Secure and Selective Dissemination of XML Documents

ELISA BERTINO
University of Milano
and
ELENA FERRARI
University of Insubria

XML (*eXtensible Markup Language*) has emerged as a prevalent standard for document representation and exchange on the Web. It is often the case that XML documents contain information of different sensitivity degrees that must be selectively shared by (possibly large) user communities. There is thus the need for models and mechanisms enabling the specification and enforcement of access control policies for XML documents. Mechanisms are also required enabling a secure and selective dissemination of documents to users, according to the authorizations that these users have. In this article, we make several contributions to the problem of secure and selective dissemination of XML documents. First, we define a formal model of access control policies for XML documents. Policies that can be defined in our model take into account both user profiles, and document contents and structures. We also propose an approach, based on an extension of the Cryptolope$^{\text{TM}}$ approach [Gladney and Lotspiech 1997], which essentially allows one to send the same document to all users, and yet to enforce the stated access control policies. Our approach consists of encrypting different portions of the same document according to different encryption keys, and selectively distributing these keys to the various users according to the access control policies. We show that the number of encryption keys that have to be generated under our approach is minimal and we present an architecture to support document distribution.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*access controls*; H.2.7 [**Database Management**]: Database Administration—*security, integrity, and protection*

General Terms: Security

Additional Key Words and Phrases: Access control, secure distribution, XML

This work was partially supported by a grant from Microsoft Research.

Authors' addresses: E. Bertino, Dipartimento di Scienze dell'Informazione, University of Milano, Via Comelico 39/41, 20135 Milano, Italy, email: bertino@dsi.unimi.it; E. Ferrari, Dipartimento di Scienze Chimiche, Fisiche e Matematiche, University of Insubria, Como, Via Valleggio, 11, 22100 Como, Italy; email: elena.ferrari@uninsubria.it.

1. INTRODUCTION

Companies and organizations are today massively using the Web as the main information distribution means both at internal and external level. Such a widespread use of the Web has pushed the rapid development of suitable standards for information representation. XML (*eXtensible Markup Language*) [W3C 1998] is currently the most prevalent standardization effort in the area of document representation through markup languages. The main advantages of XML, compared with HTML, are the possibility of defining tags, nested document structures, and document types (called DTDs—*Document Type Definitions*—or XMLSchemas) that describe the structure of documents.

As it is very often the case in most organizations, whenever one has large amounts of data to be shared among users, one must put in place suitable access control policies. In general, an access control policy states which subjects can access which objects under which modes. Once access control policies are stated, they are implemented by an access control mechanism. Because XML is becoming the most prevalent means according to which documents and data are encoded for distribution among users on the Web, there is a strong need for models and mechanisms enabling the specification and enforcement of access control policies for XML documents. Such models and mechanisms are crucial in order to facilitate a selective dissemination of XML documents, containing information of different sensitivity levels, among (possibly large) user communities. Their development poses, however, several requirements.

A first requirement is the support for varying protection granularity levels. In some cases, the same access control policy may apply to a set of documents. In other cases, different access control policies may apply to different components within the same document. Many other intermediate situations may also arise. The access control mechanism must be flexible enough to support a spectrum of protection granularity levels.

A second related requirement is the support for content-based access control. Content-based access control allows one to express authorization rules that take document contents into account. This is an important requirement since very often documents with the same type and structure have contents of different sensitivity degrees. In order to support content-based access control, access control policies must allow one to include conditions against document contents.

A third requirement is related to the heterogeneity of subjects. The population of users accessing XML document sources is often composed of subjects characterized by different skills and needs. Moreover, the population is dynamic, in that the number and type of subjects is not known a priori and can very frequently change over time. This requires access control policies based on user characteristics and qualifications, rather than based on very specific and individual characteristics (e.g., user IDs).

A last requirement is to provide mechanisms enabling a secure and selective dissemination of XML documents to users, according to the authorizations these users have. These mechanisms are particularly crucial when an *information push* approach is used for distributing documents to users. Under such an approach, a server periodically (or whenever some relevant event arises) broadcasts data to clients. Consider, for example, the case of a newsletter sent once a week to all users. In general, since different users may have privilege to see different, selected portions of the same document, supporting an information push approach may entail generating different physical views of the same document and sending them to the proper users. The number of such views may become rather large and thus such an approach cannot be practically applied.

In this article, we propose an access control model and a related mechanism addressing the above requirements. The model supports varying protection granularity levels and content-based access control. The last functionality is achieved by integrating into our model the query language proposed in Deutsch et al. [1999], in order to express conditions on document contents. In addition, our model supports the specification of policies associated with single documents, with collections of documents, and with DTDs. The last feature allows one to associate the same access control policy with all instances of a given DTD. Moreover, given the hierarchical, interlinked structure of XML documents and DTDs, we support different propagation options for our access control policies which state how a policy specified at a given level in a document/DTD hierarchy propagates to lower levels, thus allowing a concise expression of many security requirements. To better take into account user profiles in the formulation of access control policies, our model supports the use of *credentials*. A credential is a set of attributes concerning a user that are relevant for security purposes (e.g., age, position within the organization, projects the user is working on). The use of credentials allows the Security Administrator (SA) to directly express relevant security 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." Finally, to support document dissemination to large communities of users, we propose a document distribution approach based on an extension of the Cryptolope^{TM} approach [Gladney and Lotspiech 1997]. Our approach basically consists in encrypting different portions of the same document according to different encryption keys, and selectively distributing these keys to the various users according to the access control policies. We show that the number of encryption keys that have to be generated under our approach is minimal. Additionally, we present an architecture supporting information push.

The work reported in this article builds on an access control mechanism for XML documents proposed by us in Bertino et al. [2001c]. The current paper significantly extends our previous work for what concerns both the access control model and system architecture. Major extensions to the access control model are the possibility of expressing access control policies based on subject credentials and on the content of XML documents, whereas in the access control model proposed in Bertino et al. [2001c] policies are only specified in terms of user identifiers and document structures. This is a major extension for the flexibility it allows in specifying access control policies. Such flexibility is of particular relevance in an open and dynamic environment like the Web.

From the architectural point of view, the work reported in this article extends our previous work with the push mode for access control. Indeed, our previous work only supports information pull. Under such a mode (which is the one commonly used in conventional DBMSs), a subject explicitly submits an access request to the system when he/she wishes to access a document. The system checks the authorizations the subject possesses and, on the basis of these authorizations, it returns the subject a view of the requested document containing all and only those portions for which the subject has a proper authorization. The extension proposed in this article is a major extension both for the practical relevance of information push in distributed environments and for the involved theoretical and performance issues. In the current article, we address all these issues by proposing a method to support information push, which is based on a selective encryption of documents, by proving the correctness and the minimality of such method, and by developing an architecture and related algorithms for its support.

In this article, we use examples from Global Legal Information Network (GLIN) [Adam et al. 2002], a project undertaken by the Law Library of Congress (LLoC) in 1993. The broad goal of GLIN is to create a knowledge base of international laws and to make this knowledge base available to member countries from around the world (about 35 and growing). In GLIN, only a certain group of users such as members of official government agencies or U.S. Congress can gain access to certain documents. For example, the Law Library of Congress publishes monthly a *World Law Bulletin*, which includes a *blue page report* that provides a report about pending legislation which is a particular topic of interest to Congress. This report should not be public until it is open for public discussion because this could be a gold mine to lobbyists. Moreover, only certain individuals are allowed to view or generate some parts of this report. Although all our examples in this article are from the GLIN environment, our system can be applied in other domains as well.

The remainder of this article is organized as follows. Section 2 presents the basic concepts of XML. In Section 3, we show how access control policies for XML documents can be specified in our system, and we give the formal semantics of these policies. Section 4 presents the mechanisms we have developed for a controlled distribution of XML documents, according to the specified policies. Section 5 surveys related work. Finally, Section 6 concludes the article and outlines future research directions. Appendix A summarizes the notation and terminology used throughout the article, whereas formal proofs are reported in Appendix B.

2. BASIC CONCEPTS OF XML

The basic concept of an XML document is the *element*. Elements can be nested at any depth and can contain other elements (*subelements*). An element contains a portion of the document delimited by two *tags*: the *start tag*, at the beginning of the element, with the form <*tag-name*>, and the *end tag*, at the end of the element, with the form </*tag-name*>, where *tag-name* indicates the type of the element (*markup*).

```
<WorldLawBulletin Date="8/8/2000">
    <Law Country="USA" RelatedLaws="LK75">
        <Topic> Taxation </Topic>
        <Summary > ... </Summary>
    </Law>
    <Law Id ="LK75" Country="Italy" >
        <Topic> Import-Export </Topic>
        <Summary > ... </Summary>
    </Law>
    <BluePageReport>
        <Section GeoArea="Europe">
            <Law Country="Germany">
                <Topic> Guns </Topic>
                <Summary > ... </Summary>
            </Law>
            ...
        </Section>
        <Section GeoArea="NorthAmerica">
            <Law Country="USA">
                <Topic> Transportation </Topic>
                <Summary > ... </Summary>
            </Law>
            ...
        </Section>
    </BluePageReport>
</WorldLawBulletin>
```

Fig. 1. An example of XML document.

An example of XML document is shown in Figure 1. The document is a bulletin that provides information on laws all over the world. It contains a special section, that is, the Blue Page Report, for pending legislation. For each law, the document provides information on the country which issues the law, on the law topic and on related laws. Moreover, it provides a brief summary of the law. The Blue Page Report contains a section for each different geographic area. This section contains a list of pending laws for that area. With reference to Figure 1, the `WorldLawBulletin` element is an example of *document element*, that is, the outermost element containing all the elements of the document. `Law`, `Topic`, and `BluePageReport` are examples of elements at different depth in the hierarchical structure of the XML document. `Section` is an example of element with subelements in that it contains `Law`. `Summary` is an example of element containing text (*data content*). Elements can also be empty. Empty elements are characterized only by a start tag and do not contain data content or subelements.

The start tag of each element can specify a list of *attributes* providing additional information for the element (e.g., the start tag of the `Law` element in Figure 1). 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` referring to a single target or `IDREFS` referring to multiple targets). To be referenceable, an element must have an attribute of type `ID` whose value provides a unique identifier. Other elements can reference it through attributes of type `IDREF`(s).

