s_r. This is done to prevent the closure of a cycle in the path followed by the document d, which would allow subject s_r to insert some portions of old versions of d in the package, without being detected by any other subject. Finally, we assume that each subject knows the public key of all the subjects belonging to the collaborative group.

5 Document Encryption

A trivial solution for generating the encryption of a document d, denoted in the following as d^e, able to support our approach is to encrypt the document at the finest granularity level, that is, to encrypt each attribute and element of the document with a different key. This solution, although very easy to implement, may require the generation and distribution of a very large number of keys. To limit the number of keys that need to be generated, we have adopted an alternative approach in which the portions of a document to which the same policies apply are encrypted with the same key. This ensures that it is always possible to deliver to each subject all and only the keys corresponding to the portions of the documents for which it has an authorization, minimizing at the same time the number of encryption keys to be generated. Here we do not go into the details of the techniques developed to support this strategy and we only give the intuition behind them. We refer the interested reader to [3] for further details.

The encryption of a document consists of two main phases: the first, called *marking phase*, marks each protection object in the source with the identifiers of the applicable policies, whereas in the second phase the document is encrypted based on the results of the first phase. Marking can apply not only to whole protection objects (i.e., attributes and/or elements), but also to the start and end tags of an element only. This possibility allows one to correctly encrypt elements containing attributes to which different policies apply. As an example consider an element e containing two attributes a_1 and a_2, and suppose that policies acp_1 and acp_2 apply to a_1, whereas acp_3 applies to a_2. Thus, the view to be returned to a subject to which both policy acp_1 and acp_2 apply is equal to element e from which all the attributes different from a_1 have been removed, whereas the view to be returned to a subject to which only acp_3 applies is the element obtained from e by removing all attributes different from a_2. Thus, in the document encryption attributes a_1 and a_2 must be encrypted with different keys, since different policies apply to them. Additionally, another key must be used to encrypt the start and end tags of e that are to be returned to all the subjects entitled to access an attribute of element e. This leads to the definition of *atomic element*, which denotes the basic portions of an XML document to which encryption can be applied.

Definition 5.1 (Atomic Element). Let d be an XML document in $\mathcal{S}$. The set $AE(d)$ of atomic elements of d is defined as follows: 1) for each element identifier *e_id* in d, and for each attribute a in *e_id*: *e_id.a*[3] $\in AE(d)$; 2) for each element identifier *e_id* in d, *e_id.tags* $\in AE(d)$. □

While an attribute corresponds to a single portion of a document d (the attribute name and its value, or only the value for data content), elements consist of two or three non-contiguous components depending on the type of the element. Empty-elements, that is, elements of the form (<*tag-name ... />*) consist of two components: the first part of the tag name (*"<tag-name"*) and its end (*"/>"*). All other elements

[3]Here and in what follows we use the dot notation to denote a component of a given structure.

consist of three components: the first part of the start-tag (*"<tag-name"*), its end (*">"*), and the end-tag (*"</tag-name>"*). This information is important because for each atomic element ae it is necessary to define where ae's components are located in the original document.

Example 2 *Example of atomic elements in the XML document in Figure 1 are:*
a) &1.*Date corresponding to: "Date* = "10/1/2002" *";*
b) &8.*content corresponding to: "*10K*";*
c) &5.*tags corresponding to: "*< *item*" " >" "< /*item* >" ○

A marking for a document d is thus a set of pairs $(ae, \mathcal{P})$, where $ae \in AE(d)$, and $\mathcal{P}$ is a set of access control policy identifiers. The encryption algorithm groups atomic elements with the same marking and generates a different encryption key for each distinct group, which is used to encrypt all the members of the group. To limit as much as possible the size of the information that circulates among subjects, the encrypted document d^e, delivered to the various subjects, consists only of the encryption of the marked atomic elements and does not contain non marked components of the document, since these components are not accessible by any subject. The set of atomic elements which are encrypted with the same key is called a *region*. We assume that each region is uniquely identified by an identifier. In the following, given an XML document d we denote with R(d) the set of identifiers of the regions of d implied by the policies in $\mathcal{PB}$. Key information are stored into table *Key_info* which records, for each region in a document, the set of atomic elements that compose the region, the identifiers of policies that apply to that region, and the corresponding encryption key.

Example 3 *Table 2 shows the content of table Key_info associated with the document in Figure 1, according to the policies in Figure 3.* ○

Table 2: Table *Key_info* for the document in Figure 1

Region	Key	Policies	Atomic elements
R1	K1	{P1, P2, P10}	{&1.tags, &1.Date, &1.Department}
R2	K2	{P1, P4, P10, P11}	{&4.tags, &4.content, &4.overall_description}
R3	K3	{P1, P3, P6, P10}	{1.tags, 1.content, 1.ID}
R4	K4	{P1, P5, P7, P8, P9, P10}	{&3.tags}
R5	K5	{P1, P5, P7, P9, P10}	{&5.tags, &7.tags, &7.content, &8.tags, &8.content, &6.tags, &9.tags, &9.content, &10.tags, &10.content}

Our system supports several methods for key delivery [2] and the SA can select the most appropriate one depending on the characteristics of the document and of the receiving subjects. Key delivery strategies supported by our system can be classified into two main categories: *online* and *offline*. In the online mode both the keys and the package are sent to the subjects by D_s (together or separately), whereas in the offline mode keys are stored in an LDAP directory [13] at D_s and subjects retrieve the necessary keys by querying the directory.

6 Generation of Control Information

After the XML document has been encrypted, the next step is to generate the control information, to be used during the document flow for verifying the correctness of the updates performed on the document. The Control Information Generator module (see Figure 5) generates this information for each region of the document and corresponding atomic elements. The generated information differs depending on the access control privileges that can be exercised on a region. For this reason we distinguish

between *modifiable* and *non-modifiable* regions. Modifiable regions are those whose contents can be modified according to the policies in $\mathcal{PB}$. A region r is thus modifiable if, among the policies that apply to r, there exists at least a policy whose access control mode is either `delete_attr`, `delete_elemt` or `update_attr`. By contrast, a *non-modifiable* region is a region whose original contents cannot be changed according to the policies in $\mathcal{PB}$. Thus, a region is non-modifiable if the access control modes of all the policies that apply to that region belongs to the set: {`view`, `navigate`, `insert_attr`, `insert_elemt`}. Note that operations corresponding to `insert_attr` and `insert_elemt` privileges alter the document content by inserting a new element and/or attribute; however, unlike the operations corresponding to the `delete_attr`, `delete_elemt`, and `update_attr` privileges, they do not modify the original region, but they can add one or more new regions to the document. Because of this characteristic, they have been inserted among the privileges related to non-modifiable regions. The sets of the identifiers of non-modifiable and modifiable regions of a document d are denoted by $NMR(d)$ and $MR(d)$, respectively. To enable a subject to verify the integrity of a document content we need different control data structures for non-modifiable and modifiable regions. In particular, the content of the structures for non-modifiable regions is statically defined by the document server when the document is delivered to the first subject in the collaborative group and it is not altered by subjects during document transmission. By contrast, the content of structures for modifiable regions changes dynamically according to the updates made on the atomic elements belonging to those regions. In what follows, we refer to policies whose access control modes are in the set *Authoring-privileges* = {`update_attr`, `delete_attr`, `delete_elemt`, `insert_attr`, `insert_elemt`} as *authoring access control policies*.
In the remainder of this section we describe in details the control information associated with document regions. Before, presenting this information, we need to introduce the notion of *authoring certificate*, which plays an important role when dealing with modifiable regions.

6.1 Authoring Certificates

Authoring certificates are used by a subject, that has modified a document portion, to prove its right to modify that document portion to the subsequent receivers of the document. Therefore, whenever a subject modifies a document (or a document portion), it has to add the proper authoring certificates to the document control structures. Certificates are generated by the sever according to the access control policies in $\mathcal{PB}$. An authoring certificate consists of: an authoring privilege p, the id of a subject that can exercise p, and the set of atomic elements on which the subject can exercise p. Authoring certificates are formally defined as follows.

Definition 6.1 (Authoring Certificate). Let d be an XML document in $\mathcal{S}$, and let $Auth\_P(d)$ be the set of authoring access control policies that apply to document d. Let Sbj be the set of identifiers of subjects authorized to access documents in $\mathcal{S}$. An authoring certificate ac is a tuple (*priv*, *sbj_id*, *prot_obj*), digitally signed by the document server, where: $priv \in$ *Authoring-privileges*; $sbj\_id \in Sbj$; *prot_obj* is a pair (*r_id*, *at_el*), such that $r\_id \in \mathrm{R}(d)$[4], and *at_el* is a set of atomic element identifiers belonging to *r_id*. □

In the following, we denote with the term *valid certificate* an authoring certificate generated according to the policies in $\mathcal{PB}$. More precisely, an authoring certificate ac=(*priv*, *sbj_id*, *prot_obj*) is valid if subject *sbj* is authorized to exercise privilege *priv* over the set of atomic elements identified by *prot_obj* according to the policies in $\mathcal{PB}$. Moreover, given a subject s, we denote with *Cert(s)* the set of valid certificates of subject s, for the documents in $\mathcal{S}$ wrt the policies in $\mathcal{PB}$. The document server takes care of sending the certificates to the subjects according to one of the following modes: *on-line*; *partially on-line*; *off-line*. The on-line mode is based on an on-demand method for the certificates generation and distribution.

[4]We recall that R(d) denotes the set of region identifiers of document d.

Table 3: Control data structures for non-modifiable regions

Name	Notation	Structure	Semantics
Control structure for non-modifiable regions	NMR_d	set of T_{NMR}, one for each non-modifiable region of d	Information used by a subject to verify integrity of non-modifiable regions of d
Control tuple for non-modifiable regions	T_{NMR}	$(r\_id, H_{r\_id}, NMAE_d)$	Information corresponding to a specific non-modifiable region r_id of d
Control structure for atomic elements	$NMAE_d$	set of T_{NMAE}, one for each non-modifiable atomic element of d	Control information associated with the atomic elements belonging to a non-modifiable region r_id of d
Control tuple for atomic elements	T_{NMAE}	*(ae_id, position, encrypted-content)*	Information corresponding to a specific atomic element ae belonging to a non-modifiable region r_id of a document d

This mode implies the generation of a certificate and its delivery only when it is strictly needed. However this mode has the drawback that the document server can become a bottle-neck. The partially on-line mode provides the generation of the authoring certificates needed by the first subject and by all the other subjects that must receive the package as specified in the path conditions generated by the document server. Also in this case the document server can become a bottle-neck, even if the number of certificate requests addressed to the server is lower. The last mode, the off-line one, provides the preventive generation and distribution of all authoring certificates. Though this strategy could be expensive, it can be executed during the periods in which the working load for the server is lower (e.g. during the night), preventing the server from becoming a bottle-neck.
The mode is chosen taking into account the average number of simultaneously active processes, because a high number of processes active at the same time can cause the server to become a bottle-neck.

Example 4 *Consider three users Ann (sbj_id="s_{10}"), Bob (sbj_id="s_{154}"), and Tom (sbj_id="s_{104}") with credentials company management director, manager, and secretary, respectively. Suppose moreover that Bob and Tom work in the R&D Department. Consider moreover the policies in Figure 3 and information in Table 2. Then: (update_attr, s_{10}, (R1, {&1.Date, &1.Department})) is not a valid certificate, since Ann is not authorized to update attributes of region R1, but only to view their content. By contrast, (update_attr, s_{104}, (R3, {1.content, 1.ID})) and (update_attr, s_{54}, (R2, {&4.content, &4.overall_descr})) are examples of valid certificates since Tom and Bob are authorized to update the content of node 1 and &4, respectively.* ○

6.2 Control data structures for document regions

The Control Information Generator module generates different data structures for non-modifiable and modifiable regions. Since non-modifiable regions cannot be altered during the document flow, the control data structure for non-modifiable regions simply contains a hash value for each non-modifiable region. This value is computed by the server before sending the package to the first subject. The idea is that, when a subject wishes to verify the integrity of a non-modifiable region it locally computes the hash value and compares it with the one stored in the data structure. If the two hash values differ, then the region has been modified by a non-authorized subject and thus the document is corrupted. The control data structure for non-modifiable regions also contains the encryption of the content and the control information associated with the atomic elements belonging to non-modifiable regions. Table 3 presents the control data structures for non-modifiable regions of a generic document d in terms of their notation, structure and semantics, whereas Table 4 explains the semantics of the components of the control data structures introduced in Table 3.

