Chapter 12

DISTRIBUTED COOPERATIVE UPDATES OF XML DOCUMENTS

Elisa Bertino, Elena Ferrari and Giovanni Mella

Abstract

Securing web documents entails addressing two key issues: privacy and integrity. Whereas the problem of privacy has been widely investigated, the problem of how to ensure that a document, when moving among different parties, is modified only according to the stated policies still lack comprehensive solutions. In this paper we present a solution to this problem by proposing a model for specifying authoring policies, and an infrastructure supporting the specification and enforcement of these policies in a distributed and cooperative environment. The key aspect of our proposal is that, by using a combination of hash functions and digital signature techniques, a subject is able to verify, upon receiving a document, whether the update operations performed on the document till that point are correct with respect to the authoring policies, without interacting with the document server.

Keywords: Web, distributed updates, access control policies

1. Introduction

Internet has made possible a wide spectrum of distributed cooperative applications in several areas, such as collaborative e-commerce [5], distance learning, telemedicine, and e-government. A requirement common to many cooperative applications is the need for secure document exchange. By secure exchange we mean that the document privacy and integrity are ensured when documents flow among different parties. Whereas the problem of documents privacy has been widely investigated, the problem of how to ensure that a document, when moving among different parties, is modified only according to the stated policies still lack comprehensive solutions. We believe that such a comprehensive solution requires: 1) a model and a high-level language for specifying authoring policies - we believe that 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; and 2) an infrastructure supporting the specification and enforcement of such policies in a distributed environment. In this paper, we present such a com-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35697-6_26](https://doi.org/10.1007/978-0-387-35697-6_26)

E. Gudes et al. (eds.), *Research Directions in Data and Applications Security*

© IFIP International Federation for Information Processing 2003

prehensive solution. We assume that documents to be protected are encoded in XML. We have chosen to cast our approach on 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 fine-grained authoring privileges, such as those for deleting and modifying document elements and attributes, or inserting new elements and attributes into documents. Our access control model makes use of subject credentials and access control policies are expressed in terms of conditions against the subject credentials. Both credentials and authoring policies are encoded in XML. Therefore, not only we provide a high-level language for policy specifications, but we can also apply the protection mechanisms we provide for regular XML documents to credentials and access control policies. Our access control model is complemented by an infrastructure supporting secure cooperative document updates. The basic idea is that the document server sends the document to be modified to a given subject; this subject then forwards the document to a second subject and so forth. Each subject receiving the document from the server or from the previous subject along the path must be able to modify all and only those portions of the document for which he/she has a proper authorization. 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. To the best of our knowledge, the work reported in this paper is the first addressing the problem of XML document distributed updates. The remainder of this paper is organized as follows. Section 2 briefly summarizes our access control model. Section 3 presents the architecture for managing distributed updates, whereas Section 4 deals with document encryption. Required data structures are covered by Sections 5, whereas Sections 6 describes the protocol for integrity verification. Finally Section 7 concludes the paper and outlines future research direction.

2. Access Control Model for XML Documents

Our model [1] is based on the traditional access control paradigm expressing authorizations in terms of subjects, objects, and privileges.

Subjects. Subjects are qualified by means of *credentials*. Each subject has one or more associated credentials that are assigned when a subject subscribes to the system. Access control policies specify conditions on the credentials and credential attributes. These conditions, expressed by means of an XPath-based language, identify the set of subjects to which a policy applies.

Protection Objects. By protection objects 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. This wide range of protection objects is complemented with content-dependent access control, that is, access control policies based on document content.

Privileges. Access control policies can be categorized into two groups: *authoring policies*, that allow a subject to modify a protection object, and *browsing policies*, that allow a subject to access the information contained in a protection object. The semantics of the access privileges supported by our model is given in Table 1.

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/URI(s).
	navigate	To see all the links implied by attributes of type IDREF/URI(s) contained in a protection object.
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_elemnt	To insert new elements that are direct subelements of the element on which the insert_elemnt privilege is specified
	delete_elemnt	To remove the subtree rooted at the element on which the delete_elemnt privilege is specified

We denote with the term *Policy Base* ($\mathcal{PB}$) the XML file encoding the access control policies of the source $\mathcal{S}$. Figure 2 shows a $\mathcal{PB}$ referring to the XML document in Figure 1.