In the following, we introduce a formal model of XML document (based on Deutsch et al. [1999] and Milo and Zohar [1998]) and of the related concept of DTD. Let $\mathcal{I}_{\mathcal{E}}$ be a set of element identifiers, *Label* a set of element tags and attribute names, and *Value* a set of attribute/element values. An XML document can be formally defined as follows:

Definition 2.1 (*XML document*). An *XML document* is a tuple $d = (V_d, \bar{v}_d, E_d, \phi_{E_d})$, where:

—$V_d = V_d^e \cup V_d^a$ is a set of nodes representing elements and attributes, respectively. Each $v \in V_d^e$ has an associated element identifier $id_v \in \mathcal{I}_{\mathcal{E}}$, whereas each $v \in V_d^a$ has an associated value $val \in Value$;

—$\bar{v}_d$ is a node representing the document element (called *document root*);

—$E_d = E_d^e \cup E_d^a \subseteq V_d \times V_d$ is a set of edges, where $e \in E_d^e$ is an edge representing an element-subelement relationship or a link between elements due to IDREF(s) attributes (called *link edge*), and $e \in E_d^a$ is an edge representing an element-attribute relationship;

—$\phi_{E_d}: E_d \rightarrow Label$ is the edge labeling function.

According to Definition 2.1, an XML document can be seen as a labeled graph where nodes represent elements and attributes, and edges represent relationships between them. 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.

A node representing an attribute contains its associated value. For simplicity, 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 contains edges representing the element-attribute and the element-subelement relationships, and link edges, representing links between elements introduced by attributes of types IDREF.[1] Edges are labeled with the tag of the destination node (i.e., an element or an attribute) and are represented by solid lines, whereas link edges are labeled with the name of the corresponding IDREF attribute and are represented by dashed lines.

Figure 2 shows the graph representation of the XML document in Figure 1. In the figure, nodes representing elements are denoted by white circles, whereas nodes representing attributes are denoted by gray circles. Nodes representing elements have inside the corresponding *id* (system defined *id* are represented as &*n*, where *n* is a natural number). In the following, since an element/attribute is represented as a node in a graph, we use indifferently the terms *element/attribute* and *node*.

A *document type declaration* can be attached to XML documents, specifying the rules that XML documents may follow. These rules are collectively known as the *Document Type Definition* (DTD). A DTD is composed of two parts: the

[1] An attribute of type IDREFS is considered as a list of attributes of type IDREF.

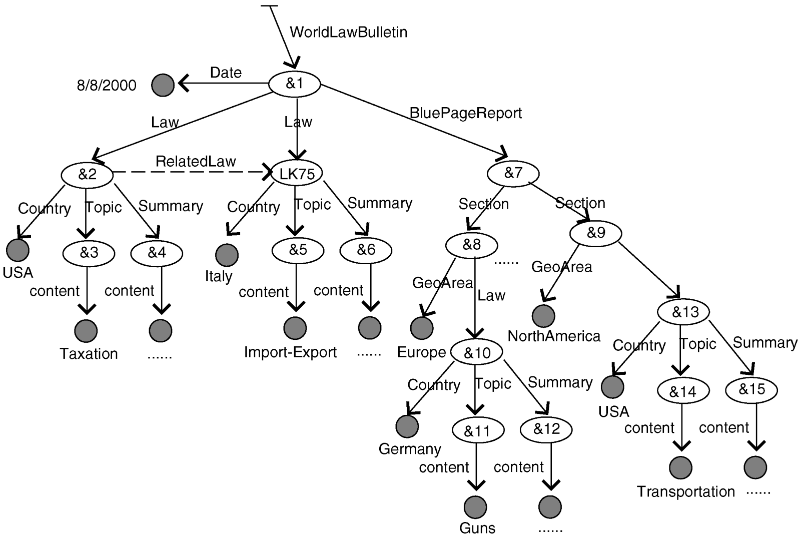

Fig. 2. Graph representation of the XML document in Figure 1.

```
<!DOCTYPE WorldLawBulletin[
    <!ELEMENT WorldLawBulletin(Law*,BluePageReport?)>
    <!ELEMENT Law(Topic,Summary)>
    <!ELEMENT Topic (#PCDATA)>
    <!ELEMENT Summary ANY>
    <!ELEMENT BluePageReport (Section+)>
    <!ELEMENT Section(Law+)>
    <!ATTLIST WorldLawBulletin Date CDATA>
    <!ATTLIST Law Id ID Country CDATA
                    RelatedLaws IDREFS>
    <!ATTLIST Section GeoArea CDATA >
]>
```

Fig. 3. DTD for the document in Figure 1.

element declarations and the *attribute list declarations*. The element declarations part specifies the structure of the elements contained in the document. In particular, for an element it specifies its subelements (if any) and their order and/or the type of the element data content. This type may be `EMPTY`, if no content is allowed, `ANY` if all kind of content is allowed, or `#PCDATA` if only data content is allowed. The attribute list declarations part specifies, for each element, the list of its attribute names, types, optionality clauses (`#IMPLIED` to denote an optional attribute, `#REQUIRED` to denote a mandatory one), and (possibly optional) default values. Figure 3 shows the DTD for the document in Figure 1. For each element, the DTD contains information on its subelements and attributes. Moreover, it is possible to specify the order of subelements, whether they are optional ('?'), whether they may occur multiple times ('$*$' or '$+$' with the usual meaning), and whether subelements are alternative with respect to each other ('|').

Let $\mathcal{I}_{\mathcal{T}_{\mathcal{E}}}$ be a set of DTD element identifiers, and $Label^*$ be the set of strings obtained through the concatenation of names in $Label$ and a symbol in $\{*, +, ?\}$.

Let *Type* be a set of types, $Type = \{\texttt{EMPTY}, \texttt{ANY}, \texttt{\#PCDATA}, \texttt{ID}, \texttt{IDREF}, \texttt{IDREFS}, \texttt{CDATA}\}$. A DTD is formally defined as follows:

Definition 2.2 (*DTD*). A *document type definition* (*DTD*) is a tuple $t = (V_t, \bar{v}_t, E_t, \phi_{E_t})$, where:

—$V_t = V_t^e \cup V_t^a$ is a set of nodes representing elements and attribute types, respectively. Each $v \in V_t^e$ has an associated DTD element identifier $id_t \in \mathcal{I}_{\mathcal{T}_\mathcal{E}}$, whereas, each $v \in V_t^a$ has an associated type $t \in Type$;

—$\bar{v}_t$ is the node representing the whole DTD element (called *DTD root*);

—$E_t \subseteq V_t \times V_t$ is a set of edges, where $e \in E_t$ represents the element–subelement or element–attribute relationship;

—$\phi_{E_t} : E_t \rightarrow Label^* \cup \{\texttt{union}, \texttt{content}\}$ is the edge labeling function.

DTDs can be represented as a graph, using a representation similar to those of XML documents.

An XML source is a set of XML documents and DTDs. Two kinds of documents can be found in an XML source, namely, *valid* and *well-formed* documents. A well-formed document follows the grammar rules of XML [W3C 1998] but does not have an associated DTD, whereas a valid document is a document conforming to a given DTD. Therefore, valid documents can be considered instances of a corresponding DTD in the source. For example, the document in Figure 1 is a valid document, since it conforms to the DTD in Figure 3. We use notation *Inst*(*dtd_id*) to denote the identifiers of documents instance of the DTD with identifier *dtd_id*.

Throughout the article, we assume that an XML source $\mathcal{S}$ is given.

3. ACCESS CONTROL POLICIES FOR XML DOCUMENTS

Access control policies regulate access to documents stored in a given source. Access control policies for XML sources must cope with a dynamic subject population, often making accesses from remote locations, and they must support a wide spectrum of protection granularities, ranging from a set of documents to specific elements within a document. Consequently, we base access control policies on the notion of subject credentials and on policy specifications at different granularity levels, with different propagation options. In what follows, we introduce the basic components on which policy specification relies. We then give the syntax and semantics of access control policy specifications.

3.1 Subject Specification

Access control policies are specified in terms of subject profiles, based on the notion of *credential*. A credential is a set of subject attributes that are needed for security purposes. Each subject has one or more associated credentials. We assume that credentials 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*.

To formalize the concepts of credentials and credential types, we use $\mathcal{AN}$ to denote a set of attribute names, $\mathcal{T}$ to denote the set of possible types of attributes in $\mathcal{AN}$, and $\mathcal{V}$ to denote the set of legal values for types in $\mathcal{T}$. Moreover, we denote

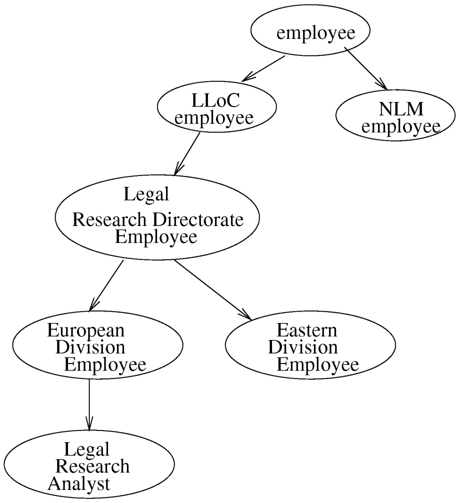
employee
LLoC employee
NLM employee
Legal Research Directorate Employee
European Division Employee
Eastern Division Employee
Legal Research Analyst

Fig. 4. An example of credential type hierarchy.

the set of credential type identifiers with $\mathcal{CT}$ and the set of credential identifiers with $\mathcal{CI}$. Finally, we use S to denote a set of subject identifiers. Credential types are formally defined as follows:

Definition 3.1 (*Credential type*). A *credential type* is a pair (*ct_id*, *attr*), where $ct\_id \in \mathcal{CT}$ is the credential type identifier, and *attr* is a set of pairs (*a_name*, *a_dom*), where $a\_name \in \mathcal{AN}$ is the attribute name (attribute names must be distinct within a given credential type) and $a\_dom \in \mathcal{T}$ is the attribute domain.