Table 4: Components of the control data structures for non-modifiable regions

Component	Semantics
r_id	identifier of a non-modifiable region of a document d
H_{r_id}	hash value computed over the encrypted (with the key of the region) atomic elements belonging to r_id
ae_id	identifier of the atomic element ae
position	it specifies where ae's components are located in the original document d and it is computed by counting, for each component of ae, the components that precede it in d. This is done by executing a pre-order depth-first left-to-right tree traversal of the graph representation of the document d and assigning a progressive integer number to each atomic element component considered during the traversal. It is important to note that when an element e has some attributes and some children elements the tree traversal assigns, an integer number to the first part of the start-tag of e, one to each attribute of e, one to the end part of the start-tag of e, one to all the atomic element components contained in the children elements, and one to the end-tag of e.
encrypted-content	it contains the encryption of the content associate with the atomic element identified by ae_id

Control information for modifiable regions is more complex than the one for non-modifiable regions because modifiable regions may change dynamically during document flow. Therefore it is not possible to compute only once the hash value for integrity verification. Such a hash value must be recomputed each time the document is modified to allow a subject to verify the correctness of the modifications performed so far on the document. By correctness of the modification we mean that if a subject has modified the document, then it must have the proper authorization. Thus the control information for modifiable regions changes dynamically during the package flow from one subject to another to reflect the operations performed on the document. To make possible the integrity verification of a region the protocol must record information about the last two subjects that have received the package and have an authoring or browsing privilege on that region. More precisely, the control data structure contains, for each region, information on the last two subjects that have *confirmed* or *modified* the region, denoted in the following as s_{last} and s_{last-1}, respectively. We say that a subject s *confirms* a modifiable region when it verifies the integrity of the region, without modifying it, and a subject s_{last}, different from s, has modified that region. In particular if a subject s performs a confirmation operation, it establishes that the updates executed by subject s_{last} are correct wrt to the policies in $\mathcal{PB}$. By contrast, a subject s $modifies$ a modifiable region, when it exercises some authoring privileges over it. Maintaining information concerning the last two subjects is necessary because a subject s, before exercising a privilege over a modifiable region, must be able to verify its integrity. To perform this control subject s must know the state of the region when s_{last-1} has sent the package to the next subject, the set of elements belonging to the region that s_{last} has modified, grouped by the privilege exercised on them, and information about the authorizations of s_{last} over that region. All these information are contained in the data structure for modifiable regions.

Before introducing the data structures for modifiable regions, we must introduce an additional information, called *cycle_path*, that the Control Information Generator inserts into the document package. This information is used by the document server when the package returns to the server for recovery. It denotes the number of cycles that the package has traversed, till that point. A cycle occurs when a package reaches a subject that has already received it before. *Cycle_path* is used to avoid that a subject, upon receiving a document, inserts in the received version of the document an old version of a document portion.

The control data structures for modifiable regions also contain the encryption of the content and the control information associated with the atomic elements belonging to modifiable regions.

The control data structures for modifiable regions of a generic document d is introduced in Table 5,

Table 5: Control data structures for modifiable regions

Name	Notation	Structure	Semantics
Control structure for modifiable regions	MR_d	set of T_{MR}, one for each modifiable region of d	Information used by a subject to verify correctness and integrity of modifiable regions
Control tuple for modifiable regions	T_{MR}	(r_id, MAE_d, $h\_c\_s_{last-1}$, $h\_c\_s_{last-1}\_dig-sig$, s_{last-1}, $h\_c\_s_{last}$, $h\_c\_s_{last}\_dig-sig$, s_{last}, h_serv_{last-1}, h_serv_{last})	Information corresponding to a specified modifiable region r_id
Control structure for atomic elements	MAE_d	set of T_{MAE}, one for each atomic element belonging to a modifiable region of d	Information used to find portions of a modifiable region r_id and to check their integrity
Control tuple for atomic elements	T_{MAE}	(ae_id, *position*, *encrypted-content* h_ae, $full$)	Information corresponding to a specific atomic element ae belonging to a modifiable region r_id of d

whereas in Table 6 we explain the semantics of the components of the control data structures introduced in Table 5.

Among the control information associated with a modifiable region, $certificates$ represents a relevant component. This component contains information about exercised privileges and involved atomic elements. More precisely, component $certificates$ contains the set of authoring certificates belonging to the subject that has modified the content of the modifiable region, and the set of atomic elements in the considered region actually modified. In particular this component is updated according to the following strategy. When a subject s wishes to exercise the `delete_elemt` privilege it has to insert into *certificates*, for each region that has some atomic elements belonging to the subtree to be deleted, its own certificate for the `delete_elemt` privilege on the subtree to be deleted. Moreover if there exists at least one atomic element already deleted by a previous subject in that region, it has to insert in that component the set of atomic elements, belonging to the subtree to be deleted, that have not been yet deleted. This is necessary because, during the execution of the subject protocol, the set of atomic elements that have been deleted by the last subject that has modified a region needs to be exactly determined. When a subject s exercises an authoring privilege operation different from `delete_elemt` over a modifiable region r_id, it has to insert in the component *certificates* of the tuple relative to r_id its own certificate for that privilege and the set of atomic elements actually modified.

Other relevant control information are represented by control hash values computed over a given region. This information includes the components: $prev\_r\_h$, $prev\_r\_dig\_h$, $last\_r\_h$, $last\_r\_dig\_h$. Those information are particularly relevant for the integrity verification process and are updated during a $confirmation$ or a $modification$ operation. A $confirmation$ is executed by a subject s by replacing information in components associated with s_{last-1} with those in components associated with s_{last} and by inserting in components associated with s_{last} information about itself, by setting component $certificates$ to $null$, and by leaving unmodified the control hash values. There is a control hash value for each atomic element ae belonging to a modifiable region of d calculated over the encryption of $ae's$ content and recorded in a component denoted as h_ae. During a confirmation the subject that confirms a modifiable region has to re-compute the hash values, contained in components h_ae, corresponding to atomic elements that were modified by subject s_{last}, to reflect the new values of the atomic elements. By contrast, a $modification$ requires a proper update of the control information by subject s. In particular, if subject s was the last subject that has previously confirmed or modified that region[5], then it executes a $modification$ by inserting in the components associated with s_{last} information regarding

[5]We recall that a document can flow back to a subject several times.

Table 6: Components of the control data structures for modifiable regions

Component	Sub-component	Semantics
r_id	-	identifier of a modifiable region of a document d
$h\_c\_s_{last-1}$	$prev\_r\_h$	hash value computed over the encrypted atomic elements belonging to r_id in the document version created by subject s_{last-1}
	$prev\_r\_dig\_h$	hash value computed over hash values (h_ae) corresponding to atomic elements in the document version created by subject s_{last-1} and belonging to r_id
	$certificates$	if subject s_{last-1} has modified region r_id, it contains authoring certificates and sets of atomic elements, it is *null* otherwise
	$cycle\_path_{last-1}$	value of *cycle_path* when s_{last-1} has operated over r_id
$h\_c\_s_{last-1}\_dig - sig$	-	hash value signed by s_{last-1}, calculated over $h\_c\_s_{last-1}$ and r_id, used to validate $h\_c\_s_{last-1}$
s_{last-1}	-	subject that verified and/or operated over r_id just before s_{last}
$h\_c\_s_{last}$	$last\_r\_h$	same meaning of $prev\_r\_h$, but referred to subject s_{last}
	$last\_r\_dig\_h$	same meaning of $prev\_r\_dig\_h$, but referred to subject s_{last}
	$certificates$	same meaning as above, but referred to subject s_{last}
	$cycle\_path_{last}$	same meaning of $cycle\_path_{last-1}$, but referred to subject s_{last}
$h\_c\_s_{last}\_dig - sig$	-	same meaning of $h\_c\_s_{last-1}\_dig - sig$, but referred to subject s_{last}
s_{last}	-	the last subject that verified and/or operated over r_id
h_serv_{last-1}	-	hash value signed by the document server calculated over $((h\_c\_s_{last-1} \,/\, \{cycle\_path_{last-1}\}) \cup \{cycle\_path, r\_id\})$ used to validate $h\_c\_s_{last-1}$ after the update of the value of *cycle_path*
h_serv_{last}	-	similar to the component previously illustrated
ae_id	-	identifier of the atomic element ae
position	-	it specifies where ae's components are located in the original document d
encrypted-content	-	it contains the encryption of the content associate with the atomic element identified by ae_id
h_ae	-	hash value computed over the encrypted content of the atomic element ae
$full$	-	hash value computed over ae_id and *cycle_path* and signed by the document server (if its value is *null* it means that the atomic element was erased by a previous subject)

the exercised privileges and involved atomic elements (component $certificates$) and by updating the control hash values; otherwise, it has to replace information in components associated with s_{last-1} with those in components associated with s_{last}, insert in components associated with s_{last} information about itself, the exercised privileges and involved atomic elements (component $certificates$), and the updated control hash values. In this case the values contained in components h_ae corresponding to the updated atomic elements are not modified by subject s.

Figure 6 illustrates a possible path followed by a document d and in particular the value of the most relevant components of the control data structure MR_d corresponding to the region with identifier r_n. Subject S_X modifies region r_n by copying information on the document server (D_s) into the components associated with s_{last-1} and by inserting in the components associated with s_{last} its identifier and its valid certificates. Subject S_Y has no access to that region and thus it does not modify it. Subject S_Z, after having verified the integrity of the region, modifies it by executing the same procedure performed by subject S_X. Finally S_T verifies the region content and confirms it, by copying the information in components corresponding to s_{last} into those corresponding to s_{last-1}, and by inserting its identifier in component s_{last}.

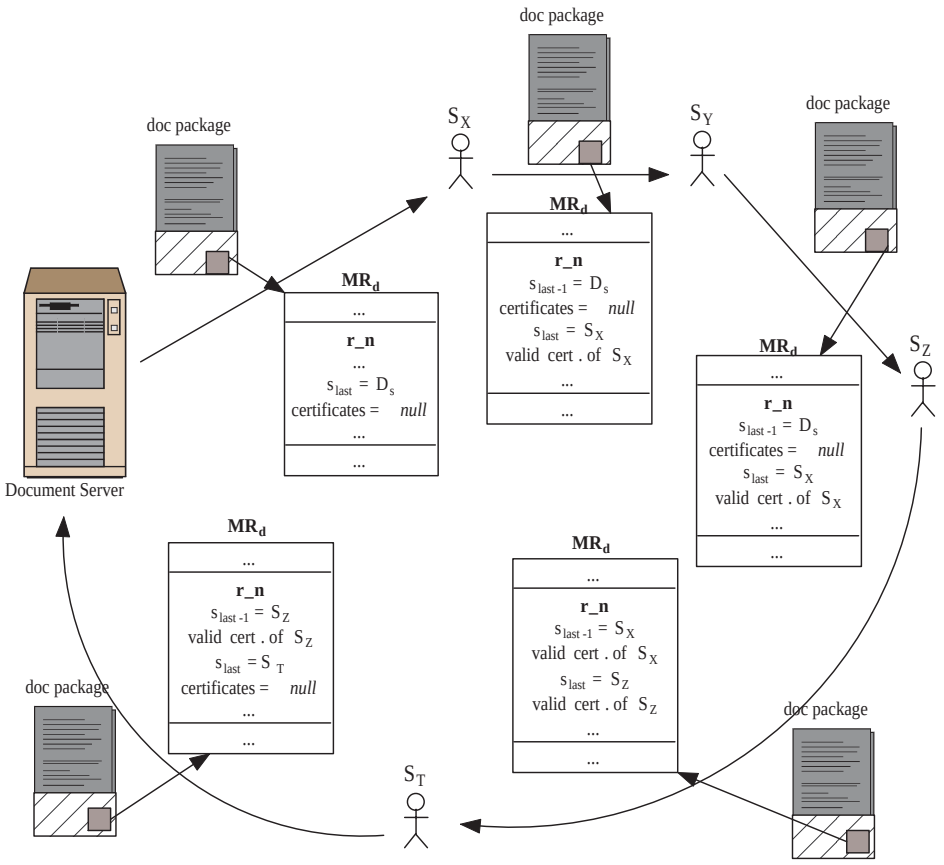

Figure 6: Modification of the structure MR_d along an hypothetical document-path

6.3 Generalized control information

The *Generalized control information* consists of some information, called *path of document d* and contained in the package, listing the set of subjects that have received the package, and of an hash value, denoted as H_{NMI}, computed over a set of information called *non-modifiable information* and signed by the document server with its private key. This non-modifiable information can be modified only by the document server and corresponds to: *cycle_path*, all control information over non-modifiable regions and atomic elements, the components (*ae_id*, *position*, *r_id*) in all tuples in the control structure for modifiable atomic elements, and the component *r_id* in all the tuples in the control structure for modifiable regions.
The path of document d is used to rebuild as much as possible the path followed by the package, when an error is detected, whereas the hash value is used to check the integrity of the information that are modifiable only by the document server.
The path of document d is incrementally updated as the package flows from one subject to another. When a subject receives the package, it inserts in the structure a tuple containing its identifier, a counter which keeps track of how many subjects have already received the package, and the identifier of the subject to which it delivers the package. Moreover, the tuple contains an hash value, signed by the subject with its private key, computed over all the tuples in the structure.