3. Distributed Updates of XML Documents

Our approach to distributed updates is motivated by the fact that often, within an organization, XML documents are subject to pre-defined cooperative update processes 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 he/she has a proper authorization. Therefore we need to develop a framework supporting this update mode, able to minimize the in-

Figure 1. (a) Sample XML document; (b) Corresponding graph representation.

teractions with the document server and, at the same time, guaranteeing the correct enforcement of access privileges on the document. The framework we have developed is based on an approach similar to CryptolopeTM [4], which consists of using different keys for encrypting different portions of the same document. 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 must receive the document for updating or reading it. Each subject in the collaborative group receives the key(s) for the portion(s) he/she is enabled to see and/or modify and information on the order in which the other subjects in the collaborative group must receive the document.

```

<policy_base>
  <policy_spec pid="P1" cred_expr="//manager[department="R&D"]"
    target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]"
    priv="view"/ >
  <policy_spec pid="P2" cred_expr="//manager[department="R&D"]"
    target="Department_monthly_report.xml"
    path="//Department_monthly_report[@Department="R&D"]/approval"
    priv="update_attr"/ >
  ...
</policy_base>

```

Figure 2. Policy Base example.

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. This is achieved by attaching to the encrypted document additional *control information*, that make 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*. When a subject finds that a package is compromised, it interacts with the server to obtain the last correct version of the package. To support this schema, we propose the architecture shown in Figure 3 which consists of three main components. The document is first processed by an *Encryption Module* which, on the basis of the specified access control policies, encrypts it with different keys. The results are the encrypted document and a table, named *Key_Info*, containing information on the encryption keys. This table is received as input by the *Control Information Generator*, along with the specified access control policies. On the basis of the information received as input, this module generates the control information. The *Dispatcher* is in charge of generating the document package, containing the encryption of the document and the control information, and of sending it to the first subject in the collaborative. By contrast, encryption keys are separately sent to each subject in the group. Finally, the *Recovery Manager* receives recovery requests from subjects and sends back to them, at the end of the recovery procedure, the last correct version of the package. In the following section we describe the main components of the proposed schema.

4. Document Encryption

The approach we have adopted for generating the encryption of a document d consists of encrypting with the same key those portions of d to which the same policies apply. Due to lack of space we only give the intuition behind the encryption strategy. We refer the interested reader to [2] for further details. The encryption of a document consists of two main phases: the first, called *marking phase*, marks each document portion 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 tag of an element only, allowing one to correctly encrypt elements containing attributes to which different policies apply. This leads to the definition of *atomic element*, which denotes the basic portions of an XML document to which encryption can be applied.

Let d be an XML document in 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

Figure 3. Distributed document updates: Overall schema.

attribute a in e_id : $e_id.a \in AE(d)$; 2) for each element identifier e_id in d , $e_id.tags \in AE(d)$; $\square$

Example of atomic elements in the XML document in Figure 1 are:

- a) &1.Date corresponding to: "Date = "10/1/2000"";
- b) &8.content corresponding to: "10K";
- c) &5.tags corresponding to: "< item " >" "< /item >" $\circ$

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. The set of atomic elements which are encrypted with the same key is called a *region*. 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 set of policies that apply to that region, and the corresponding encryption key. Table 2 shows the content of table *Key_info* for the document in Figure 1, according to the policies in Figure 2.

5. Generation of Control Information

After the XML document has been encrypted, the next step is to generate the control information which form, with the encryption of the document, the document package (Figure 4 presents a graphical representation of a package P_d). The Control Information Generator module generates this information for each region of the document. This information differs according to the access control privileges that can be exercised on a region. For this reason we distinguish between *modifiable* and *non-modifiable* regions. A region r is modifiable if, among the policies that apply to r , there exists at least a policy whose access control mode is either `delete_attr`, `delete_elemnt`, or `update_attr`. A region is non-modifiable if the access control modes of all the policies that apply to that region belong to the set: $\{\text{view, navigate, insert_attr, insert_elemnt}\}$. Note that, operations corresponding to

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

<i>R</i>	<i>K</i>	<i>Policies</i>	<i>Atomic elements</i>
R1	K1	{P1}	{&1.tags, &1.Date, &1.Department}
R2	K2	{P2}	{&4.tags, &4.result}

`insert_attr` and `insert_elem` privileges alter the document contents by inserting a new element and/or attribute; however they do not modify the original region, but they add a new region to the document. Structures for non-modifiable regions are statically defined by the document server when the document is delivered to the first subject and are not altered by subjects during document flow. 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 the remainder of this section we describe the control information associated with document regions. Before presenting this information we need to introduce the notion of *authoring certificate*.