Example 3.1. The following is an example of credential type:

(employee, {(age,string), (address,string),(salary,integer),
(nationality,string), (national origin,string)}).

Credential types are organized into a hierarchy, referred to as *credential type hierarchy*, according to a partial order $\prec_{\mathcal{CT}}$. Given, two credential types ct_1 and $ct_2 \in \mathcal{CT}$ we say that ct_2 is a *subcredential type* (or shortly subtype) of ct_1 if and only if $ct_2 \prec_{\mathcal{CT}} ct_1$. Each credential type contains all the attributes of the credential types preceding it in the hierarchy. Moreover, it can contain additional attributes, apart from the inherited ones. Given a credential type $ct \in \mathcal{CT}$, $A(ct)$ denotes the set of attributes of instances of ct.[2] Figure 4 gives an example of credential type hierarchy for the GLIN domain.

A credential can be formally defined as follows:

Definition 3.2 (*Credential*). A *credential* c is a tuple (*c_id*, *sbj_id*, *state*, *ct_id*), where $c\_id \in \mathcal{CI}$ is the credential identifier; $sbj\_id \in S$ id the identifier of the subject to which the credential refers; $state = (a_1 : v_1, \ldots, a_n : v_n)$, where $a_1, \ldots, a_n \in A(ct\_id)$ are the names of the attributes of c, $v_1, \ldots, v_n \in \mathcal{V}$ are their

[2] Here, we use the terminology of object-oriented data models. Thus, a credential is an instance of a credential type ct and is in turn a member of all credential types ct', such that $ct \prec_{\mathcal{CT}} ct'$.

values, and $ct\_id \in \mathcal{CT}$ is the identifier of the credential type of which c is an instance.

Example 3.2. The following is an example of credential for an employee:

(c_1,Bob,(age:null,address:Queen Street, salary:70k,nationality:US, national origin:Italy),employee).

The *Credential Base*, denoted $\mathcal{CB}$, is the set of credentials associated with subjects in $\mathcal{S}$.

We have defined an XML-based language for encoding credentials and credential types. For simplicity, we do not report it here, details can be found in Bertino et al. [2001b]. The language provides a uniform syntax for both documents and credentials. We can therefore apply our access control policies to credentials themselves. For instance, some credential attributes (such as the user name) may be made accessible to everyone, whereas other attributes may be visible only to a restricted class of subjects. Additionally, the use of an XML language for specifying credentials facilitates secure credential submission and distribution, in that, to guarantee credential authenticity and integrity, we can apply the digital signatures and encryption mechanisms we have developed for XML documents.

In our model, subjects to which a policy applies are implicitly defined by imposing a set of conditions on their credentials. Conditions on credentials are expressed by means of a formal language. Expressions that can be specified in the language, referred in the following as *credential expressions*, are formally defined as follows:

Definition 3.3 (*Credential Expression*). The set $\mathcal{CE}$ of credential expressions is defined as follows:

—each element in $\mathcal{CT}$ is a credential expression;

—if $a \in \mathcal{AN}$, $v \in \mathcal{V}$, and $\text{OP} \in \{=, \neq, >, <, \geq, \leq, \in, \notin, \subseteq, \nsubseteq, \subset, \not\subset, \supset, \supseteq, \not\supset, \nsupseteq\}$, then $a \ \text{OP} \ v$ is a credential expression;[3]

—if $ce_1, ce_2 \in \mathcal{CE}$, then $(ce_1 \wedge ce_2)$ and $(ce_1 \vee ce_2)$ are credential expressions.

Example 3.3. The following are examples of credential expressions in $\mathcal{CE}$:

—(European Division Employee $\vee$ LLoC Employee): this expression denotes subjects with a credential of type European Division Employee or LLoC Employee;

—(employee $\wedge$ age $>$18): this expression denotes the employees whose age is greater than 18.

3.2 Object Specification

Access control policies are specified both at the DTD and at the document level. Policies can be specified for a whole DTD(s) (respectively, document(s))

[3] The operators that can be used in an expression $a \ \text{OP} \ v$ are actually a subset of the ones listed above, since they depend on the type of a. We assume that there is some mechanism in place to verify that only meaningful operators are used.

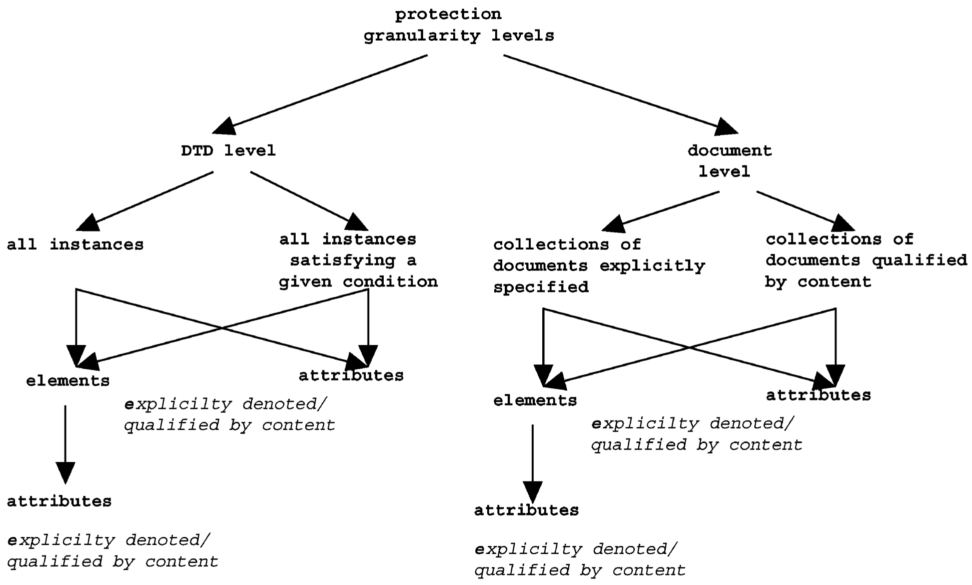
protection granularity levels
DTD level
document level
all instances
all instances satisfying a given condition
collections of documents explicitly specified
collections of documents qualified by content
elements
attributes
explicilty denoted/ qualified by content
elements
attributes
explicilty denoted/ qualified by content
attributes
explicilty denoted/ qualified by content
attributes
explicilty denoted/ qualified by content

Fig. 5. Taxonomy of protection granularity levels.

or for selected elements and/or attributes within a DTD(s) (respectively, document(s)). Elements/attributes can be either explicitly specified, or can be selected on the basis of their content.

Policies specified at the DTD level apply to valid documents in that they propagate to all DTD instances, or only to instances which satisfy a given condition. Additionally, policies can be specified which apply to collections of both valid and well-formed XML documents. A taxonomy of the protection granularity levels supported by our access control mechanism is reported in Figure 5.

In the following, we use the term, *protection object*, to denote the documents (or portions of documents) to which a policy refers. The specification of protection objects requires three steps: (1) the specification of one or more documents or DTDs; (2) the specification of selected element(s) within the specified DTD(s)/document(s); (3) the specification of selected attribute(s) within the specified element(s). Steps (2) and (3) are optional.

Elements can be denoted by listing their identifiers, or by giving a path from the DTD/document root to the node(s) representing the element(s). These possibilities are formalized by the notion of *element specification*, defined as follows:

Definition 3.4 (*Element Specification*). Let $d = (V, \bar{v}, E, \phi_E)$ be a DTD or a document. An element specification for d, denoted by `element-spec`, is:

—a set of element identifiers in d, that is, `element-spec` $= \{id_1, \ldots, id_n\}$, with $id_i \in \{id_v \mid v \in V^e \setminus \{\bar{v}\}\}$, or

—a path expression, that is, `element-spec` $= path\_expr$, where $path\_expr$ is specified according to the following grammar: $path\_expr$ $::= * \mid tag \mid$ $path\_expr.\,path\_expr$, where tag is an edge label corresponding to a tag of an element in d.

Example 3.4. `Law.Topic`, {`LK75`, `&47`}, `BluePageReport.Section` are examples of element specifications for the document in Figure 2.

In the following, we denote with $\mathcal{PE}$ the set of path expressions for source $\mathcal{S}$. Moreover, let $pe \in \mathcal{PE}$ be a path expression, then *Eval*(pe) denotes the set of identifiers of the elements denoted by pe. We are now ready to introduce the definition of *protection object specification*.

Definition 3.5 (*Protection Object Specification*). A protection object specification is of the form:

`DTDs-spec | docs-spec.[elements-spec].[attrs]`[4]

where:

—`DTDs-spec` $\in \mathcal{I}_{\mathcal{T}_\mathcal{E}} \cup \{\#\}$ is a DTD root identifier or the symbol #. Symbol # is used to denote all the DTDs in $\mathcal{S}$;

—`docs-spec` $\in 2^{\mathcal{I}_\mathcal{E}}$ is a set of document root identifiers;

—`elements-spec` is an element specification for the DTDs (respectively, documents) denoted by `DTDs-spec` (respectively, `docs-spec`). If `DTDs-spec` = #, then `elements-spec.attrs` must be omitted;

—`attrs` is a set of attributes of the elements denoted by `elements-spec`, if `elements-spec` is not omitted; it is a set of attributes of the DTDs (respectively, documents) denoted by `DTDs-spec` (respectively, `docs-spec`), otherwise.

In the following, we denote with $\mathcal{PO}$ the set of protection object specifications for source $\mathcal{S}$.

A protection object specification that contains a DTD identifier, or the symbol #, implicitly denotes a set of valid XML documents (or parts of them). Specifically, it denotes *all the documents* (*or portions*) *that are instances of the DTDs denoted by the* `DTDs-spec` *component of the specification*. However, specifying policies that always apply to all DTDs instances is not always reasonable in practice. For instance, consider two collections of XML documents: *projects* and *secret-projects*, both with the same DTD, but with different security requirements (documents belonging to the secret-project collection require more restrictive access control policies). This means that different policies should be applied to the two collections. Moreover, different portions of the same document may have different protection requirements depending on their content. As an example, consider the document in Figure 2 and a policy stating that *Laws pertaining to a given country contained in the* `BluePageReport` *section can be made available only to subjects belonging to that country* (since these laws are pending). To support the possibility of expressing both the above protection requirements, we associate conditions with protection object specifications: the policy is thus applied to all the documents (or portions of them) denoted by the specification *that satisfy* the specified condition. In the following, $\mathcal{C}$ denotes the set of conditions that can be expressed in our model. In the current

[4] Square brackets denote optional components, whereas symbol '|' denotes alternative components.

version of our access control mechanism, the language to express conditions is based on the query language proposed in Deutsch et al. [1999]. However, other languages could also be used as well.