Definition 6.2 (Path of document d). Let d be an XML document. The *Path of document d* ($Path_d$) is a set of tuples (*s_id_c*, *prog*, *s_id_next*, $h_{control}$), such that: *s_id_c* is the identifier of a subject, *prog* is the position of subject *s_id_c* in the path followed by the document, and *s_id_next*

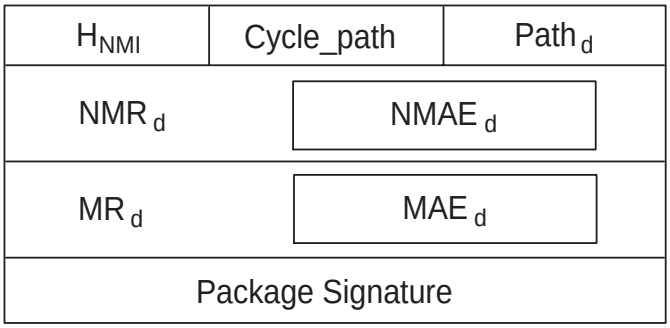

Figure 7: Graphic representation of a package

is the identifier of the subject to which s_id_c sends the package. Component $h_{control}$ is an hash value, signed by s_id_c, computed over: $\{(t.s\_id\_c, t.s\_id\_next, t.prog) | t \in Path_d \wedge t.prog \leq prog\} \cup \{cycle\_path, d\_id\}$, where d_id is the identifier of d. □

Once the Control Information Generator has initialized the above control data structures for an encrypted document d^e, the Dispatcher module generates the package to be sent to the first subject. Figure 7 provides a graphic representation of a package P_d.
After the creation of the package P_d, the Dispatcher, first of all, locally stores a copy of the package to be used during recovery operations, then signs the package with the private key of D_s and sends it to the first subject, using the SSL protocol [14].

7 Subject and Server Protocols

In this section we present the protocols executed by a subject and by the server during a distributed and collaborative update process for an XML document. In particular Section 7.1 describes how a subject performs the correctness check for a received package, how it exercises its update rights over the received document and which steps it must follow to send a package to another subject. Section 7.1 also shows an example of the subject protocol execution. Section 7.2 describes how the server manages a recovery request raised by a subject.

7.1 Subject Protocol

A subject *sbj*, according to the chosen key delivery strategy and certificate dissemination mode, obtains from D_s a set of keys, and a set of corresponding region identifiers, enabling it to decrypt the portions of the document d it is able to access. Additionally, *sbj* receives from D_s the set of its authoring certificates, generated according to the access control policies in $\mathcal{PB}$, from which it can determine which privileges can be exercised and on which atomic elements of d.
The subject protocol consists of three main steps: 1) verification of the package integrity and authentication, 2) package update, and 3) package delivery to the next subject. Section 7.1.1 presents an algorithm for performing step 1, whereas Section 7.1.2 summarizes the strategies we have devised for steps 2 and 3. Finally Section 7.1.3 presents a possible update scenario.

7.1.1 Package integrity and authentication

A subject *sbj*, upon receiving a package, verifies its authenticity and the integrity of the corresponding control information and of the document content it is authorized to access. If no errors occur during this phase, *sbj* decrypts all the portions of d it is able to access with the received decryption keys.

Algorithm 1 *An algorithm for verifying package integrity and authenticity*

INPUT: The package P_d coming from a subject s_n
The receiver subject *sbj*
OUTPUT: A package P_d containing both correct control structures and correct document content
METHOD:

1. Extract from P_d the Package Signature $S_{s_n}(P_d)$
2. Let H_{P_d} be an hash value calculated by *sbj* over P_d
3. **If** $(D_{KU_{s_n}}[S_{s_n}(P_d)] \neq H_{P_d})$: reject the package
   else: (a) Let *h_nmi* be the hash value calculated by *sbj* over non-modifiable information in the package P_d
   Check the structure $Path_d$
   If $(h\_nmi \neq D_{KU_{D_s}}[H_{NMI}] \vee$ error in $Path_d)$:
   Send a recovery request to D_s
   Receive from D_s another package in which the generalized control information are correct
   endif
   error := 0
   (b) Let Reg = $\{r_1, \ldots, r_n\}$ be the set of region identifiers belonging to $NMR(d)$ for which *sbj* has an authorization
   For each r ∈ Reg:
   Let h_reg be the hash value calculated by *sbj* over r's atomic elements
   Let NMR_d[r].H_{r_id} be the H_{r_id} component of the tuple belonging to NMR_d, with r_id = r
   If (h_reg ≠ NMR_d[r].H_{r_id})
   error := 1
   Send a recovery request to D_s
   Receive from D_s another package containing correct control structures and correct document content
   break
   (c) **If** (¬ error):
   Let RM = $\{r_1, \ldots, r_m\}$ be the set of region identifiers belonging to $MR(d)$ for which *sbj* has an authorization
   For each rm ∈ RM:
   (d) **If** (MR_d[rm].$h_c_s_{last}$_*dig-sig*.certificates = *null*): Control instructions for confirmed regions
   (e) **else** Control instructions for modified regions
   endif

Figure 8: An algorithm for verifying package integrity and authenticity

Then, it can exercise over the decrypted portions all the privileges derived by its authoring certificates. By contrast, if an error is detected, *sbj* sends a recovery request to D_s to obtain the last correct version of the package. The steps executed to verify package integrity are presented in the algorithm in Figure 8, whereas Figure 9 gives a graphic representation of the main steps of the algorithm.
The overall strategy of the algorithm is to verify the authenticity and the integrity of the received package and of the associated data structures by locally calculating some hash values over the package, by decrypting the hash values contained in the document package P_d with the public key of the subjects that have already received the package and by verifying the correspondence between the locally calculated hash values and the decrypted ones. If these values are different, the package is considered incorrect and thus is not accepted. A recovery request is raised by the protocol to obtain another package containing the last correct version of the portions detected as corrupted.
The algorithm starts (step 3) with the integrity verification of the package by extracting the package signature and comparing it with an hash value locally computed over the same elements. If the two values are different the algorithm requests the sender subject to send the package again. Otherwise, the algorithm verifies the correctness of the content of non-modifiable information through the comparison of

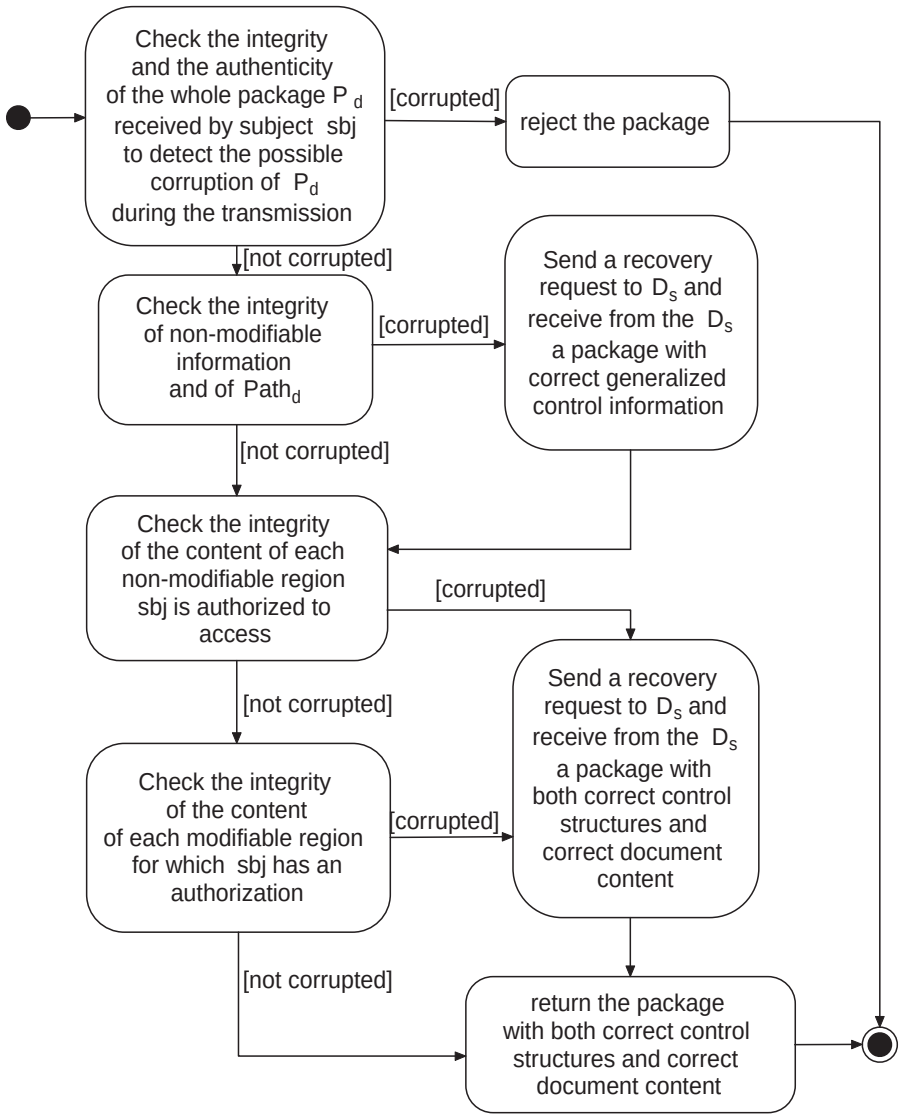

Figure 9: Main steps of the package integrity and authenticity algorithm

an hash value (h_nmi) computed over these information with the one stored in the package (step 3.a).[6] The check operated over $Path_d$ (step 3.a) consists of the verification of the correct correspondence between the subject identifier in a tuple (s_id_next) and the corresponding one (s_id_c) in the next tuple in the structure. Moreover, all hash values contained in the tuples must be correct, that is, each component $h_{control}$, decrypted using the public key of the subject specified in the component s_id_c, must be equal to that calculated by the algorithm over the information specified in Definition 6.2. Finally, the last subject specified in $Path_d$ must be equal to the sender of the received package, to prevent a subject from intentionally not inserting itself in $Path_d$, with the purpose of inserting old versions of document portions when it receives again the package. The correctness of this structure is important during the execution of the recovery operations, because it is used to rebuild as much as possible the path followed by package P_d (for more details see Section 7.2). If an error occurs the algorithm sends a recovery request to D_s to receive a correct version of the package.
Then, the algorithm verifies the integrity of the atomic elements belonging to non-modifiable regions for

[6] We denote with D and E the operations of decryption and encryption of the object in square brackets, respectively. Moreover, with KU_s and KR_s we denote the public and the private key of a subject s, respectively.

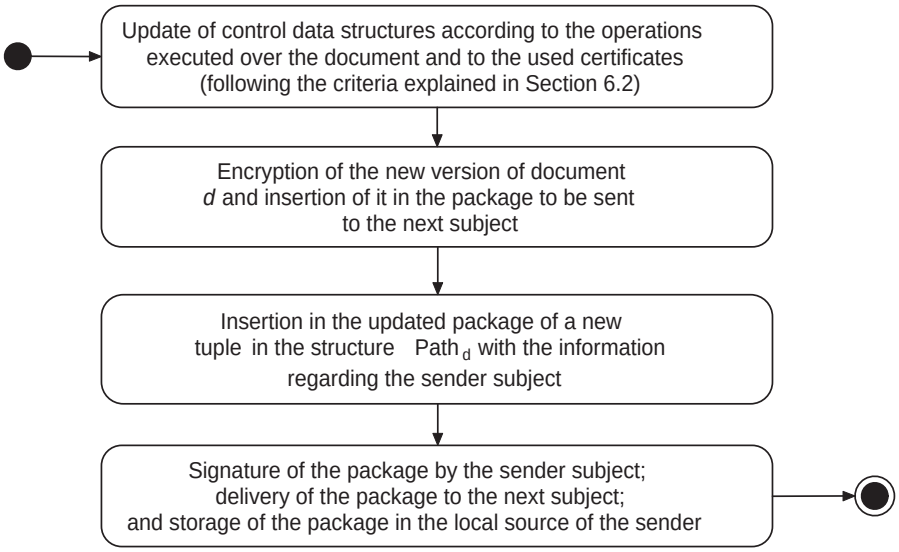

Figure 10: Update and delivery processes

which *sbj* has an authorization (step 3.b) using the same strategy presented above. If an error occurs the algorithm sends a recovery request to D_s. If no error is detected, the algorithm verifies the integrity of atomic elements belonging to modifiable regions (step 3.c) by verifying, for each modifiable region r such that *sbj* has an authorization on it, the authenticity and the integrity of the information inserted by the last two subjects s_{last-1} and s_{last}. If the region is confirmed (step 3.d), its content must be correct with respect to the hash value stored in component $last\_r\_h$ and with respect to that in component $prev\_r\_h$. Moreover the hash values in components h_ae, referring to the atomic elements belonging to that region, must be correct with respect to the hash values stored in $last\_r\_dig\_h$ and $prev\_r\_dig\_h$. If a region is modified (step 3.e), then by using the hash value in component $prev\_r\_dig\_h$ and the one in the components h_ae of the atomic elements belonging to that region, the algorithm can build the state of the region before the modification and then verify that the operations performed over it are correct with respect to that state. This is executed by checking that the atomic elements which have not been declared as modified (in the component $certificates$) have maintained their previous values and verifying that the updated atomic elements were modified according to the declared privileges (those contained in the certificates inserted in component $certificates$).
In each of the above cases if an error is detected, a recovery request is sent to D_s to obtain a correct package. Finally, Algorithm 1 returns a package P_d, containing both correct control structures and correct document content. A copy of this package is stored for recovery purposes (together with the id of the sender subject, the corresponding value of *prog* component in $Path_d$ and the current value of *cycle_path*) by the receiver in its local store.