5.1 Authoring Certificates

Authoring certificates are used by a subject, that has modified a document portion, to prove his/her right to modify that document portion to the subsequent receivers of the document. Therefore, whenever a subject modifies a document (or a document portion), he/she has to add the proper authoring certificate to the document control structure. 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 . The document server takes care of sending all the certificates to the correspondent subjects.

Consider two users Ann (`sbj_id`="s₁₀") and Bob (`sbj_id`="s₅₄") with credentials company management director and manager. Suppose that Bob works for the R&D Department, and consider the policies in Figure 2. Then: (`update_attr`, s₁₀, (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`, s₅₄, (R2, {&4.result})) is a valid certificate since Bob is authorized to update attribute result. ○

5.2 Control Data Structures for Document Regions

The Control Information Generator module generates different data structures for non-modifiable and modifiable regions. Control data structures for

non-modifiable regions simply contain a hash value for each non-modifiable region, which 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 he/she 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. We refer the reader to [3] for more details on those data structures. Control information for modifiable regions are more complex because modifiable regions may change dynamically during the document flow. To allow the integrity verification of a modifiable region the protocol must record in the control data structure, 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 he/she does not modify it, and a subject s_{last} , different from s , has modified that region. In particular if a subject s performs a confirmation operation, he/she establishes that the updates executed by subject s_{last} are correct wrt policies in $\mathcal{PB}$. By contrast, a subject s *modifies* a region, when he/she exercises some authoring privileges over it. To be able to verify the integrity of a modifiable region s must know the state of the region when s_{last-1} sent the package to the next subject, the set of elements belonging to the region that s_{last} has modified, 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 it, we must introduce an additional information, called *cycle_path*, that the Control Information Generator inserts into the document package. *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. Control data structures for modifiable regions of a generic document d are introduced in Tables 3 and 4. Among the control information associated with a modifiable region, *certificates* represents a relevant 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. Other relevant control information are represented by control hash values computed over a given region (e.g. *prev_r_h*, *prev_r_dig_h*, *last_r_h*, *last_r_dig_h*) that are particularly relevant for the integrity verification process and are updated during a *confirmation* or a *modification*. A *confirmation* is executed by a subject s by replacing information in components associated with s_{last-1} with the information recorded in the components associated with s_{last} and by inserting in components associated with s_{last} information about s , by setting component *certificates* to *null*, and by leaving unmodified the control hash values. The data structure contains 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 . The subject that confirms a modifiable region has to re-compute the hash value corresponding to atomic elements that was modified by subject s_{last} , to reflect the new value 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 (we recall that a document can flow back to a user in different document flow cycles) then he/she executes a *modification* by inserting in the components associated with s_{last} information regarding the exercised privileges and involved atomic elements (component *certificates*) and by updating the control hash values; otherwise, he/she has to replace information in components associated with s_{last-1} with the information recorded in the components associated with s_{last} , insert in components associated with s_{last} information about himself/herself, 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 4. Graphical representation of a package.

5.3 Generalized Control Information

The last information that needs to be generated, called *generalized control information* consists of some information, called *path of document d* $Path_d$ (see Figure 4), listing the set of subjects that have received the package, and of a hash value, denoted as H_{NMI} , computed over a set of information called *non-modifiable information*. *Non-modifiable information* can be modified only by the document server and corresponds to: *cycle_path*, all control information over non-modifiable regions, the components (*ae_id*, *position*, *r_id*) in all tuples in control structure for modifiable atomic elements, and the component *r_id* in all the tuples in control structure for modifiable regions. $Path_d$ is used to rebuild as much as possible the path followed by a 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. $Path_d$ is incrementally updated as the package flows from one subject to another. Once the Control In-

Table 3. 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}	$(r_id, h_c_s_{last-1}, h_c_s_{last-1_dig} - sig, s_{last-1}, h_c_s_{last}, h_c_s_{last_dig} - sig, s_{last}, h_serv_{last-1}, h_serv_{last})$	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}	$(ae_id, position, h_ae, r_id, full)$	Information corresponding to a specific atomic element ae belonging to a modifiable region r_id of d

formation Generator has initialized the above described control data structures the Dispatcher generates the package to be sent to the first subject.