Definition 3.6 (*Document Specification*). A document specification is a pair (C, po), where $C \in \mathcal{C} \cup \{true\}$ is a condition, and $po \in \mathcal{PO}$ is a protection object specification.

If a document specification has a *true* condition, then it denotes all documents (or portions of documents) identified by its protection object specification. The set of document specifications for source $\mathcal{S}$ is denoted with $\mathcal{DS}$.

Example 3.5. Consider the DTD in Figure 3 and the XML document in Figure 2. The following are examples of document specifications:

— (GeoArea = Europe,WorldLawBulletin.BluePageReport.Section): it denotes the section pertaining to Europe of the BluePageReport of all the instances of WorldLawBulletin;

— (Topic ≠ Guns,WorldLawBulletin.Law): it denotes the laws contained in all the instances of WorldLawBulletin (apart from those contained in the BluePageReport element) whose topic is not Guns;

— (true,&1.{LK75}): it denotes the element with identifier LK75.

3.3 Privilege Specification

Our model supports two categories of privileges: *browsing* and *authoring* privileges.

Browsing privileges allow subjects to read the information in a document (or in some of its elements) and/or to see the relations occurring among subelements (defined trough IDREF(s) attributes). Three browsing privileges are supported: view, navigate, and browse_all. The view privilege authorizes a subject to read the values of all (or some of) the attributes of a document (or of some of its elements), apart from attributes of type IDREF(s). For IDREF(s) attributes, the navigate privilege is introduced: if a subject has the navigate privilege on a document (respectively, element), the subject can see all the semantic relations implied by the attributes of type IDREF(s) contained in the document (respectively, element). The navigate privilege can also be given on selected attributes of a document and/or elements. Clearly, the view the subject has on the referenced elements depends on the authorizations the subject has on such elements. The navigate privilege is thus equivalent to a view privilege given on IDREF(s) attributes only. However, the distinction between view and navigate privilege makes it possible to grant subjects the access to an element without disclosing the links of this element with other elements. For example, if a subject has the view privilege on the document in Figure 2, he/she can view all the laws contained in the document, but he/she cannot see the relations of a law with other laws in the document. To acquire this information, the subject needs a navigate privilege on the document. Finally, the browse_all privilege subsumes both the navigate and the view privilege, thus allowing a compact specification of access control policies.

Authoring privileges allow subjects to modify (or delete) an element/attribute or a document. We support three distinct authoring privileges: `append`, `write`, and `auth_all`. The `append` privilege allows a user to write information in an element (or in some of its attributes), without deleting any preexisting information. By contrast, the `write` privilege allows a user to modify the content of an element/attribute, possibly deleting them. Thus, the `write` privilege subsumes the `append` privilege. Finally, the `auth_all` privilege subsumes all the other authoring privileges.

The set of all browsing and authoring privileges is denoted by $\mathcal{P}$.

3.4 Propagation Options

Different *propagation options* can be exploited in the specification of an access control policy. Propagation options state how policies specified at a given level of a DTD/document hierarchy propagate (partially or totally) to lower levels. We use a natural number `n` or the special symbol '$*$' to represent the propagation option associated with a given policy. This number indicates the "depth" of the propagation, in that:

—if propagation option is equal to $*$, then the policy propagates to all the *direct and indirect* subelements of the element(s) specified in the policy specification;

—if propagation option is equal to `0`, then no propagation of the policy is enacted;

—if propagation option is equal to `n`, $\mathtt{n} \geq 1$, then the policy propagates to the subelements of the element(s) specified in the policy specification, which are at most `n` levels down in the document/DTD hierarchy.

3.5 Policy Specification

Based on the components we have previously introduced, access control policies for XML documents can now be formally defined as follows:

Definition 3.7 (*Access Control Policy*). An *access control policy* is a tuple (`subject-spec`, `document-spec`, `priv`, `prop-opt`), where: `subject-spec` $\in \mathcal{CE}$ is a credential expression; `document-spec`$\in \mathcal{DS}$ is a document specification; `priv` $\in \mathcal{P}$ is a privilege; `prop-opt`$\in \mathbb{N} \cup \{*\}$ is the propagation option.

Moreover, the Policy Base, denoted as $\mathcal{PB}$, is the set of access control policies defined for source $\mathcal{S}$.

Example 3.6. The following are examples of access control policies:[5]

—$\mathtt{P}_1$ = `((LLoC Employee` $\vee$ `European Division Employee),WorldLawBulletin.Law,browse_all,*)`: this policy authorizes the LLoC and European Division employees to view the laws (not contained in the `BluePageReport` element) in all the instances of `WorldLawBulletin`. Relations among laws (i.e., the `RelatedLaws` attributes, are also displayed);

[5] For simplicity, we omit *true* conditions from document specifications.

—P$_2$ = ((LLoC Employee ∨ European Division Employee),&1,view,0): this policy authorizes LLoC and European Division employees to read the attributes of the element with identifier &1 of the document in Figure 2;

—P$_3$ = (NML Employee, WorldLawBulletin.Law,view,*): this policy authorizes the NML employees to view the laws (not contained in the BluePageReport element) in all the instances of WorldLawBulletin. Relations among laws are not displayed. For instance, with reference to the XML document in Figure 2, this policy allows an NML employee to view element &2 and LK75, but not the link (denoted in the figure by the dashed line) between the two;

—P$_4$ = (European Division Employee,(GeoArea = Europe,WorldLawBulletin. BluePageReport.Section),browse_all,*): this policy authorizes European Division employees to view the section pertaining to Europe of the BluePageReport in all the instances of the WorldLawBulletin;

—P$_5$ = (LLoc Employee, WorldLawBulletin.Law,auth_all,*): this policy authorizes LLoC Employee to modify the laws (not contained in the BluePageReport element) in all the instances of the WorldLawBulletin.

Clearly, there are some restrictions on policy specifications. For instance, policies which are defined for selected attributes in a document(s) or element(s) must have the propagation option equal to 0, whereas policies for browse_all and auth_all privileges apply only to documents and elements, not to attributes. Finally, the view privilege cannot be granted on an IDREF(s) attribute, whereas the navigate privilege cannot be granted on a non IDREF(s) attributes. These conditions are checked at the time a policy is specified and the insertion of the policy is accepted only if the conditions are satisfied.

In what follows, given an access control policy acp, we use subject-spec(acp), document-spec(acp), priv(acp), and prop-opt(acp) to denote, respectively, the credential expression, the document specification, the privilege, and the propagation option in acp.

3.6 Policy Semantics

With the term *policy semantics*, we denote the set of access authorizations entailed by a given access control policy. Access authorizations can be represented as a triple (s , o , p), where $s \in S$ is a subject, $p \in \mathcal{P}$ is a privilege, and o is an object denoting a document or a portion of a document (i.e., an element or attribute). Thus, o can have one of four different formats:

(1) $o = d\_id$, where $d\_id \in \mathcal{I}_{\mathcal{E}}$ is a document root identifier;

(2) $o = d\_id.a$, where $d\_id \in \mathcal{I}_{\mathcal{E}}$ is a document root identifier, and $a \in Label$ is the name of an attribute in d_id;

(3) $o = d\_id.e\_id$, where $d\_id \in \mathcal{I}_{\mathcal{E}}$ is a document root identifier, and $e\_id \in \mathcal{I}_{\mathcal{E}}$ is the identifier of an element in d_id;

(4) $o = d\_id.e\_id.a$, where $d\_id \in \mathcal{I}_{\mathcal{E}}$ is a document root identifier, $e\_id \in \mathcal{I}_{\mathcal{E}}$ is the identifier of an element in d_id, and $a \in Label$ is the name of an attribute in e_id.

Table I. Credential Expression Semantics.

Credential expression	Semantics
$ct \in \mathcal{CT}$	$\{s \mid (c\_id, sbj\_id, state, ct\_id) \in \mathcal{CB}$, $sbj\_id = s$, and $ct\_id = ct\}$
a OP v', $a \in \mathcal{AN}$, $v' \in \mathcal{V}$	$\{s \mid (c\_id, sbj\_id, state, ct\_id) \in \mathcal{CB}$, $sbj\_id = s$, $a \in A(ct\_id)$, and $v(a)$ OP v' holds$\}$
$ce_1 \wedge ce_2, ce_1, ce_2 \in \mathcal{CE}$	$\psi(ce_1) \cap \psi(ce_2)$
$ce_1 \vee ce_2, ce_1, ce_2 \in \mathcal{CE}$	$\psi(ce_1) \cup \psi(ce_2)$

Example 3.7. Suppose that `Ann` is a LLoC Employee. Thus, `(Ann,&1.Date, view)` is an example of authorization implied by policy $\mathtt{P}_2$ of Example 3.6.

In the following, given an access authorization `A = (s,o,p)`, we denote with `s(A)`, `o(A)`, and `p(A)`, the subject, object, and privilege in `A`.

To give the semantics of an access control policy, it is first necessary to identify the set of subjects denoted by its credential expression and the set of documents, elements and/or attributes denoted by its document specification. Subjects denoted by a credential expression are formally defined as follows:

Definition 3.8 (*Credential Expression Semantics*). Let $ce \in \mathcal{CE}$ be a credential expression. The semantics of ce, denoted as $\psi(ce)$, is defined in Table I.[6]

The semantics of a document specification can be given based on the semantics of the protection object specification it contains. In defining the semantics of a protection object specification, we make use of function $\mathcal{ID}$, defined as follows: $\mathcal{ID}(d) = d$, if $d \in \mathcal{I}_{\mathcal{T}_\mathcal{E}}$ is a DTD root identifier; $\mathcal{ID}(d) = \{d \mid d$ is the identifier of a DTD root in $\mathcal{S}\}$, if $d = \#$.