7.1.2 Package update and delivery

After a subject s has executed Algorithm 1 on a document package P_d, it can read or modify the regions of document d for which it has some authorizations. Then s must update the data structures to keep track of the operations it has performed on document d and send the updated package to the next subject in the collaborative group that satisfies the received path conditions. The operations performed in these steps are graphically summarized in Figure 10.
A subject can locally exercise all privileges, authorized by its certificates, apart from privileges `insert_attr` and `insert_elemt`. These privileges must be executed by D_s, because the corresponding operations could generate new regions. In this case a subject s must send the package, and new

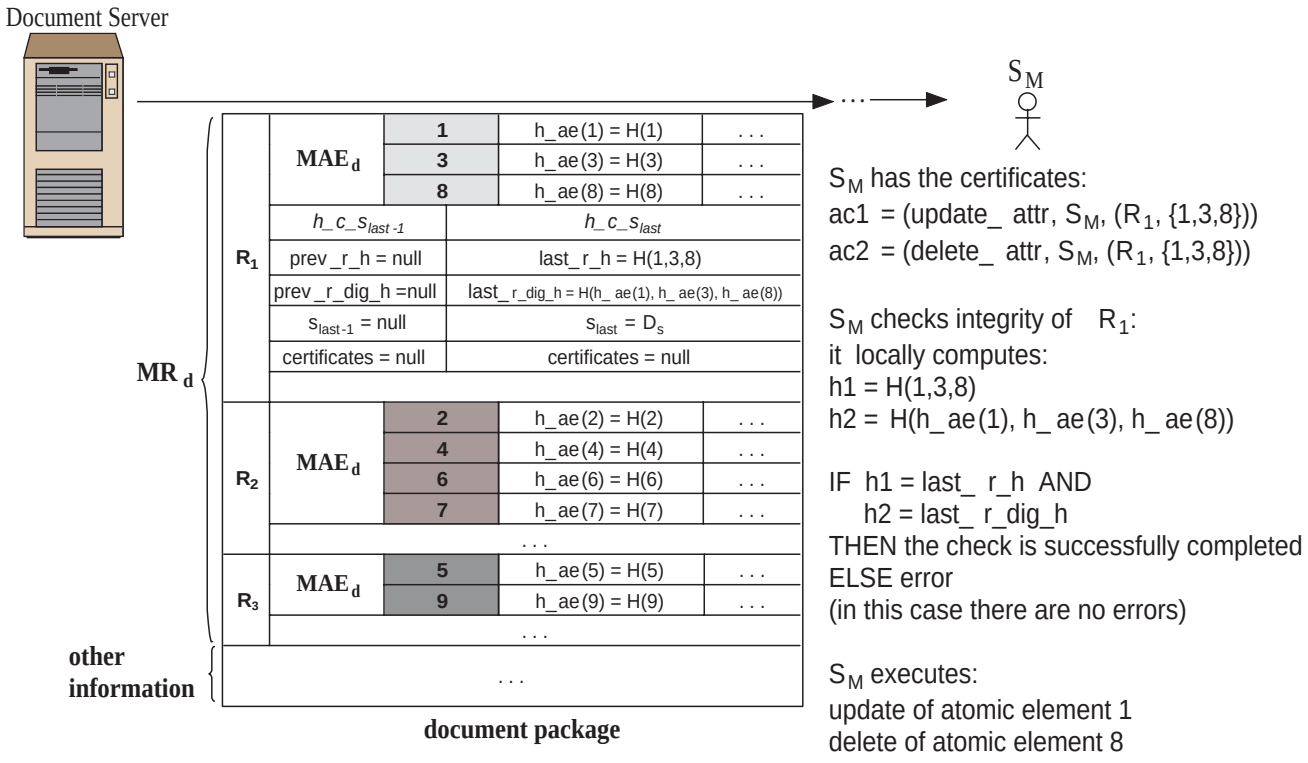

Figure 11: Update flow of a modifiable region (Step 1)

portions it wishes to insert into the document d to D_s, that takes care of executing these operations and sending the updated package to the specified receiver. In particular, new inserted atomic elements are marked according to the policies in $\mathcal{PB}$ by D_s and then inserted in the corresponding new or old region according to their marking. For each newly created region a new entry in the table *Key_info* is inserted, whereas control information corresponding to each old region to which new atomic elements have been added is updated to reflect the new content of the region.
If the receiver s_r of the package is already present in the structure $Path_d$, the sender instead of sending the package to subject s_r, sends it to the D_s that updates the value of *cycle_path* and of the other involved structures, appends the content of the structure $Path_d$ in a local store, re-initializes the $Path_d$ structure and then sends the package to subject s_r. We do such steps to prevent the closure of a cycle in the path of document d. Such an event would allow subject s_r to replace some portions of d in the package with old ones, without being detected by any other subject.

7.1.3 An Illustrative Example

In this section we discuss the example reported in Figures 11, 12, and 13. We focus on the operations executed by some subjects over the atomic elements and control data structures belonging to region R1. In particular that region is composed by three atomic elements with identifiers 1, 3, and 8, respectively. At the beginning of the process the region is covered only by the hash values computed by D_s, therefore the components in the MR_d control data structure associated with s_{last-1} are empty (value *null*). When S_M receives the package, region R1 was never modified. This subject possesses two certificates for that region: one containing the `update_attr` privilege and the other containing the `delete_attr` privilege. First of all the subject checks the integrity of that region by computing two hash values, one over the encrypted contents of the atomic elements belonging to region R1 (`h1`), and the other over the *h_ae*

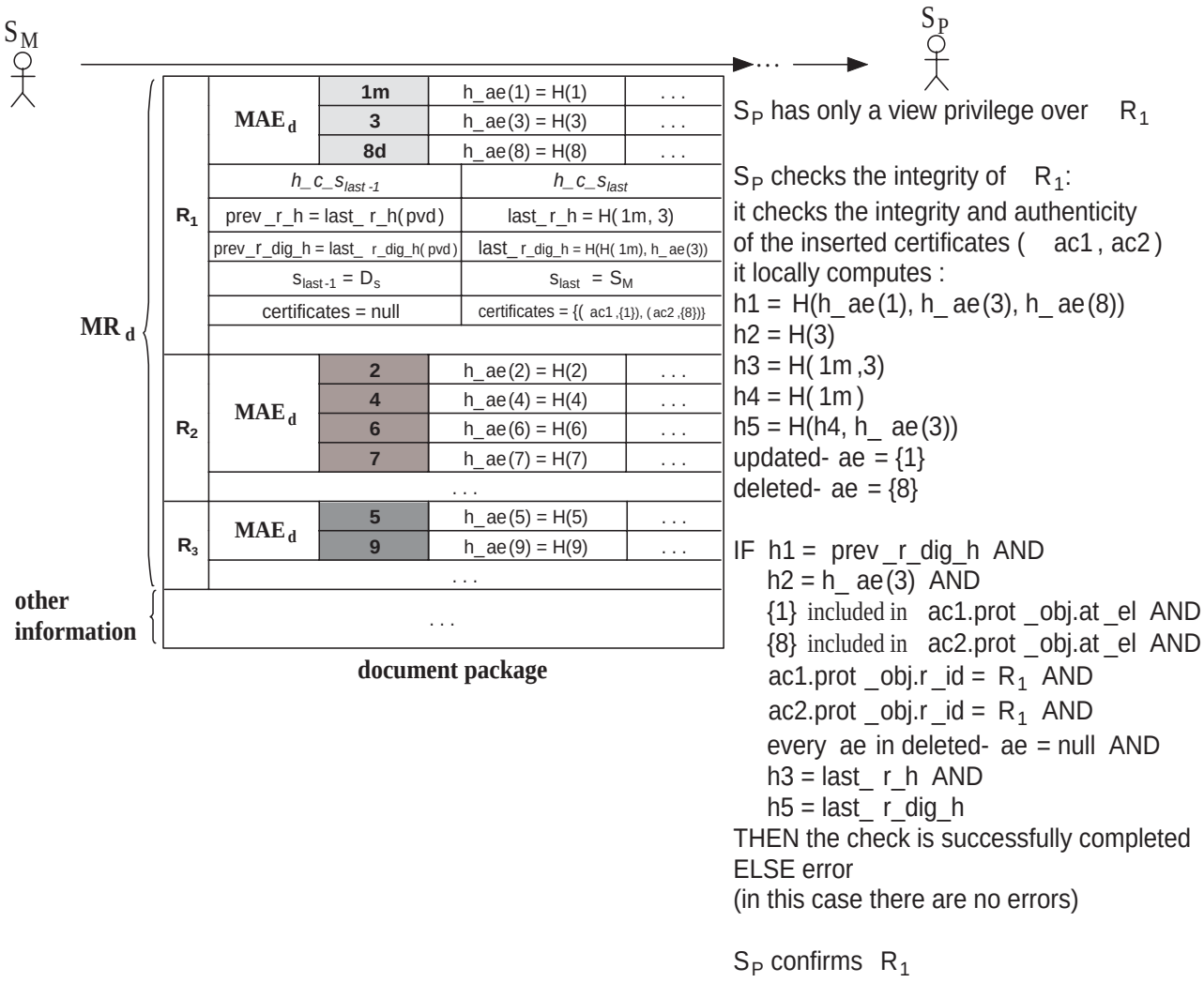

Figure 12: Update flow of a modifiable region (Step 2)

components associated with those atomic elements (`h2`), and then checks that `h1` and `h2` match the hash values, $last\_r\_h$ and $last\_r\_dig\_h$, associated with R1 and stored in the MR_d control data structure. Finally, S_M updates the atomic element 1, deletes the atomic element 8, updates the corresponding control information in MR_d and sends the updated package to another subject. After a certain number of subjects the package reaches S_P, that can only view the content of region R1. S_P checks the integrity of R1 by executing the following steps:

- It checks the integrity and authenticity of the inserted certificates (`ac1`, `ac2`). Then it determines the set of updated atomic elements belonging to region R1, corresponding to 1 and the set of deleted atomic elements belonging to R1 corresponding to 8. Such sets of atomic elements are obtained by using the information contained in the component `certificates` of the tuple associated with region R1 belonging to the MR_d control data structure. Obviously if a certificate with privilege different from `update_attr`, `delete_attr` or `delete_elemt` is found in the `certificates` component, an error is raised.

- It locally computes some hash values. Hash value `h1` is computed over the h_ae components associated with the atomic elements belonging to R1 not yet deleted or declared as deleted in

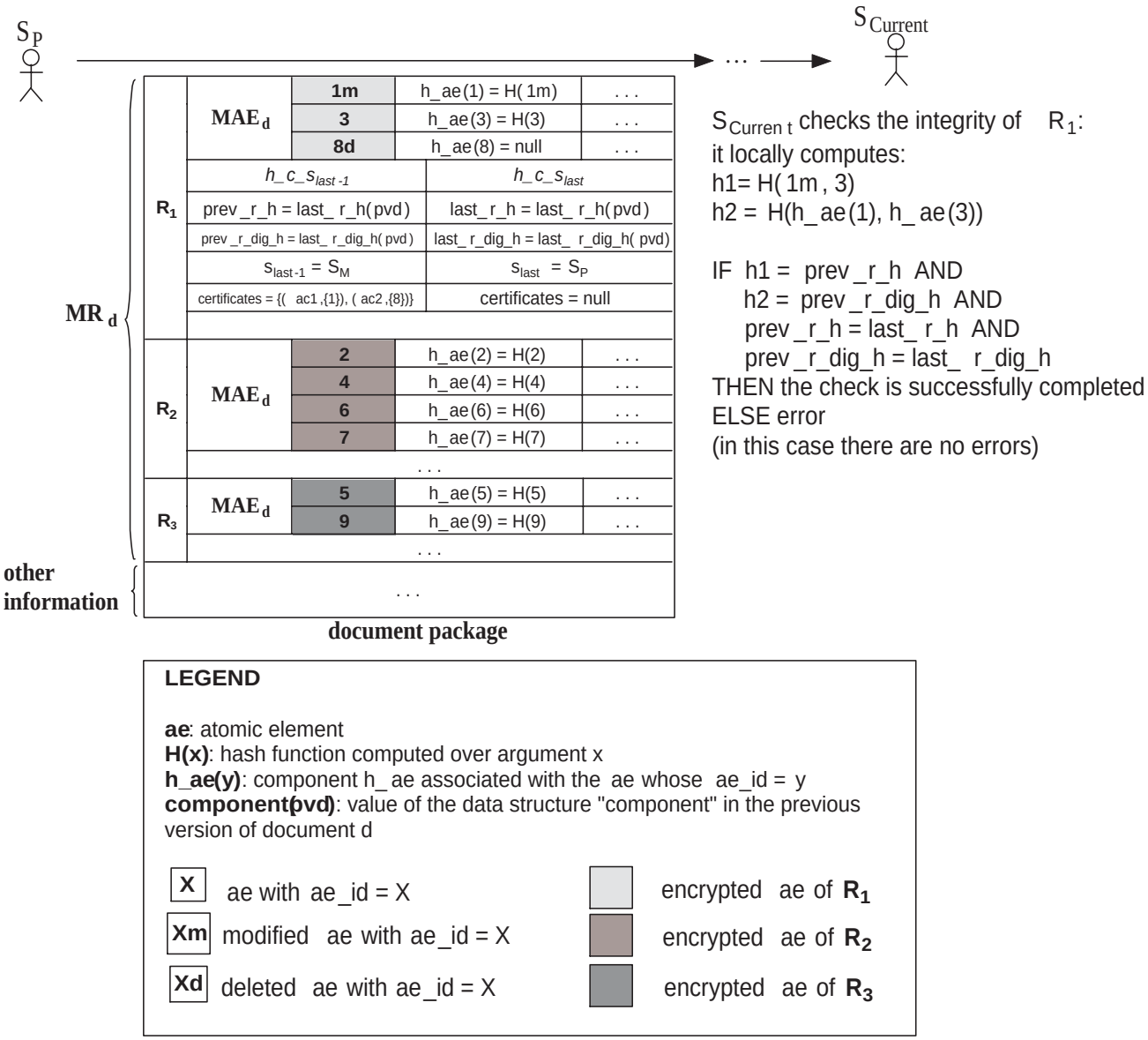

Figure 13: Update flow of a modifiable region (Step 3)