6. Subject Protocol

A subject s , before receiving the package, obtains from the document server a set of keys, and a set of corresponding region identifiers, enabling him/her to decrypt the portions of the document in the package he/she is able to access and his/her authoring certificates. Whenever s needs a public key of another subject he/she gets it from the document server. The subject protocol consists of three main steps: 1) verification of the package integrity and authentication of the sender, 2) package update, and 3) package delivery to the next subject. Subject s , upon receiving a package, verifies its authenticity and the integrity of the corresponding control information and of the document content he/she is authorized to access. If no errors occur during this phase, s decrypts all the portions of d he/she is able to access with the received decryption keys. Then he/she can exercise over the decrypted portions all the privileges derived by his/her authoring certificates. By contrast, if an error is detected s sends a recovery request to the document server to obtain the last correct version of the package. Before sending the package to the next subject in the collaborative group s must update the data structures to keep track of the operations

he/she has performed on document d . A subject can locally exercise all privileges, authorized by his/her certificates, apart from privileges `insert_attr` and `insert_elem`, because the corresponding operations generate new regions. In this case, the subject must send the package, and new portions he/she wishes to insert into a document, to the document server, that takes care of executing this operation and sending the updated package to the specified receiver. It is important to note that the current version of the subject protocol relies on a set of assumptions that we discuss in what follows. We assume that each time a subject detects that a region is in an inconsistent state (that is, a previous subject has operated on the region violating the policies in $\mathcal{PB}$) sends a recovery request to the document server to obtain the last correct version of the package. 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, without being detectable by any other subject, we assume that 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 document server that updates the value of *cycle_path* and of the other involved structures and then sends the package to subject s_r .

7. 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 control information required by our approach. Implementation is underway. 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 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 documents must be taken during the workflow execution.

References

- [1] E. Bertino, S. Castano and E. Ferrari, Author- $\mathcal{X}$: A comprehensive system for securing XML documents, *IEEE Internet Computing*, vol. 5(3), pp. 21-31, 2001.
- [2] E. Bertino and E. Ferrari, Secure and selective dissemination of XML documents, *ACM Transactions on Information and Systems Security*, vol. 5(3), pp. 290-331, 2002.

- [3] E. Bertino, E. Ferrari and G. Mella, A framework for distributed and cooperative updates of XML documents, Technical Report DSI n.27802, University of Milan, 2002.
- [4] H. Gladney and J. Lotspiech, Safeguarding digital library contents and users: Assuring convenient security and data quality, *D-lib Magazine*, 1997.
- [5] B. Thuraisingham, A. Gupta, E. Bertino, E. Ferrari, Collaborative commerce and knowledge management across borders, *Knowledge and Process Management*, vol. 9(1), pp. 43-53, 2002.

Table 4. Components of control data structures for modifiable regions.

Component	Sub-component	Semantics
r_id	-	identifier of a modifiable region of 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 s_{last-1}
	$prev_r_dig_h$	hash value computed over hash values corresponding to atomic elements in the document version created by s_{last-1} and belonging to r_id
	$certificates$	if subject s_{last-1} has modified the region r_id , it contains authoring certificates and sets of atomic elements, it is <i>null</i> 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 ,
s_{last-1}	-	subject that verified and/or operated over r_id just before s_{last}
$h_c_s_{last}$	$last_r_h$	like $prev_r_h$, but referred to s_{last}
	$last_r_dig_h$	like $prev_r_dig_h$, but referred to s_{last}
	$certificates$	like $certificates$, but referred to s_{last}
	$cycle_path_{last}$	like $cycle_path_{last-1}$, but referred to 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}	-	like h_serv_{last-1}
ae_id	-	identifier of the atomic element ae
$position$	-	localization of ae 's components in d
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