Definition 3.9 (*Protection Object Specification Semantics*). Let $po \in \mathcal{PO}$ be a protection object specification. The semantics of po, denoted as $\tau(po)$, is defined in Table II.

Based on function τ, we can now define the semantics of a document specification.

Definition 3.10 (*Document Specification Semantics*). Let $ds \in \mathcal{DS}$ be a document specification. The semantics of ds, denoted as $\delta(ds)$, is defined as follows:

—$\delta((true, po)) = \tau(po)$, $po \in \mathcal{PO}$;

—$\delta((C, po)) = \{t \mid t \in \tau(po)$, t satisfies $C\}$, $po \in \mathcal{PO}$, $C \in \mathcal{C} \backslash \{true\}$.

We are now ready to introduce the semantics of an access control policy, that is, the set of authorizations a policy entails. In defining the semantics, we make use of function $Succ^n()$, $n \in \mathbb{N} \cup \{*\}$. This function receives as input an element identifier $e\_id \in \mathcal{I}_\mathcal{E}$. If $n = *$, it returns the identifiers of all the direct and indirect subelements of e_id; if $n = 0$, it returns the empty set, whereas if $n \in \mathbb{N}$, $n \geq 1$, it returns the identifiers of the subelements of e_id that are at most n level down in the document hierarchy with respect to e_id.

[6] In the table, we use $v(a)$ to denote the value of an attribute a.

Table II. Protection Object Specification Semantics

Protection object specification	Semantics
$dtd\_id, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}} \cup \{\#\}$	$\{d\_id \mid d\_id \in Inst(d), d \in \mathcal{ID}(dtd\_id)\}$
$\{d\_id_1, \ldots, d\_id_n\}, d\_id_i \in \mathcal{I}_{\mathcal{E}}, i = 1, \ldots, n$	$\{d\_id_1, \ldots, d\_id_n\}$
$dtd\_id.\{a_1, \ldots, a_m\}, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}},$ $a_i \in Label, i = 1, \ldots, m$	$\{d\_id.a \mid d\_id \in Inst(dtd\_id),$ $a \in \{a_1, \ldots, a_m\}\}$
$\{d\_id_1, \ldots, d\_id_n\}.\{a_1, \ldots, a_m\},$ $d\_id_j \in \mathcal{I}_{\mathcal{E}}, j = 1, \ldots, n, a_i \in Label, i = 1, \ldots, m$	$\{d\_id.a \mid d\_id \in \{d\_id_1, \ldots, d\_id_n\},$ $a \in \{a_1, \ldots, a_m\}\}$
$dtd\_id.\{e\_id_1, \ldots, e\_id_m\}, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}},$ $e\_id_i \in \mathcal{I}_{\mathcal{E}}, i = 1, \ldots, m$	$\{d\_id.e\_id \mid d\_id \in Inst(dtd\_id),$ $e\_id \in \{e\_id_1, \ldots, e\_id_m\}\}$
$\{d\_id_1, \ldots, d\_id_n\}.\{e\_id_1, \ldots, e\_id_m\}, d\_id_j,$ $e\_id_i \in \mathcal{I}_{\mathcal{E}}, j = 1, \ldots, n, i = 1, \ldots, m$	$\{d\_id.e\_id \mid d\_id \in \{d\_id_1, \ldots, d\_id_n\},$ $e\_id \in \{e\_id_1, \ldots, e\_id_m\}\}$
$dtd\_id.pe, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}}, pe \in \mathcal{PE}$	$\{d\_id.e\_id \mid d\_id \in Inst(dtd\_id), e\_id \in$ $Eval(pe)\}$
$\{d\_id_1, \ldots d\_id_n\}.pe, d\_id_i \in \mathcal{I}_{\mathcal{E}}, i = 1, \ldots, n,$ $pe \in \mathcal{PE}$	$\{d\_id.e\_id \mid d\_id \in \{d\_id_1, \ldots, d\_id_n\},$ $e\_id \in Eval(pe)\}$
$dtd\_id.\{e\_id_1, \ldots, e\_id_m\}.\{a_1, \ldots, a_n\}, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}},$ $e\_id_i \in \mathcal{I}_{\mathcal{E}}, i = 1, \ldots, m, a_j \in Label, j = 1, \ldots, n$	$\{d\_id.e\_id.a \mid d\_id \in Inst(dtd\_id), e\_id \in$ $\{e\_id_1, \ldots, e\_id_m\}, a \in \{a_1, \ldots, a_n\}\}$
$\{d\_id_1, \ldots d\_id_n\}.\{e\_id_1, \ldots, e\_id_m\}.\{a_1, \ldots, a_h\},$ $d\_id_i, e\_id_j \in \mathcal{I}_{\mathcal{E}}, i = 1, \ldots, n, j = 1, \ldots, m,$ $a_l \in Label, l = 1, \ldots, h$	$\{d\_id.e\_id.a \mid d\_id \in \{d\_id_1, \ldots, d\_id_n\},$ $e\_id \in \{e\_id_1, \ldots, e\_id_m\}, a \in \{a_1, \ldots$ $\ldots, a_h\}\}$
$dtd\_id.pe.\{a_1, \ldots, a_n\}, dtd\_id \in \mathcal{I}_{\mathcal{T}_{\mathcal{E}}}, pe \in \mathcal{PE},$ $a_i \in Label, i = 1, \ldots, n$	$\{d\_id.e\_id.a \mid d\_id \in Inst(dtd\_id),$ $e\_id \in Eval(pe), a \in \{a_1, \ldots, a_n\}\}$
$\{d\_id_1, \ldots, d\_id_m\}.pe.\{a_1, \ldots, a_n\}, d\_id \in \mathcal{I}_{\mathcal{E}},$ $i = 1, \ldots, m, pe \in \mathcal{PE}, a_j \in Label, j = 1, \ldots, n$	$\{d\_id.e\_id.a \mid d\_id \in \{d\_id_1, \ldots, d\_id_m\},$ $e\_id \in Eval(pe), a \in \{a_1, \ldots, a_n\}\}$

Definition 3.11 (*Access Control Policy Semantics*). Let acp be an access control policy. The semantics of acp, denoted as $\mathcal{E}(\mathtt{acp})$, is defined as follows:

—if $\mathtt{prop\text{-}opt(acp)} = 0$: $\mathcal{E}(\mathtt{acp}) = \{(\mathtt{s,o,p}) \mid \mathtt{s} \in \psi(\mathtt{subject\text{-}spec(acp)}), \mathtt{o} \in \delta(\mathtt{document\text{-}spec(acp)}), \mathtt{p} = \mathtt{priv(acp)}\}$;

—if $\mathtt{prop\text{-}opt(acp)} = \mathtt{n}$, $\mathtt{n} \in \mathbb{N}$, $\mathtt{n} > 0$, or $\mathtt{n} = *$: $\mathcal{E}(\mathtt{acp}) = \bigcup_{\mathtt{acp}' \in Ext(\mathtt{acp})} \mathcal{E}(\mathtt{acp}')$, where $Ext(\mathtt{acp}) = \{\mathtt{acp}' \mid \mathtt{subject\text{-}spec(acp')} = \mathtt{subject\text{-}spec(acp)}, \mathtt{priv(acp')} = \mathtt{priv(acp)}, \mathtt{prop\text{-}opt(acp')} = 0$, and $\mathtt{document\text{-}spec(acp')} \in \{\mathtt{document\text{-}spec(acp)} \cup \{d.e' \mid d.e \in \delta(\mathtt{document\text{-}spec(acp)}), d, e \in \mathcal{I}_{\mathcal{E}}$, and $e' \in Succ^n(e)\} \cup \{d.e' \mid d \in \delta(\mathtt{document\text{-}spec(acp)}), d \in \mathcal{I}_{\mathcal{E}}$, and $e' \in Succ^n(d)\}\}$.

4. CONTROLLED DISTRIBUTION OF XML DOCUMENTS

Once the access control policies for a given source have been specified, XML documents belonging to the source can be released to subjects, on the basis of the specified policies. Release of XML documents can be done according to two different modalities:

—*Information pull*. Under this mode, subjects request XML documents to the source when needed. When a subject submits an access request, the access control system checks which authorizations the subject has on the requested document, according to the specified access control policies. Based on these authorizations, the subject is returned a *view* of the requested document that contains all and only those portions for which he/she has a corresponding authorization. When no authorizations are found, the access is denied.

—*Information push*. Under this mode, the source periodically broadcasts data to its clients. Also in this case, different subjects may have privileges to see different, selected portions of the same document. Thus, different views of the same document are sent to the various subjects.

Our system supports distribution of XML documents based on both the above modes. Here, we describe the mechanisms we have developed for enforcing information push. Details on information pull can be found in Bertino et al. [2001a]. Moreover, in the following we consider access control policies for browsing privileges only. However, similar methods have been developed for supporting information push for authoring policies [Bertino and Ferrari 2000].

4.1 Access Control within Information Push

The main problem in supporting access control within information push is that it entails generating different physical views of the same document and sending them to the proper subjects. Due to the possibly high number of subjects accessing an XML source, and the wide range of access granularities we provide, the number of such views may become considerable large and thus such an approach cannot be practically applied. We have therefore adopted a different solution, based on an approach similar to Cryptolope$^{\mathrm{TM}}$ [Gladney and Lotspiech 1997]. The approach consists of using different keys for encrypting different portions of the same document. Each portion is encrypted with one and only one key. The same (encrypted) copy of the document is then broadcasted to all subjects, whereas each subject only receives the key(s) for the portion(s) he/she is enabled to access.

Thus, three main issues need to be addressed: *How to encrypt the documents in a source*, that is, which portions of the documents should be encrypted with different keys; *Which keys should be distributed to which subject*, in that a subject must be given all and only the keys that allow him/her to see the portions of the document for which the subject has a corresponding authorization; and *How to distribute keys to subjects* in such a way that keys are received only by subjects that are entitled to receive them.