component `certificates` associated with s_{last}; `h2` is computed over the encrypted content of the atomic element 3, because it has not yet been deleted and it is not declared as modified in component `certificates` associated with s_{last}; `h3` is computed over the encrypted content of all the atomic elements of R1 not yet deleted; `h4` is computed over the encrypted content of the updated atomic element 1; `h5` is computed over the component h_ae associated with atomic element 3 and `h4`, that are the hash values computed over the atomic elements of R1 not yet deleted.

- It compares hash value `h1` with the component $prev\_r\_dig\_h$ in MR_d. The correspondence between these two assures that no subject has modified the values associated with the h_ae components.
- It compares hash value `h2` with the component h_ae associated with the atomic element 3. Their correspondence and the satisfaction of the previous check assure that the content of the atomic

element 3 was not modified.

- It checks that the atomic elements declared as updated or deleted belong to the set of atomic elements contained in the inserted certificates (`ac1`, `ac2`), and that the region specified in those certificates is R1, and finally that each atomic element declared as deleted has a *null* value in its *encrypted-content* component. These conditions assure that S_M possesses the proper rights to update and delete atomic elements 1 and 8, respectively, and that atomic element 8 was really deleted.

- It compares hash value `h3` with the component $last\_r\_h$ in MR_d. The correspondence between them assures that no subject had modified the content associated with the atomic elements of R1 not yet deleted, after the modification executed by S_M.

- It compares hash value `h5` with the component $last\_r\_dig\_h$ in MR_d. The correspondence between them assures that components $last\_r\_dig\_h$ and $last\_r\_h$ cover the same content associated with the atomic elements of R1 not yet deleted, in an indirect mode, the former one, and in a direct mode, the latter one.

Before sending the package to the next subject, S_P confirms region R1 by: updating the value of the components h_ae corresponding to its modified atomic elements 1 and 8; moving the information associated with the components in MR_d associated with s_{last} into those associated with s_{last-1}; and inserting into the components associated with s_{last} the hash values that cover directly ($last\_r\_h$) and indirectly ($last\_r\_dig\_h$) the current content of the atomic elements, together with the identifier of S_P. Finally, subject $S_{Current}$ receives the package and checks its integrity. Since region R1 is a confirmed one, $S_{Current}$ must locally compute a hash value (`h1`) over the encrypted content of the atomic elements of R1 not yet deleted and another one over the corresponding h_ae components. Then, $S_{Current}$ checks that those hash values match both the ones stored in the components associated with s_{last} in the MR_d control data structure and the other ones stored in the components associated with s_{last-1}. These conditions assure that no subject after S_P has modified the region content and also that S_P itself did not modify the content of the region.

7.2 Server Protocol

When a subject detects that a package is compromised, it requires from the server a correct version of the document package. In particular, there are two types of recovery requests that a subject may send to the document server.
The first type of recovery request is sent when an error to the generalized control information has been detected. In this case the server protocol checks the structure $Path_d$, in the received package attached to the request, and saves the portion of it that is correct. Then, it sends each subject in the collaborative group a message by which it requires the structure $Path_d$ of the package (if any) they have stored, having the value of *cycle_path* equal to the current value stored by the server. By checking (following the criteria explained in Section 7.1.1) the received structures $Path_d$, and by matching them with the correct portion of that one in the corrupted package, the server rebuilds as much as possible the path (denoted in the following as *rebuilt-path*) followed by the package. This is executed by considering the structures $Path_d$, obtained from the subjects, in ascending order with respect to the number of subjects they contain. The path saved by the server is matched with the first structure $Path_d$. Every match generates a partially *rebuilt-path* which is used in the subsequent matches. Such path consists of the subjects which appear in both paths and of the subjects contained in the path with the highest number of subjects. The rebuilding process terminates when there are no more paths to evaluate or when two paths have, in a tuple, equal value for component *prog* and different values in one of the other components. Now it is possible to find the last correct version of the package by requiring the

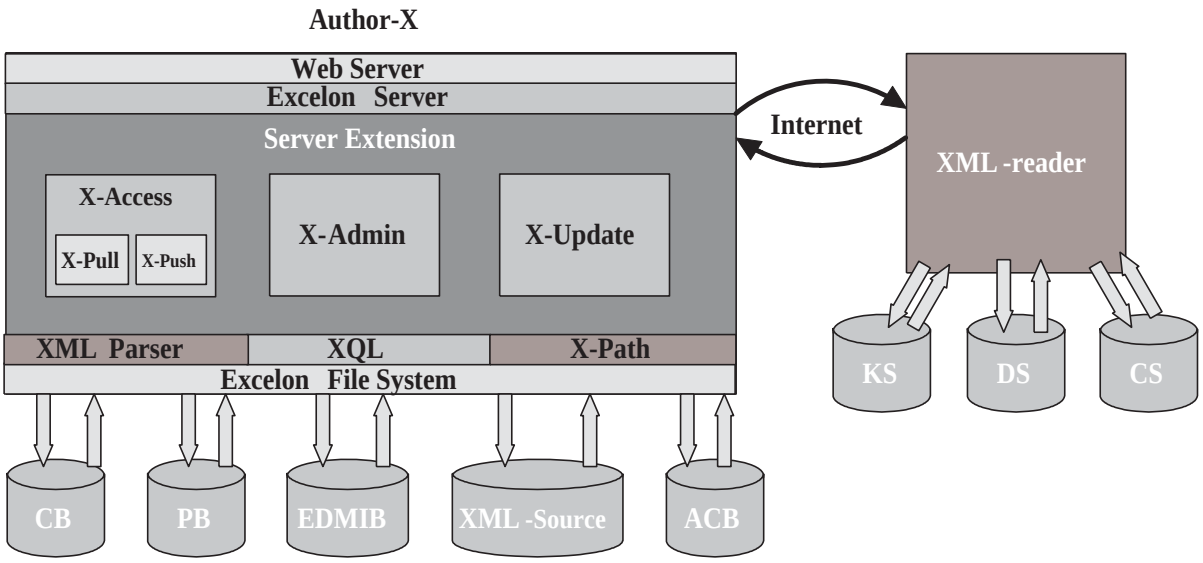

Figure 14: Author-$\mathcal{X}$ architecture

subjects listed in the *rebuilt-path* (beginning from the one with the highest value in the component *prog* and stopping the requesting process when a correct version is found) their last sent and stored package having the value of *cycle_path* equal to that one stored by the server.
Finally, the server appends the *rebuilt-path* to the path (denoted as *global-path*) already saved in a local store, re-initialize the Path_d structure, updates the value of *cycle_path* as well as the data structures of the received package and sends this updated package to the subject that has required the recovery.
The second type of recovery request is sent when an error in the content of a region is discovered. In this case if the error affects only non-modifiable regions, the server can directly solve the problem without the help of other subjects, because it has stored the original value of this information in its local repository. It can thus replace the corrupted information with the saved ones. Otherwise, if the $Path_d$ structure has been compromised the server rebuilds the last portion of the path followed by the package, by using the strategies explained above. In any case the server, by using the information in table *Key_Info*, selects the subjects from which it requires a package containing the last correct version of the corrupted modifiable regions. The request process is executed starting from the selected subject with the highest value in component *prog* in the *global-path* obtained appending the *rebuilt-path* to the one already saved and stopping the process when a correct version of the region is found or the selected subject has a value in component *prog* less than the one of the subject that has received the last correct version of the region found and saved by the server during a previous recovery session. If a correct version of the region is not found through this process the initial version of the region is inserted into the package. Finally, the server re-initialize the Path_d structure, updates the value of *cycle_path* as well as the data structures of the received package and sends this updated package to the subject that has required the recovery.

8 System Implementation

The protocols described in the previous sections are currently being implemented in the framework of the Author-$\mathcal{X}$ system [2]. Author-$\mathcal{X}$ is a java-based system supporting selective, secure and distributed dissemination of XML documents. Its architecture, presented in Figure 14, is based on the client-server paradigm. The server system, built on top of the eXcelon XML data server [5], manages all information required for controlling access to documents. In particular the server database is organized in terms of

five repositories: the *Policy Base* ($\mathcal{PB}$), the *Credential Base* ($\mathcal{CB}$), the *Encrypted Document and Management Information Base* ($\mathcal{EDMIB}$), *the XML Source*, and the *Authoring Certificate Base* ($\mathcal{ACB}$). In addition to the database, the server includes the following main components: $\mathcal{X}$-*Admin*, $\mathcal{X}$-*Access*, and $\mathcal{X}$-*Update*. We briefly describe the first two, and then focus on the last one which implements the protocols defined in this paper. The $\mathcal{X}$-*Admin* component provides functions supporting administrative operations, such as for instance specifying or modifying policies, or updating credentials. The $\mathcal{X}$-*Access* component consists of two subcomponents: $\mathcal{X}$-Pull and $\mathcal{X}$-Push. The former component supports the selective distribution of the XML documents, stored in the *XML Source* repository according to the policies in $\mathcal{PB}$ using the traditional user-on-demand paradigm. By contrast, the latter component is in charge of supporting document broadcast to user groups. As such, it supports a push-based distribution of the XML documents stored at the server site. In order to support group distribution of the same document and yet to enforce selective access to different components of the document, document components are encrypted by using different keys that are generated according to the stated access control policies. Each subject in the group then receives only the keys for decrypting the document components it can access. The last component of the Author-$\mathcal{X}$ server is $\mathcal{X}$-*Update*, that manages the collaborative and distributed update process described in this paper. It generates and updates all control information associated with an XML document. It also generates all certificates associated with a document according to the policies in $\mathcal{PB}$, and performs the initial steps for the creation and delivery of packages. A package contains the encrypted[7] version of an XML document and the associated control information. Finally, the $\mathcal{X}$-*Update* module manages the recovery process. In particular, $\mathcal{X}$-*Update* uses different XML encodings for modifiable and non-modifiable atomic elements of the document, generating an XML structure called *MAE*, for *modifiable* atomic elements, and another XML structure called *NMAE* for *non modifiable* ones. Figure 15 shows an example of how $\mathcal{X}$-*Update* encodes some atomic elements of the XML document reported in Figure 1, before and after the encryption and control information generation processes, according to the specification of the structures T_{NMAE} and T_{MAE} presented in Tables 3 and 5, respectively.

The information, common to both structures and visible in the unencrypted encoding in Figure 15(a) are: 1) an atomic element identifier (corresponding to *ae_id*), that univocally identifies a specific modifiable/non modifiable atomic element within the document; 2) a position information (corresponding to *position*) that, in the case of an attribute indicates the position of that attribute within the document (P1), whereas in the case of an element indicates the position of its start-tag (P1), of the end of the start-tag (P2), and of the end-tag (P3); 3) the atomic element itself (corresponding to the atomic element content in the original XML document), that is, the tag associated with an element or the name and the corresponding value associated with an attribute. The *MAE* structure also contains two additional information: *h_ae* and *full*, denoting respectively an hash value and a digital signature (cfr. Table 6). In particular note that the position information and the atomic element content are separately encrypted. The reason is that different hash values are computed over these information. The position information, that can never be modified, must be always covered by the H_{NMI} hash value, whereas the atomic element content is covered by the hash value (stored in component H_{r_id}, if it is non-modifiable, or in *prev_r_h* and *last_r_h*, if it is modifiable) computed over all the atomic elements of its region and also by the *h_ae* hash value, only if it is modifiable. It is important to note that the encrypted atomic element content (corresponding to the *encrypted-content* component) is contained within a *MAE*/*NMAE* XML structure, together with its corresponding control information, since this makes easier building the correct view at the client side, as explained in what follows. Figure 16 shows an example of authoring certificate, encoded in XML and compliant with the W3C recommendation for the generation of digital signatures [19].