In the following sections, we first formally states the notion of correct encryption, then we illustrate in details how we address the above issues. Finally, we discuss the architecture of our system.

4.2 Formal Definitions

Before proceeding to illustrate our approach to secure information push, we need to formally state when a document encryption is correct. Intuitively, a document encryption is correct when: (1) each portion of the document is encrypted with one and only one key, and (2) each subject can be given the keys to access all and only those portions of the document for which the subject has a corresponding authorization, based on the specified policies.

To formally state the notion of encryption correctness, we first need to introduce the notion of *view of a document with respect to a subject*. This definition is given based on the notions of *view of a document with respect to*

Table III. View of a Document with Respect to an Authorization A

o(A)	p(A)	V_d(A)
$e\_id, e\_id \in \mathcal{I}_{\mathcal{E}}$	browse_all	the element whose identifier is e_id
	view	the element whose identifier is e_id, if e_id does not contain IDREF(s) attributes; $\bigcup_{\mathtt{A}_i \in S_A} V_d(\mathtt{A}_i)$, $S_A = \{(\mathtt{s}',\mathtt{o}',\mathtt{p}') \mid \mathtt{s}' = \mathtt{s(A)}, \mathtt{p}' = \mathtt{p(A)}, \mathtt{o}' =$ $e\_id.a$, a is a non IDREF(s) attribute in $e\_id\}$, otherwise
	navigate	the element whose identifier is e_id, if e_id does not contain non IDREF(s) attributes; $\bigcup_{\mathtt{A}_i \in S_A} V_d(\mathtt{A}_i)$, $S_A = \{(\mathtt{s}',\mathtt{o}',\mathtt{p}') \mid \mathtt{s}' = \mathtt{s(A)}, \mathtt{p}' = \mathtt{p(A)}, \mathtt{o}' =$ $e\_id.a$, a is an IDREF(s) attribute in $e\_id\}$, otherwise
$e\_id.a, e\_id \in \mathcal{I}_{\mathcal{E}}$, $a \in Label$	browse_all, view, navigate	the element whose identifier is e_id from which all the attributes, apart a, have been removed

an authorization and of *view of a document with respect to an access control policy*, defined as follows.

Definition 4.1 (*View of a Document with respect to an Authorization*). Let d be an XML document and let A = (s,o,p) be an authorization such that $\exists$acp $\in \mathcal{PB}$ with A $\in \mathcal{E}$(acp). The view of d with respect to A, denoted as V_d(A), is defined in Table III.

Definition 4.1 above states that if the authorization applies to an attribute a, then the view is equal to the element containing the attribute, from which all the attributes different from a have been removed. If the authorization applies to an element, then the view depends on the authorization privilege. If the privilege is browse_all, then the view is equal to the element. In case of a view authorization, all the IDREF(s) attributes must be removed from that view, whereas when the authorization is for a navigate privilege, the non-IDREF(s) attributes must be removed.

Based on Definition 4.1, we now introduce the definition of view of a document wrt an access control policy. This view contains the document portions that can be released under the given policy.

Definition 4.2 (*View of a Document with respect to an Access Control Policy*). Let d be an XML document and let acp $\in \mathcal{PB}$ be an access control policy. The view of d wrt acp is defined as follows:

$$V_d(\mathtt{acp}) = \bigcup_{\mathtt{A}_i \in Auth_d} V_d(\mathtt{A}_i),$$

where $Auth_d = \{\mathtt{A} \mid \mathtt{A} \in \mathcal{E}(\mathtt{acp})$ and o(A) contains the identifier of $d\}$.

Now we are ready to define the view of a document with respect to a subject, that is, the portions of a document that can be released to a subject according to the policies in the Policy Base.

Definition 4.3 (*View of a Document with respect to a Subject*). Let d be an XML document and let $s \in S$ be a subject. The view of d with respect to s is defined as follows:

$$V_d(s) = \bigcup V_d(\mathtt{acp}_i),\ \forall\ \mathtt{acp}_i \in \mathcal{PB} \text{ such that } s \in \psi(\mathtt{subject\text{-}spec(acp}_i\mathtt{)}).$$

We can now introduce the notion of correct encryption, that is, an encryption that allows a subject to access the correct view of a document.

Definition 4.4 (*Correct Encryption*). Let d be an XML document. Let d^e be an encryption of d, and let *Keys* be the set of keys used for generating d^e. d^e is said to be correct iff both the following conditions hold: 1) each portion of d^e is encrypted with one and only one key; (2) $\forall$ subject $s \in S$, $\exists K' \subseteq Keys$ such that the document obtained by decrypting d^e with keys in K' is equal to $V_d(s)$.

In what follows, we are interested in a particular class of correct encryptions, that is, those that encrypt the documents on the basis of the access control policies stored in the Policy Base. We call these encryptions *well-formed* encryptions.

Definition 4.5 (*Well-Formed Encryption*). Let d be an XML document. Let d^e be an encryption of d, and let *Keys* be the set of keys used for generating d^e. d^e is said to be well-formed iff both the following conditions hold: (1) each portion of d^e is encrypted with one and only one key; (2) $\forall$ $\texttt{acp} \in \mathcal{PB}$, $\exists K' \subseteq Keys$ such that the document obtained by decrypting d^e with keys in K' is equal to $V_d(\texttt{acp})$.

The following proposition states the correctness of well-formed encryptions.

PROPOSITION 4.1. *Each well-formed encryption is correct.*

In a well-formed encryption, there is thus a correspondence between encryption keys and the policies that apply to the document. Each access control policy *implies* a set of keys for the document (i.e., those that are needed to access the corresponding view). A well-formed encryption has the nice property of making revocation/insertion and update of access control policies easier to be managed with respect to correct encryptions. Indeed, if a well-formed encryption is used, different portions of a document are encrypted with the same key only if the same policies apply to these portions. Thus, it is easier to develop algorithms that efficiently modify the encryption of a document upon the execution of an administrative operation. In particular, we have developed algorithms [Carminati and Ferrari 2002] that incrementally maintain the document encryption by modifying all and only those portions of the encryption that are really affected by the administrative operation, without the need of reencrypting the document from scratch. We have also proved that all such algorithms preserve the well-formed property of encryptions, that is, they generate a well-formed encryption when applied to a well-formed encryption.

In the next section, we describe how to generate well-formed encryptions of XML documents.

4.3 Encryption of XML Documents

A possible solution for generating a well-formed encryption of a document is to encrypt it at the finest level granularity, that is, to encrypt every single attribute of the document or of its elements 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 to be generated, we have

```
ALGORITHM 1. The Marking Algorithm

INPUT:   An XML document $d = (V_d, \bar{v_d}, E_d, \phi_{E_d})$ in $\mathcal{S}$, the Policy Base $\mathcal{PB}$
OUTPUT:  $M_d$, a marking of document $d$, and $\Pi_d(\mathcal{PB})$

(1)   Let $\Pi_d(\mathcal{PB})$ = {acp | acp $\in \mathcal{PB}$, priv(acp) is a browsing privilege $\exists$ (s,o,p) $\in \mathcal{E}$(acp)
                such that the id of $d$ appears in o}
(2) $M_d$ is initialized to be empty
(3)   For each acp$_i \in \Pi_d(\mathcal{PB})$:
        (a) If document-spec(acp$_i$) is of the form DTDs-spec or docs-spec:
              For each $e\_id \in \{id_{\bar{v_d}}\} \cup Succ^{\texttt{prop-opt(acp}_\texttt{i}\texttt{)}}(id_{\bar{v_d}})$:
                  case priv(acp$_i$) of
                  browse-all: Add ($e\_id$,acp$_i$) to $M_d$
                  view: If $e\_id$ does not have IDREF(s) attributes: Add ($e\_id$,acp$_i$) to $M_d$
                        else: For each non IDREF(s) attribute $a$ in $e\_id$: Add ($e\_id.a$,acp$_i$) to $M_d$
                  navigate:If $e\_id$ does not have non IDREF(s) attributes: Add ($e\_id$,acp$_i$) to $M_d$
                            else: For each IDREF(s) attribute $a$ in $e\_id$: Add ($e\_id.a$,acp$_i$) to $M_d$
        (b) If document-spec(acp$_i$) is of the form DTDs-spec.elements-spec or docs-spec.elements-spec:
              Let $E'$ = {$id$ | $id$ is the identifier of an element in $d$ which is denoted by elements-spec}
              Let $E^*$ = {$e\_id$ | $e'\_id \in E'$ and $e\_id \in Succ^{\texttt{prop-opt(acp}_\texttt{i}\texttt{)}}(e'\_id)$}
              For each $e\_id \in E' \cup E^*$:
                  case priv(acp$_i$) of
                  browse-all: Add ($e\_id$,acp$_i$) to $M_d$
                  view: If $e\_id$ does not have IDREF(s) attributes: Add ($e\_id$,acp$_i$) to $M_d$
                        else: For each non IDREF(s) attribute $a$ in $e\_id$: Add ($e\_id.a$,acp$_i$) to $M_d$
                  navigate: If $e\_id$ does not have non IDREF(s) attribute: Add ($e\_id$,acp$_i$) to $M_d$
                             else: For each IDREF(s) attribute $a$ in $e\_id$: Add ($e\_id.a$, acp$_i$) to $M_d$
        (c) If document-spec(acp$_i$) denotes a set of attributes $A$:
              For each $a \in A$:
                  Let $e\_id$ be the identifier of the element to which $a$ belongs to
                  Add ($e\_id.a$,acp$_i$) to $M_d$
              endfor
    endfor
(4)   For each $e\_id \in \mathcal{I}_{\mathcal{E}}$ such that $\exists(el, ID_p) \in M_d, el = e\_id.a$:
        If $\exists(el', ID'_p) \in M_d, el' = e\_id$
          Let $A$ be the set of attributes in $e\_id$
          $temp = \emptyset$
          For each $a \in A$:
              If $\nexists(el, ID_p) \in M_d, el = e\_id.a$: Add ($e\_id.a, ID'_p$) to $M_d$
              else:
                  $temp = temp \cup ID_p$
                  Add ($e\_id.a, ID'_p$) to $M_d$
              endif
          endfor
          Replace $(el', ID'_p)$ with $(el'.\text{TAG}, ID'_p \cup temp)$ in $M_d$
        else:
            Let $S = \bigcup\{ID_p \mid (el, ID_p) \in M_d, el = e\_id.a\}$
            Add($e\_id.\text{TAG}, S$) to $M_d$
        endif
    endfor
```

Fig. 6. The Marking Algorithm.

adopted a solution in which the portions of the document to which the same policies apply are encrypted with the same keys.

The generation of a well-formed encryption can be logically decomposed into two main phases: the first phase checks which keys are implied by the access control policies stored into the Policy Base, whereas in the second phase the document is encrypted based on the results of the first phase.

An algorithm performing the first task is reported in Figure 6. The algorithm receives as input a document and generates a *marking* of the document which keeps track of which policies apply to the various portions of the document. A document marking is formally defined as follows:

Definition 4.6 (*Document Marking*). Let $d = (V_d, \bar{v}_d, E_d, \phi_{E_d})$ be an XML document. A marking of d is a set of pairs (el, ID_p), where el is:

(1) an identifier of an element in d, that is, $el = e\_id$, $e\_id = id_v, v \in V_d^e \cup \{\bar{v}_d\}$, or

(2) an expression univocally denoting an attribute in d or in one of its elements, that is, $el = e\_id.a$, where $e\_id = id_v, v \in V_d^e \cup \{\bar{v}_d\}$, and a is the name of one of the attributes of e_id, or

(3) an expression of the form $el = e\_id.\text{TAG}$, $e\_id = id_v, v \in V_d^e \cup \{\bar{v}_d\}$.

and ID_p is a set whose elements are identifiers of policies in $\mathcal{PB}$.[7]

According to Definition 4.6, the marking can be done both at the attribute and at the element level. Each attribute/element is marked with the identifiers of the policies that apply to it. If the marking is at the element level, it means that the same policies apply to all the attributes in the element, otherwise each single attribute of the element is marked with the proper policy identifiers. Symbol TAG is used for a correct encryption of elements containing attributes to which different policies apply. Indeed, suppose to have an element e which contains two attributes a_1 and a_2. Suppose moreover that policies P_1 and P_2 apply to a_1, whereas P_3 applies to a_2. Thus, according to Definition 4.2, the view of element e with respect to P_1 and P_2 is equal to element e from which all the attributes different from a_1 have been removed, whereas the view of e with respect to P_3 is the element obtained from e by removing all the attributes different from a_2. Thus, in a well-formed 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 tag of e. Symbol TAG is then used to notify the system that the marking applies only to the start and end tag of an element and not to all its attributes.

Algorithm 1 builds the marking of the input document by considering each policy that can be applied to such document (the set of these policies, denoted by $\Pi_d(\mathcal{PB})$, is computed by Step (1)). The attributes/elements to which a policy applies depend on the document specification in the policy as well as on the privilege and propagation option. The algorithm makes use of function 'Add()', to build a document marking. The result of the statement 'Add (el, id) to M_d' is the addition of element (el, id) to marking M_d, if there does not exist a pair containing el in M_d. Otherwise, it is the addition of id to the set of policy identifiers in that pair.

Example 4.1. Consider the document in Figure 2 and let $\mathcal{PB} = \{\text{P}_1, \text{P}_2, \text{P}_3, \text{P}_4\}$, where $\text{P}_1, \ldots, \text{P}_4$ are the policies in Example 3.6. The output of Algorithm 1 is shown in Figure 7. In the figure, keyword TAG is used to denote that the marking applies only to the start and end tags of the corresponding element.

[7] We assume that each policy is identified by a unique label assigned by the system at the time of its insertion.

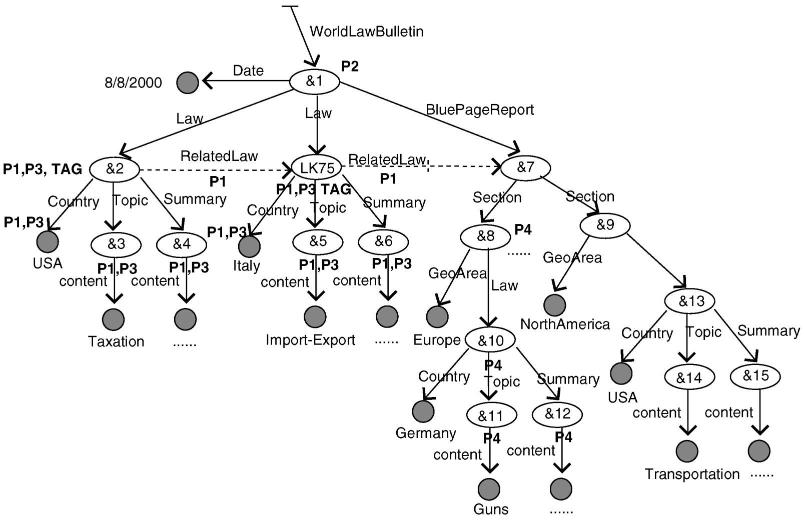

Fig. 7. Marking of the XML document in Figure 3.

Once a marking has been generated, the next step is to generate the keys, based on the marking, and to encrypt the document with these keys. The goal is to generate a well-formed encryption which minimizes the number of keys to be generated. An algorithm performing these operations is reported in Figure 8. The algorithm receives as input a document and its marking generated by Algorithm 1. Algorithm 2 groups elements and attributes according to their marking (i.e., elements and attributes with the same marking are put in the same group). Then, for each distinct group, it generates a different key. Such key is then used to encrypt the members of the group. Then, the algorithm generates an additional key, and encrypts each portion of the document that has not been encrypted in the previous step with such key. The algorithm also builds a table, named *Key_Info*, that contains information on the keys implied by each policy that can be applied to the input document. Each entry of the table has the form (*acp, components, key*), where *acp* is the identifier of an access control policy, *components* is a set of portions of the document to which the policy applies, and *key* is an encryption key generated for this portion. The table contains an additional entry (DEFAULT, C, k) which stores the default key, that is, the key which is used to encrypt those portions of the document which are not covered by any policies.

The algorithm makes use of function 'Add()' to modify *Key_Info* and variable MARK. MARK is a set whose elements are sets of policy identifiers. Thus, the result of the statement 'Add m to MARK' is the inclusion of the set m into set MARK. By contrast, the result of the statement 'Add (acp, C, k) to *Key_Info*' is the addition of tuple (acp, C, k) to *Key_Info*, if there is no entry for acp in the table. Otherwise, it is the addition of C and k to the entry corresponding to acp.

Example 4.2. Suppose we apply Algorithm 2 to the document in Figure 7. MARK = $\{\{P_1, P_3\}, \{P_1\}, \{P_2\}, \{P_4\}\}$. Thus, Step (4) of the algorithm identifies four groups, generates one key for each group, and encrypts the group members with

```
ALGORITHM 2.   The Encryption Algorithm

INPUT:    1. An XML document $d = (V_d, \bar{v_d}, E_d, \phi_{E_d})$ in $\mathcal{S}$
          2. The marking $M_d$ of $d$, returned by Algorithm 1
OUTPUT:   1. $d^e$, the encrypted version of $d$
          2. Table Key_Info. Each tuple in Key_Info has the form $(id\_p, C, key\_set)$,
             where $id\_p$ is the identifier of a policy in $\mathcal{PB}$, $C$ is a set of components of $d$,
             and key_set is a set of keys
METHOD:

(1) MARK = $\emptyset$
(2) For ēach $(el, ID_p) \in M_d$: If $ID_p \notin$ MARK: Add $ID_p$ to MARK
(3) Let $d'$ be a copy of $d$
(4)     For each $m \in$ MARK:
            Let $S_m = \{el \mid (el, ID_p) \in M_d$ and $ID_p = m\}$
            Generate a key $k$
            For each $el \in S_m$:
                If $el$ denotes a node $n$:
                    Encrypt $n$ in $d'$ with $k$
                If $el$ denotes the start and end tags of an element $e$:
                    Encrypt the tags of $e$ in $d'$ with $k$
            endfor
            For each acp $\in m$: Add (acp,$S_m$,$k$) to Key_Info
    endfor
(5) Generate a key $k$
(6) Let $G$ be the set of components in $d'$ which are not encrypted
(7) Encrypt each element in $G$ with $k$
(8) Add (DEFAULT,$G$, $k$) to Key_Info
```

Fig. 8. The Encryption Algorithm.

Table IV. Groups Generated by Algorithm 2

Group members	Key
&1	k_1
&2.Country,&2.TAG,&3,&4,LK75.Country, LK75.TAG,&5, &6	k_2
&2.RelatedLaws, LK75.RelatedLaws	k_3
&8,&10,&11,&12	k_4
&7,&9,&13,&14, &15	k_5

this key. These groups and the corresponding keys are listed in the first four rows of Table IV. Then, in Step (5), a key is generated for all the portions of the document which have not yet been encrypted (this group is in the last line of Table IV). The resulting table *Key_Info* is shown in Figure 9.

Note that Algorithm 2 is highly flexible in terms of the encryption algorithm (i.e., symmetric vs. asymmetric, etc.) actually used to encrypt the various portions of the XML documents and on the techniques used to encrypt such portions. As such, techniques recently proposed by the W3C for the encryption of XML documents [W3C 2000] can be used in the algorithm. However, as new developments in XML encryption will be proposed they can be easily incorporated into our framework as well.

Policy	Components	Keys
P_1	{&2.Country,&2.TAG,&3,&4,LK75.Country, LK75.TAG,&5, &6, &2.RelatedLaws, LK75.RelatedLaws}	$\{k_2,k_3\}$
P_2	{&1}	$\{k_1\}$
P_3	{&2.Country,&2.TAG,&3,&4,LK75.Country, LK75.TAG,&5, &6}	$\{k_2\}$
P_4	{&8,&10,&11,&12}	$\{k_4\}$
DEFAULT	&7,&9,&13,&14, &15	$\{k_5\}$

Fig. 9. Table *Key_Info* for the document in Figure 7.

We are now ready to prove the correctness of Algorithms 1 and 2.