In the first portion of the certificate there is the authentication information, whereas the Object element

[7]Encryption is compliant with W3C recommendation [17].

Figure 15: Structures MAE and NMAE (a) before and (b) after the encryption and control information generation processes

```
<Signature IdDoc="1" IdSig="1">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>iie89RrlwmQ6Q/JiOIWduLvoovQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>LD1MKic3GHah2CXKzBguBOIKMiCnJmu2jK+7/gXMmHLMBeQxTlUYzZJuHRGyBNc+LtSqdW
                  dMHrHia5kdbLofL0EcNXM4uVqd2OwzD1jy3nvel9fF/0+aM3p8ZHz8tfmdu1ToMXj9EoORuz84aB5
                  pXypNqbja2tQayO1aE6sH3qg=
  </SignatureValue>
  <Object>
    <Privilege>update_attr</Privilge>
    <Subject>104</Subject>
    <ProtObj>
      <Region>3</Region>
      <AtomicElements>
        <AE>5</AE>
        <AE>6</AE>
      </AtomicElements>
    </ProtObj>
  </Object>
</Signature>
```

Figure 16: An XML authoring certificate

contains all the required components of the certificate, according to Definition 6.1.
The Author-$\mathcal{X}$ client side, called *XML-reader*, supports several functions for issuing queries and receiving query replies, and for the enforcement of the subject protocol presented in this paper. We discuss the last function in more details, since it is more relevant for the discussion in the present paper, by describing how the *XML-reader* generates the correct view of a received XML document using some information stored in its repositories. As shown in Figure 14, the *XML-reader* manages three repositories: *DocumentStore* ($\mathcal{DS}$), *KeyStore* ($\mathcal{KS}$), and *CertificateStore* ($\mathcal{CS}$). The first one records the XML document views obtained as answer to a pull request, or during a server push dissemination of XML documents, or in case of a collaborative and distributed update process. The second repository contains all keys used to decrypt the portions of the XML documents for which the subject has the proper rights. Each key has associated a document and a region identifier to identify which portions of a document can be decrypted using that key. The third repository stores all authoring certificates received from the server. To generate the correct view of an encrypted document d^e, received by the server or by another subject, the *XML-reader* retrieves in $\mathcal{KS}$ the keys associated with d^e and using the region identifier associated with each key decrypts all the atomic elements in d^e belonging to the corresponding region. Finally, to generate a well-formed view of d^e, according to the structure of the original document, the *XML-reader* uses the position information associated with each decrypted atomic element. In particular, that information is used to determine the correct place in the generated view, where to insert each decrypted atomic element.

9 Complexity Analysis of the Proposed Approach

In this section we present a complexity/cost analysis for the most relevant operations executed by the protocols on which our approach relies and we compare them with the equivalent operations executed in a conventional centralized system. In particular we evaluate the following complexity/cost measures:

1. communication cost;

2. size of exchanged information;

3. number of certificates generated for a document d;

4. execution time of the integrity check protocol and distributed view generation/decryption vs execution time of the centralized view generation/encryption and distributed decryption.

The complexity/cost is expressed in terms of several parameters that are reported in Table 7. According to the context such parameters will be interpreted as sets of elements or as the cardinalities corresponding to these sets of elements.

Table 7: Parameters involved in the complexity/cost analysis

Parameter	Semantics
$\mathcal{P}$	the set of policies that apply to a specified XML document d
AE(d)	the set of atomic elements associated with d
R(d)	the set of regions associated with d
$\mathcal{CG}$	the collaborative group
$Path_d$	the path of document d
$\mathcal{N}_i$	number of interactions, that is, the number of sessions per document opened by the subjects in a centralized approach (a subject can open more than one non-simultaneous session per document) or the number of subjects reached by the package in a distributed and cooperative approach (a subject reached more than one time by the same package is counted exactly as many times as it is reached)

Before evaluating the above-mentioned complexity/cost measures we believe that an explanation of what a *conventional centralized system* means is required to better understand the comparison between our distributed approach and a centralized one.
A conventional centralized system, that manages collaborative updates of an XML document, has to generate the document view for each subject involved in the collaborative process, encrypt this view with a session key (different for each subject involved) before sending the view to the proper subject s, execute the access requests received by s and decrypt the updated portions attached to the access requests (in case of update access requests). Note that access requests can be fully/partially executed or also denied, according to the identity of the requester and to the access control policies in $\mathcal{PB}$.
A centralized collaborative update process is realized as follows: the document server generates the view for the first chosen subject s_1 and a session key used to encrypt that view. Then, the system sends this encrypted information to s_1. s_1 uses the attached simmetric key to decrypt the received encrypted information and then sends some access requests to the system to update the view content. The system evaluates each received access request and only when the access request is evaluated as correct wrt the identity of s_1 and the policies in $\mathcal{PB}$, the system updates the document content and sends s_1 a positive response. A negative response is sent back otherwise. A subject can send more than one access request to the system that individually evaluates them. Subject s_1 at the end of its job, chooses the next subject to be involved in the collaborative update process and sends the identifier of that subject to the document server. The document server repeats the same steps followed for the first receiver, and so on.

9.1 Communication cost

The communication cost is estimated in term of the number of messages exchanged among the server and the various subjects. We estimate such cost for both a conventional centralized system and our distributed approach. We are thus able to compare those two approaches.

1. In a *conventional centralized system* the number of messages sent by the document server and the subjects is equal to two times the average number of access requests ($\mathcal{AR}_{avg}$) multiplied by the number of interactions, that in this case corresponds to the number of sessions opened to issue access control requests. The resulting communication cost is thus: $2\mathcal{N}_i \cdot \mathcal{AR}_{avg}$.

2. In *our distributed approach* the number of messages sent by the document server or by the subjects involved in the collaborative update process, is as follows:

   $$\#\textit{sent-messages} = (\mathcal{N}_i + 1) + 2(\mathrm{R}_R + \mathrm{P}_R) + 2\mathrm{P}_R(\mathcal{CG} - 2) + 2\mathcal{SRM}(\mathrm{R}_R, \min\{\mathrm{P}_R, (\mathcal{N}_i - 2)\})$$

   where:

   (a) R_R, with $0 \leq \mathrm{R}_R \leq (\mathcal{N}_i - 1)$, is the number of region recovery requests. The upper bound for this parameter is $(\mathcal{N}_i - 1)$, because the first subject receiving the package from the server does not certainly need a recovery.

   (b) P_R, with $0 \leq \mathrm{P}_R \leq (\mathcal{N}_i - 1)$, is the number of path recovery requests. Also in this case the upper bound for this parameter is $(\mathcal{N}_i - 1)$, because the first subject receiving the package from the server does not certainly need a path recovery.

   (c) $(\mathcal{N}_i + 1)$ is the number of messages, containing the package, needed to reach, one or more times, all the interacting subjects and to return to the document server.

   (d) $2(\mathrm{R}_R + \mathrm{P}_R)$ is the global number of recovery requests and answers

   (e) $2\mathrm{P}_R(\mathcal{CG} - 2)$ is the total number of path recovery requests and answers sent to/by the subjects during a collaborative update process. Whenever a path recovery request reaches the document server, the server sends all the subjects in the collaborative group, apart the path recovery request sender and the subject from which that subject has received the corrupted path, a request to obtain the last path they have received within the package.

   (f) $2\mathcal{SRM}(\mathrm{R}_R, \min\{\mathrm{P}_R, (\mathcal{N}_i - 2)\})$ is the number of sent recovery messages needed to manage R_R region recovery requests, given P_R path recovery requests. In particular, we consider the worst case in which both the region recovery requests and the path recovery requests are respectively sent by the last R_R and P_R subjects in the path. Function $\mathcal{SRM}(x, y)$ is defined as follows:

   $$\mathcal{SRM}(x, y) = \sum_{i=\mathcal{N}_i-2-(x-1)}^{\mathcal{N}_i-2} min\{i, \mathcal{N}_i - 2 - y, \mathcal{CG}\},\ y \leq \mathcal{N}_i - 2$$

   Note that values for variable y must be less or equal than $\mathcal{N}_i - 2$, because when the number of path recovery requests is equal to $\mathcal{N}_i - 1$ the recovery protocol does not send any region recovery message, thus the behavior followed is the same of when $y = \mathcal{N}_i - 2$. Moreover, for each region recovery request the server sends a number of messages, to obtain the correct version of one or more regions, equal to the number of subjects that precede the last one in the Path_d,[8] and however at most $\mathcal{CG}$ messages.

Proposition 9.1 (Communication cost). Let r_r and p_r be rational numbers such that $r_r = \frac{R_R}{\mathcal{N}_i}$ and $p_r = \frac{P_R}{\mathcal{N}_i}$. Our distributed approach has a lower communication cost than a conventional centralized system when $\mathcal{AR}_{avg} \geq (1 + p_r + r_r + p_r \cdot \mathcal{CG} + r_r \cdot \mathcal{CG})$. △

Proof: According to the analysis given above we have that in general the number of sent messages in a conventional centralized system is equal to $2\mathcal{N}_i \cdot \mathcal{AR}_{avg}$, whereas in our distributed approach is estimated as follows:

[8] Note that also in this case we consider the worst case in which all the considered subjects are distinct.

$$\begin{aligned} \#\textit{sent-messages} &\leq (\mathcal{N}_i + 1) + 2(\mathrm{R}_R + \mathrm{P}_R) + 2\mathrm{P}_R{\cdot}\ \mathcal{CG} + 2\mathrm{R}_R{\cdot}\ \mathcal{CG} \\ &\leq (\mathcal{N}_i + 1) + 2(\mathrm{r}_r \cdot \mathcal{N}_i + \mathrm{p}_r \cdot \mathcal{N}_i) + 2\mathrm{p}_r{\cdot}\mathcal{N}_i{\cdot}\ \mathcal{CG} + 2\mathrm{r}_r{\cdot}\ \mathcal{N}_i{\cdot}\ \mathcal{CG} \\ &< 2\mathcal{N}_i(1 + \mathrm{p}_r + \mathrm{r}_r + \mathrm{p}_r{\cdot}\ \mathcal{CG} + \mathrm{r}_r{\cdot}\ \mathcal{CG}) \end{aligned}$$

It is now clear that when the hypothesis is true the number of sent messages in our distributed approach is less than the number of sent messages required in a conventional centralized system. ◇

According to the above results it is clear that whether our approach is more efficient than a centralized one depends on the frequency of recovery. Based on this observation we plan to extend our system with an adaptive behaviour for recovery management. The adaptive behaviour will allow the system to use a centralized or distributed protocol according to an estimated average number of access requests and the regions/path recovery rates.

9.2 Size of exchanged information

Here we are interested in the amount of data exchanged in each step of a collaborative update process in both a conventional centralized system and in our distributed approach. Tables 8 and 9 contains data useful for such estimation in both the approaches. More precisely, in Table 8 we specify the size of the building blocks that compose a package for the centralized approach, the distributed one, and for an access request.

Table 8: Size of the building blocks of a package/acces request

Building Block	Size	Semantics
id	*S(id)*	size of a region/atomic element/subject identifier
hash	*S(hash)*	size of an hash value
digital signature	*S(digital signature)*	size of a digital signature
d^e	$S(d^e)$	size of the encrypted document
ar	*S(ar)*	size of the access request declaration given in terms of atomic elements to be updated
ac	*S(ac)*	size of an authoring certificate
up_{ae}	$S(up_{ae})$	size of the updated portion (*up*), sent to replace the old portions of an atomic element (*ae*), with old size S(*ae*), contained in the document.

By contrast, in Table 9 we show the size of three basic package structures that compose a package for the distributed approach: a *modifiable atomic element*, a *modifiable region*, and a *path specification*, in terms of the size of their building blocks.

1. In a *conventional centralized system* the worst case occurs when the view to be generated for the next receiver is equal to the whole XML document d. Indeed the system has to generate a simmetric session key and encrypt the whole XML document before sending it to the next receiver.[9] The system also computes a signature over the generated view and then attaches this signature to the view itself, forming a package to be sent to the designated receiver. Moreover, during a step of a centralized collaborative update process a subject s in the worst case, that is when a different access control policy applies to each atomic element and all these policies apply to s, sends a number of access requests, and corresponding updated portions, equal to AE(d), the number of atomic elements that compose d. We can estimate the amount of data exchanged

[9]Here we do not consider the generation and distribution of the corresponding authoring certificates, that will be treated in Section 9.3

Table 9: Size of the basic package structures

Structure	Size	Component	Semantics
modifiable atomic element	*S(modifiable atomic element)*	4S(id)	size of the atomic element and three position identifiers
		S(hash)	size of the component *h_ae*
		S(digital signature)	size of the component *full*
modifiable region	*S(modifiable region)*	5S(id)	size of one region, two subjects, $cycle\_path_{last-1}$ $cycle\_path_{last}$ identifiers
		4S(hash)	size of the components *prev_r_h*, *prev_r_dig_h*, *last_r_h* and *last_r_dig_h*
		4S(digital signature)	size of the digital signatures stored in the components $h\_c\_s_{last-1}$*_dig-sig*, h_serv_{last-1} $h\_c\_s_{last}$*_dig-sig* and h_serv_{last}
		2S(ac)	size of the components *certificates* each containing one certificate
path specification	*S(path specification)*	3S(id)	size of components *s_id_c*, *prog* and *s_id_next*
		S(digital signature)	size of the component $h_{control}$

between the centralized system and a subject during a step of the centralized collaborative update process as follows.

Proposition 9.2 (Size of exchanged information for the centralized approach). Let d be an XML document, CP_d be the generated corresponding package for the centralized approach, $\mathrm{S}(CP_d)$ be the size of CP_d, and $\mathrm{S}(AR)$ be the size of AR, where AR is the set of access requests sent by a subject s during a step of a centralized collaborative update process. In the worst case: $\mathrm{S}(CP_d) + \mathrm{S}(AR) \leq c{\cdot}\mathrm{AE}(d),\ c \in \mathbb{N}$. △

Proof: The size of a package for the centralized approach and the size of the set of access requests sent by a subject during a step of a centralized collaborative update process are as follows:

1) $\mathrm{S}(CP_d) = \mathrm{S}(\mathrm{d}^e) + \mathrm{S}(\text{digital signature})$.
2) $\mathrm{S}(AR) = \mathrm{AE}(d){\cdot}\mathrm{S}(ar) + \sum_{ae \in AE(d)} \mathrm{S}(up_{ae})$

By considering that: a) $\exists\ \widehat{c} \in \mathbb{N}$: $\mathrm{S}(\mathrm{d}^e) \leq \widehat{c}{\cdot}\mathrm{AE}(d)$, because the size of the encrypted document is linear in the number of atomic elements that compose the original document; b) S(digital signature), $\mathrm{S}(ar)$ are constant values; c) $\exists\ \widetilde{c} \in \mathbb{N}\ \forall\ ae \in \mathrm{AE}(d)$: $\mathrm{S}(up_{ae}) \leq \widetilde{c}{\cdot}\mathrm{S}(ae)$; and d) $\exists\ \bar{c} \in \mathbb{N}$: $\sum_{ae \in AE(d)} \mathrm{S}(ae) \leq \bar{c}\mathrm{AE}(d)$, it is clear that exists a natural number c such that:

$\mathrm{S}(CP_d) + \mathrm{S}(AR)$
$\leq \mathrm{AE}(d){\cdot}[\widehat{c} + \mathrm{S}(ar)] + \widetilde{c}\sum_{ae \in AE(d)} \mathrm{S}(ae) + \mathrm{S}(\text{digital signature})$
$\leq \mathrm{AE}(d){\cdot}[\widehat{c} + \mathrm{S}(ar) + \widetilde{c}{\cdot}\bar{c}] + \mathrm{S}(\text{digital signature})$
$\leq c{\cdot}\mathrm{AE}(d)$. ◇

2. In *our distributed approach* the worst case occurs when the following conditions hold: 1) the number of regions associated with d, $\mathrm{R}(d)$, is equal to the number of atomic elements of d, $\mathrm{AE}(d)$; 2) all the regions are modifiable ones; and 3) the current subject updates all the atomic elements. The size of the information exchanged during a step of a distributed collaborative update process

is equal to the size of the package sent by the current subject to the next chosen receiver. The size of such a package is defined as follows.

Proposition 9.3 (Size of a package for the distributed approach). Let d be an XML document, DP_d be the generated corresponding package for the distributed approach, and $\mathrm{S}(DP_d)$ be the size of DP_d. In the worst case: $\mathrm{S}(DP_d) \leq c{\cdot}\mathrm{AE}(d)$, $c \in \mathbb{N}$. △

Proof: The size of a package for the distributed approach is as follows:
$\mathrm{S}(DP_d) = \mathrm{S}(\mathrm{d}^e) + 2\mathrm{S}(\text{digital signature}) + \mathrm{S}(\mathrm{id}) + \mathrm{R}(d){\cdot}\ \mathrm{S}(\text{modifiable region}) + \mathrm{AE}(d){\cdot}\ \mathrm{S}(\text{modifiable atomic element}) + Path_d{\cdot}\ \mathrm{S}(\text{path specification}) = \mathrm{S}(\mathrm{d}^e) + \mathrm{S}(\mathrm{id}){\cdot}\ [5\mathrm{R}(d) + 4\mathrm{AE}(d) + 3Path_d + 1] + \mathrm{S}(\text{digital signature}){\cdot}\ [4\mathrm{R}(d) + \mathrm{AE}(d) + Path_d + 2] + \mathrm{S}(\mathrm{hash}){\cdot}\ [4\mathrm{R}(d) + \mathrm{AE}(d)] + 2\mathrm{S}(ac){\cdot}\mathrm{R}(d)$.
By considering that: a) $\exists\ \widehat{c} \in \mathbb{N}$: $\mathrm{S}(\mathrm{d}^e) \leq \widehat{c}{\cdot}\mathrm{AE}(d)$, because the size of the encrypted document is linear in the number of atomic elements that compose the original document; b) S(id), S(digital signature), S(hash) and S(ac) are constant values; c) the assumption that R(d) in the worst case has the same cardinality of AE(d); d) the fact that a package contains: 2 digital signatures (one computed over the entire package, and another over the H_{NMI} control information), and one identifier (component cycle_path); and e) cardinality of $Path_d \ll \mathrm{AE}(d)$; it is clear that exists a natural number c such that $\mathrm{S}(DP_d) \leq c\ {\cdot}\mathrm{AE}(d)$. ◇

According to the above results, it is clear that the size of exchanged information in both approaches linearly grows wrt the number of atomic elements that compose the original document, thus our distributed approach can offer a bandwidth cost similar to the one for the centralized approach. The major benefit of our approach wrt the centralized one remains the reduced number of messages sent during the collaborative update process, thus the communication cost is still the effective parameter of choice between the distributed approach and the centralized one.

9.3 Number of certificates generated for a document d

The worst case occurs when the graph representation of the document d is a list of nodes and all the policies that apply to this document contain the `delete_elemt` privilege with propagation option equal to CASCADE. In particular, the policies apply to the document as follows: the first policy applies to the root of the document, the second policy applies to the child (second node in the list) of the document and so forth. According to this scenario the number of certificates that must be generated is equal to $[\sum_{i=0}^{\mathcal{P}-1} \mathrm{S}_{i+1} \cdot (\mathrm{AE}(d) - i)]$, where S_j is the set of subjects satisfying the j^{th} policy ($1{\leq}j{\leq}\mathcal{P}$). The number of certificates is always less than or equal to $max\{\mathrm{S}_j \mid 1{\leq}j{\leq}\mathcal{P}\} \cdot \mathcal{P} \cdot (\mathrm{AE}(d) - \frac{\mathcal{P}-1}{2})$. In the worst case $\mathcal{P}$ is equal to AE(d) and thus the number of certificates has an upper bound equal to $max\{\mathrm{S}_j \mid 1{\leq}j{\leq}\mathcal{P}\} \cdot \frac{AE(d)^2}{2}$.
Only the distributed approach presents this cost associated with the generation and dissemination of the certificates. Moreover, the possible update and/or revocation of subject credentials and access control policies add a further cost due to the generation of new certificates and the revocation of those ones no more valid. Since our distributed approach postpones these events at the end of a collaborative update process, this last cost can be managed off-line without weighing on the process itself. According to the result above to make our approach better than the centralized one we propose an adaptive system that evaluates the number of certificates to be generated before starting a collaborative process to choose the best strategy of generation and dissemination of certificates among those presented in Section 6.1.

9.4 Execution time of the integrity check protocol and distributed view generation/decryption vs execution time of the centralized view generation/encryption and distributed decryption

In this section we analyze and then we compare the time cost required to enable the next receiver to view document portions for which it possesses a privilege, and to modify the document content according to its modification rights stated in the policies belonging to the $\mathcal{PB}$. In a conventional centralized system this time is spent by the centralized system to generate and encrypt the receiver document view and by the receiver to decrypt such a view, whereas in our distributed approach this time concerning the execution of the integrity check protocol, and the generation/decryption of the receiver view.

1. In a *conventional centralized system* the worst case occurs when a different access control policy applies to each atomic element of the document and also all these policies apply to the next receiver, implying that the view to be generated and encrypted by the centralized system and decrypted by the receiver consists of the whole document. The time required to accomplish the task of generating and encrypting the next receiver document view by the centralized system and to decrypt that view by the receiver can be evaluated as follows.

   Proposition 9.4 (Centralized view generation/encryption and distributed view decryption time) . Let d be an XML document, $\mathcal{P}$ be the set of access control policies that apply to d, nr be the next receiver, and $T(view)$ be the time required by the centralized system to generate and encrypt the nr's document view and by nr to decrypt that view. In the worst case: $\exists\ c \in \mathbb{N}$: $T(view) \leq c{\cdot}[\mathrm{AE}(d)]^2$. $\triangle$

   Proof: The time required to generate the receiver document view, takes also into account the time required for parsing the document and searching policies that apply to each parsed element. Thus in the above-mentioned worst case: a) the number of policies that apply to the document is equal to $\mathrm{AE}(d)$; b) the process used to find out the policy p that applies to a particular atomic element ae, implies a sequential search in the set $\mathcal{P}$ that stops when p is found; and c) the encryption/decryption phases have a cost in time, respectively denoted as $T(view\ enc)$ and $T(view\ dec)$, that is proportional to the size of the document, that is $\exists\ \bar{c} \in \mathbb{N}:\ T(view\ enc) \leq \bar{c}{\cdot}\mathrm{AE}(d)$ and $\exists\ \tilde{c} \in \mathbb{N}:\ T(view\ dec) \leq \tilde{c}{\cdot}\mathrm{AE}(d)$; it is clear that there exists a natural number c such that: $T(view) = \sum_{i=1}^{AE(d)} i + T(view\ enc) + T(view\ dec) \leq \frac{AE(d){\cdot}[AE(d)+1]}{2} + \mathrm{AE}(d)[\bar{c}+\tilde{c}] \leq c{\cdot}[\mathrm{AE}(d)]^2$ $\diamond$

2. In *our distributed approach* the worst case occurs when: the number of regions $\mathrm{R}(d)$ is equal to the number of the atomic elements $\mathrm{AE}(d)$ that compose the document d; and there is at least a policy with privilege `update_attr` that applies to each region. This is the case in which the protocol requires the highest number of operations to perform the region integrity check and the number of regions that requires such a set of operations is maximum. Moreover the receiver view to be generated and decrypted consists of the whole document. In this case the integrity check of an updatable region r requires the following steps:

   (a) Integrity check of the information inserted by the last but one subject (s_{last-1}) in the region r, that requires the local computation of an hash value and the decryption of a digital signature (component $h_c_s_{last-1}$_dig-sig).

   (b) Integrity check of the components h_ae associated with the atomic elements belonging to r, that requires the local computation of an hash value and its comparison with the component $prev\_r\_dig\_h$.

(c) Local computation of an hash value, one for each atomic element belonging to r, over the encrypted content of an atomic element.

(d) Integrity check of the information inserted by the last subject (s_{last}) in the region r, that requires the local computation of an hash value and the decryption of a digital signature (component $h_c_s_{last}$_dig-sig).

(e) Check that the value contained in component *last_r_dig_h* is equal to the hash value locally computed over the hash values computed over the atomic elements belonging to r.

(f) Check that the value contained in component *last_r_h* is equal to the hash value locally computed over the atomic elements belonging to r.

(g) Integrity check of the certificate inserted in the component *certificates* by the last subject (s_{last}), that requires the local computation of an hash value and the decryption of a digital signature, only if the region contains an atomic element that is an attribute, since only in this case a certificate is inserted and a correct update over that attribute can be performed by the last subject.

Table 10 shows the time required to perform some basic operations, whereas Table 11 shows the time required to perform the integrity check protocol operations.

Table 10: Time required to perform the basic integrity check protocol operations

Basic operation	Time	Semantics
hash	*T(hash)*	time required to compute an hash value
digital signature	*T(digital signature)*	time required to decrypt a digital signature using the corresponding public key

Table 11: Time required to perform the integrity check protocol operations

Operation	Time Notation	Time expression
package signature check	*T(package signature check)*	T(hash) + T(digital signature)
H_{NMI} *check*	*T*(H_{NMI} *check*)	T(hash) + T(digital signature)
$Path_d$ *check*	*T*($Path_d$ *check*)	$Path_d$ · [T(hash) + T(digital signature)]
step_a	*T(step_a)*	T(hash) + T(digital signature)
step_b	*T(step_b)*	T(hash)
step_c	*T(step_c)*	T(hash)
step_d	*T(step_d)*	T(hash) + T(digital signature)
step_e	*T(step_e)*	T(hash)
step_f	*T(step_f)*	T(hash)
step_g	*T(step_g)*	T(hash) + T(digital signature)

Furthermore given the set of regions accessible by the receiver, denoted as *AccReg*, the view generation process requires a sequential research in the package to find out each accessible region and their atomic elements.

Proposition 9.5 (Integrity check protocol and distributed view generation/decrption time). Let d be an XML document, P_d be the corresponding document package, nr be the next receiver, and $T(view)$ be the time spent to check the package integrity and to generate/decrypt the nr's document view. In the worst case: $\exists\ c \in \mathbb{N}$: $T(view) \leq c \cdot [\mathrm{AE}(d)]^2$. △

Proof: The time required to execute the integrity check protocol applied to P_d, denoted as $T(P_d)$, can be estimated as follows:
$T(P_d)$ = T(package signature check) + T(H_{NMI} check) + T($Path_d$ check) + R(d) · [T(step_a) + T(step_b) + T(step_c) + T(step_d) + T(step_e) + T(step_f) + T(step_g)] = T(hash)·[2 + $Path_d$ + 7R(d)] + T(digital signature)·[2 + $Path_d$ + 3R(d)]. Since in the worst case: a) the number of regions R(d) is considered equal to the number of the atomic elements AE(d), b) T(hash) and T(digital signature) can be considered as constants, c) cardinality of $Path_d \ll$ AE(d), d) the decryption time, denoted as $T(view\ dec)$, is proportional to the document size, that is $\exists\ \bar{c} \in \mathbb{N}$: $T(view\ dec) \leq \bar{c}$·AE($d$), e) cardinality of *AccReg* is equal to AE(d), it is clear that exists a natural number c such that:
$T(view) = T(P_d) + \sum_{i=1}^{AE(d)} i + T(view\ dec) \leq c$·[AE($d$)]^2 ◇

The above results shows that both approaches require a similar time cost to enable a receiver to have access to its document view. The communication and generation/dissemination costs are the parameters according to which it is possible to choose the best approach.

10 Parallel Document Updates

In our current approach we assume that each subject can send the package to only one subsequent subject; therefore we disallow parallel updates to the same document. Relaxing such an assumption, and thus supporting parallel updates to documents would be however an important improvement to our protocol. Here we briefly discuss two alternative approaches referred to as *restricted parallel update* and *fully parallel update*. These approaches differ in that the first only allows parallel updates on disjoint portions of the same document, whereas the second does not impose such a restriction.
The first approach can be supported by inserting some additional information in the $Path_d$ structure in order to manage the case in which a subject receives more than one package containing the same document. In particular such information, specified by the subject that sends the same package to a set of subjects, should contain for each next receiver the set of regions that could be modified by that receiver and all the subsequent receivers belonging to that new parallel path. A partition of the set R(d) of all document regions is therefore generated and a first receiver is associated with each subset of R(d) belonging to the generated partition. Moreover, all the chosen first receivers must be distinct. Whenever a subject receives more than one package, it merges all the documents obtaining a new document containing the regions modified by the subjects of the first parallel path, those modified by the subjects of the second parallel path and so forth. The structure $Path_d$ will thus contain all the parallel paths belonging to merged packages. In this scenario each time the information *cycle_path* is updated by the server the structure $Path_d$ is not re-initialized and the portion of it that must be checked to detect a new cycle will be that following the server identifier inserted by the server itself in the Path$_d$ structure during the recovery procedure. Obviously, this check will have to be performed over all the parallel paths stored into $Path_d$ structure. It is also necessary to store in the merged package all the values of the component *cycle_path* contained in the received packages.
The second approach can be supported by storing in the package all the versions concerning the same region, that is, by keeping track of all modifications occurred to a region by giving to a set of pre-defined subjects the possibility of selecting that one that they consider the "best version". The other subjects can only add their updates to the region generating a new additional version of it. Whenever a subject receives more than one package it merges them by collecting from the received packages all the stored versions grouped by region.

11 Concluding Remarks

In this paper, we have proposed an approach for secure and selective document updates in a distributed environment. In particular, we have specified all protocols required by our approach and we have provided complexity results for the proposed approach.
We plan to extend this work in several directions. A first direction concerns the extension of our approach to peer-to-peer architectures. A second direction concerns the extension of our protocols to relax the assumptions introduced in Section 4.2. To reach this goal the idea is that of recording the *modification history* of a document. According to this strategy, all the modifications executed by the subjects on a document are stored in the associated package and broadcasted to other subjects in the collaborative group. In this way other subjects are able to know which subjects have received that document and which privileges they have exercised on it. This approach however requires sending a higher number of messages and introducing additional control information in the package. We thus plan to offer as part of our system a suite of distributed collaborative update protocols and letting the subjects choosing the protocol to be used for a specific document, according to the trade-off they want to make between security and efficiency. A third direction deals with introducing authorizations related to the modification of document flow paths. This feature is relevant in particular in decentralized workflow management systems where decisions about routing and re-routing of documents must be taken during the workflow execution. In this respect, it is important to state authorizations specifying which subjects can modify the document flow path and under which circumstances. A fourth direction deals with revocation and updates of keys and certificates. More precisely whenever credentials associated with subjects or access control policies in $\mathcal{PB}$ are deleted or updated a change in the document marking and in the set of rights possessed by a subject occurs. The obvious consequence is the update of the keys used to encrypt the document portions associated with a changed marking and the re-encryption of those portions using new keys. By contrast the change in the set of rights possessed by a subject cause the revocation of the certificates that granted the rights no more available, inserting them in a Revocation List accessible by every subject involved in collaborative update processes, and the generation/dissemination of new certificates enabling that subject to exercise the new obtained rights.

A Correctness of the Subject Protocol

In this section we present correctness results for the subject protocol. The main property that this protocol assures is that a subject is not able to exercise an authoring privilege for which it does not possess the proper authorization. In particular, we show that the protocol can detect whether the document content or the control data structures have been tampered. In what follows we focus on some tampering cases, and we show how the protocol detects those incorrect modifications. Tampering cases on which we focus are the following:

1. modification of non-modifiable information;
2. modification of the document content associated with a non-modifiable region;
3. modifiable region tampering:
   (a) use of an authoring privilege for which a subject does not possess the proper authorization;
   (b) removal of the control information associated with a modifiable region;
   (c) substitution of a modifiable region content and the corresponding control information with those of a previous version;

1) Modification of non-modifiable information.
This type of tampering cannot arise because a modification to non-modifiable information implies a corresponding modification of the H_{NMI} hash value, that can be modified only by the D_s, since this hash value is signed with the D_s private key.
2) Modification of the document content associated with a non-modifiable region.
In this case a modification of the content of a non-modifiable region *r_id* implies a corresponding modification of the H_{r_id} hash value stored in the tuple of the NMR_d control data structure associated with that region. A modification to a H_{r_id} hash value implies a corresponding modification of the H_{NMI} hash value. Based on the same reasonsing we have used in item 1, we can conclude that this type of tampering cannot arise.
3.a) Use of an authoring privilege for which a subject does not possess the proper authorization.
In this case a subject illegally modifies the content of a modifiable region *r_id* without inserting at least one authoring certificate in the component *certificates* associated with *r_id* or inserting in this component one or more proper and improper authoring certificates or only improper authoring certificates. In the former type of tampering the protocol detects the illegal modification to the content of that region, because at least one of the two hash values stored in components last_r_h and prev_r_h does not match the one locally computed over the content of *r_id*. This is due to the fact that the malicious subject cannot modify both these control hash values according to the new illegal content of the region, because they are signed by two different subjects as required by the protocol.
In the latter type of tampering the protocol detects the illegal modification of the content of that region, because at least one of the following checks raises an error. First of all the protocol checks the integrity of the inserted certificates to detect a possible modification of their contents. Then, the protocol checks: 1) whether the subject specified in the authoring certificates matches the one declared in component s_{last}, to detect the case in which a subject uses an authoring certificate of another subject to validate its modification; 2) whether the region specified in the authoring certificates matches *r_id*; 3) whether the atomic elements not yet deleted and not declared as modified have kept their previous content and that the other ones declared as modified have been modified according to the privileges contained in the inserted certificates;
3.b) Removal of the control information associated with a modifiable region.
This tampering cannot arise because a modification of a modifiable region identifier implies a corresponding modification of the H_{NMI} hash value, that cannot be modified by a subject, as stated in item 1 above. Therefore it is not possible for the subject to delete a whole tuple in MR_d. Moreover also assuming that the region content in d^e is empty, there must still be two hash values, i.e. *last_r_h* and *prev_r_h*, signed by two different subjects, which state that the region was correctly deleted.
3.c) Substitution of a modifiable region content and the corresponding control information with those of a previous version.
In this case a malicious subject s inserts in d^e and in MR_d of its current package version the content associated with the atomic elements belonging to a modifiable region *r_id* and the corresponding values for the components associated with *r_id* in MR_d, all stored in a previous version of the package. Since, according to the assumptions given in Section 7.1.2, the value of *cycle_path* and of all involved control data structures are updated whenever a subject finds in the structure $Path_d$ its next chosen receiver, s cannot perform such a substitution, because the control information belonging to a previous version of a package, stored in the local repository of s, was necessarily generated on a value of *cycle_path* that is different by the current one stored in the current package received by s and moreover the current value of *cycle_path* is not modifiable, as stated in item 1 above.
A type of illegal substitution is however possible, if the following conditions are all satisfied:

1. `s` keeps a version of the package (`pvp`) that precedes the current one (`cvp`) that `s` receives;

2. among the subjects that received the package after `s` only one of them (`sbj`) modified a modifiable region *r_id*, exercising only the `update_attr` privilege over some its atomic elements;

3. the modifiable region *r_id* results not yet confirmed in `cvp`.

According to the previous conditions `s` is able to copy in `cvp` the portion of `pvp` associated with the atomic elements belonging to *r_id*, delete the information inserted by `sbj` in the tuple associated with *r_id* in MR_d of `cvp` and confirm the new content of *r_id*. At this point the region has a previous content and the successor subjects are not able to detect this illegal substitution.
We are however developing an approach able to address this last case, at the price however of an increased communication and storage complexity. We have outlined such an approach in the concluding section.

References

[1] E. Bertino, S. Castano, E. Ferrari. On Specifying Security Policies for Web Documents with an XML-based Language. *Proc. of SACMAT'2001, ACM Symposium on Access Control Models and Technologies*, Fairfax, VA, May 2001.

[2] E. Bertino, S. Castano, E. Ferrari. Author-$\mathcal{X}$: a Comprehensive System for Securing XML Documents, *IEEE Internet Computing*, 5(3):21–31, May/June 2001.

[3] E. Bertino, and E. Ferrari. Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and Systems Security*, 5(3): 290-331 (2002).

[4] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML Documents. In *Proc. 6th International Conference on Extending Database Technology*, Konstanz, Germany, March 2000, pages 121-135.

[5] The Excelon Home Page. http://www.exceloncorp.com

[6] eXtensible Access Control Markup Language TC. XACML 1.0 Specification Set (18 Feb. 2003): OASIS Standard. Available at: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

[7] E. Fernandez, E. Gudes, and H. Song. A Model for Evaluation and Administration of Security in Object-Oriented Databases. *IEEE Transactions on Knowledge and Data Engineering*, 6:275–292, April 1994.

[8] C. Geuer Pollmann. The XML Security Page. http://www.nue.et-inf.uni-siegen.de/geuer-pollmann/xml_security.html

[9] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Trans. on Database Systems*, 16(1):88–131, March 1991.

[10] R. Sandhu et al. Role-based Access Control Models. *IEEE Computer*, pages 38-47, 1996.

[11] P. Samarati, E. Bertino, and S. Jajodia. An Authorization Model for a Distributed Hypertext System. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):555–562, 1996.

[12] Security Assertion Markup Language, SAML v1.1 Standard Specification set (2 September 2003): OASIS Standard. Available at: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`

[13] D. Srivastava. Directories: Managing Data for Networked Applications. Tutorial presented at the *16th IEEE International Conference on Database Engineering (ICDE'00)*, San Diego (CA), March 2000.

[14] SSL Protocol (Secure Socket Layer) Available at: `http://developer.netscape.com/docs/manuals/security/sslin/`.

[15] B.Thuraisingham, A. Gupta, E.Bertino, E.Ferrari. Collaborative Commerce and Knowledge Management Across Borders, *Knowledge and Process Management*, Vol.9, No. 1, pp. 43-53, January 2002.

[16] World Wide Web Consortium. Extensible Markup Language (XML) 1.0, (Third Edition) 2004. Available at http://www.w3.org/TR/2004/REC-xml-20040204

[17] World Wide Web Consortium. XML Encryption Syntax and Processing, 2002. Available at: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[18] World Wide Web Consortium. XML Path Language (Xpath), 1.0, 1999. Available at: http://www.w3.org/TR/1999/REC-xpath-19991116

[19] World Wide Web Consortium. XML Signature Syntax and Processing, 2002. Available at: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/