THEOREM 4.1. *Let d be an XML document and let d^e be the encryption of d generated by Algorithms 1 and 2. $\forall$acp $\in \mathcal{PB}$, let Keys(acp) be the set of keys associated with* acp *in the table* Key_Info *of document d . The document obtained by applying the keys in Keys(acp) to d^e is equal to V_d(acp).*

The encryption generated by Algorithms 1 and 2 is a well-formed encryption and satisfies a minimality property, that is, it is the well-formed encryption using the minimum number of keys.

THEOREM 4.2. *The encryption of d generated by Algorithms 1 and 2 is well-formed.*

THEOREM 4.3. *Let d be an XML document, and let d^e be the encryption of d generated by Algorithms 1 and 2. No well-formed encryption $d^{e'}$ of d exists, $d^{e'} \neq d^e$, such that the number of keys used to obtain $d^{e'}$ is less than the number of keys used to obtain d^e.*

Note that, Algorithm 1 is not activated each time a document is broadcasted to a set of subjects, but it is activated only once, when the document is acquired by the source. This makes the release of XML documents under the push mode very efficient, since the documents are encrypted before the release, and the decision of which keys each subject needs requires only a query on table *Key_Info*. Clearly, when a policy is added/removed to/from the Policy Base, the *Key_Info* tables of documents to which the policy applies must be updated accordingly, and the encryption of the document may change. However, due to the properties of well-formed encryption (cfr. Section 4.2) efficient algorithms [Carminati and Ferrari 2002] can be devised that change only those portions of the encryption that are really affected by the modification to the Policy Base, thus minimizing the computational overhead. Moreover, in real situations, administrative operations changing the Policy Base are considerably less frequent than document distribution. Note that operations changing some keys of a document[8] can also be easily supported since they do not require to rebuild the document marking. From table *Key_Info*, one can easily determine which portions of the document are encrypted by such keys and thus reencrypt them with the new keys.

[8] One needs to change a key if, for example, the key has been broken [Summers 1997].

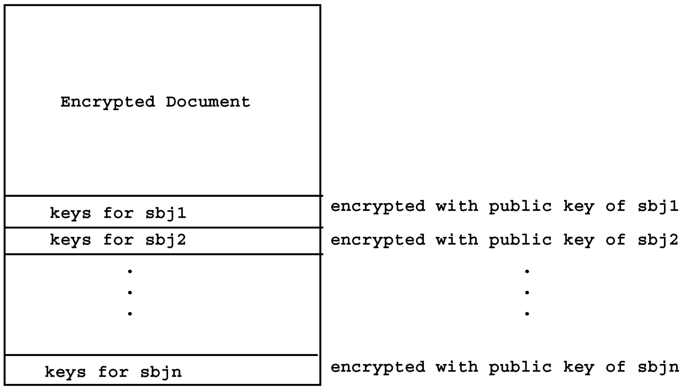
Encrypted Document
keys for sbj1
encrypted with public key of sbj1
keys for sbj2
encrypted with public key of sbj2
.
.
.
keys for sbjn
encrypted with public key of sbjn

Fig. 10. Package structure.

4.4 Document and Key Distribution

Once a well-formed encryption of a document has been generated, the next step is its delivery to a set of subjects along which the keys necessary for its decryption. In our system, two different methods for key distribution are supported: under the first mode (called *online mode*) the keys are delivered to subjects together with the encrypted document, whereas under the second mode (called *offline mode*) the XML source sends only the encrypted document to subjects, whereas keys are retrieved by the subject by further interactions with the XML source. It is important to note that the choice of the key distribution method depends on many factors such as the number of subjects to which a document must be released, the explicit preferences stated by subjects (in the case they have explicitly requested a particular method), the number of keys generated during document encryption, the sensitivity of the information the document contains, and the subject behavior (i.e., whether they are always connected to the network or they are mobile users seldomly connected to the net). Since so many factors influence key distribution techniques, we have designed our system to provide support for a spectrum of key distribution methods thus making the SA able to select the most appropriate one for each document to be distributed under information push. In the following, we describe the key distribution methods supported by our system.

4.4.1 *Online Key Distribution.* In the online approach to key distribution, the XML source sends subjects the encrypted document as well as the corresponding keys. The system supports two different modes for online distribution. Under the first, the encrypted document is encapsulated into a *package* (see Figure 10) that consists of multiple parts. The package contains the encrypted document and the keys each subject needs for a correct decryption. The keys are encrypted with the public key of the subject (or with a shared symmetric key). Once a subject receives the package, he/she decrypts the document keys using his/her private key (or a shared symmetric key). Then, the subject uses these keys to decrypt the document content.

```
ALGORITHM 3. The Document Package Generation Algorithm
INPUT:   1. The encryption $d^e$ of a document $d$ and table Key_Info, returned by Algorithm 2
         2. $\Pi_d(\mathcal{PB})$, returned by Algorithm 1, and the Credential Base $\mathcal{CB}$
         3. A set of subjects $\{s_1, \ldots, s_n\}$ to which $d$ should be sent
OUTPUT:  A package $P_d$ containing document $d$ and the keys for subjects $\{s_1, \ldots, s_n\}$
METHOD:

(1) Insert $d^e$ into $P_d$
(2)   For each $s \in \{s_1, \ldots, s_n\}$:
          Let $C_s \in \mathcal{CB}$ be the set of credentials of subject $s$
          Let $P = \{$acp | acp $\in \Pi_d(\mathcal{PB})$ and $C_s$ satisfies subject-spec(acp)$\}$
          $Keys = \emptyset$
          For each acp $\in P$:
              Retrieve the tuple corresponding to acp in Key_Info
              Let $k_1, \ldots, k_m$ be the keys in the tuple
              $Keys = Keys \cup \{k_1, \ldots, k_m\}$
          endfor
          Encrypt the keys in Keys with the public key of $s$ (or with a shared symmetric key)
          Add the encrypted keys to $P_d$
    endfor
(3) Send $P_d$ to $s_1, \ldots, s_n$
```

Fig. 11. An algorithm for document package generation.

An algorithm for the generation of document packages is shown in Figure 11. The algorithm receives as input the encryption of a document d and its table *Key_Info*, returned by Algorithm 2, and a set of subjects to which the document has to be sent. Additionally, it receives as input $\Pi_d(\mathcal{PB})$, computed by Algorithm 1, that is, the set of policies that apply to d, and the Credential Base $\mathcal{CB}$. For each subject, the algorithm determines the keys the subject needs to access the correct view of the document. Then, the algorithm generates the package according to the methods previously sketched and sends it to all the subjects received in input. The algorithm retrieves information on the keys by querying table *Key_Info*. Thus, by Theorem 4.1, we are assured that each subject accesses the correct view of the document. Note moreover that this approach has the advantage of sending the same package to all the subjects to which the document should be released. Moreover, content protection is ensured since each encrypted part can only be decrypted by an owner of the corresponding private key. The drawback of this approach is that the dimension of the package could be considerable when the number of subjects to which the document should be released is high and when each subject has several keys associated with him/her. Since the keys of all subjects are contained in the same package, this approach can be more vulnerable to attacks by malicious subjects, such as for instance denials of service attacks where a malicious subject deletes from the package the keys corresponding to another subject thus making the subject enable to decrypt the document content. For all these reasons, the system also supports an alternative mode for online distribution under which the keys are not encapsulated into the package but are sent to each subject in a separate

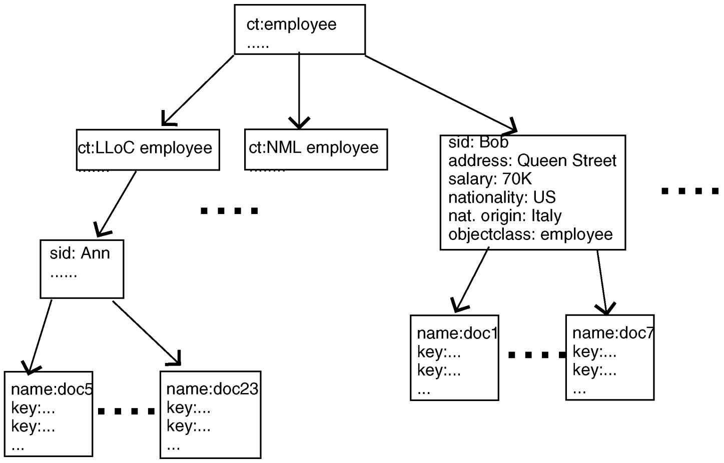

Fig. 12. LDAP directory structure.

message using secure mail techniques such as Pretty Good Privacy (PGP) and S/MIME [Stallings 2000].

4.4.2 *Offline Key Distribution.* In the offline techniques for key distribution the system does not directly send keys to subjects; rather it sends only the (same) encrypted copy of the document to all the subjects, whereas keys are retrieved by subjects through further interactions with the server. To this purpose, the system employs the Lightweight Directory Access Protocol (LDAP) [Srivastava 2000] to store decryption keys. Subjects obtain the appropriate keys by querying the LDAP directory. In a nutshell, an LDAP directory consists of nodes (called *directory entries*) organized in a forest. Each entry contains a set of (*attribute, value*) pairs denoting properties of the entity and has a type (referred to as *objectclass*) associated with it. Retrieval of information in an LDAP directory can be done through the LDAP query language. Prevalent features of the query language are the possibility of defining a scope for the query (e.g., a portion of the LDAP directory tree) and of specifying *Boolean filters*. Boolean filters are obtained by combining conditions on attribute values using the AND, OR, and NOT operators. A filter answer consists of all the entries whose (attribute, value) pairs satisfy the Boolean filter.

Since access control policies are specified in terms of subject credentials the LDAP directory is structured based on the credential type hierarchy (Figure 12 reports a portion of the LDAP directory for the credential type hierarchy of Figure 4). More precisely, the directory contains three different types of entries:

—it contains an entry for each credential type in the credential type hierarchy

—it contains an entry for each subject for which the offline mode for key distribution has to be used

—it contains an entry for each document for which the offline mode for key distribution has to be used.

Children of an entry corresponding to a credential type are all the entities corresponding to its subcredential types in the credential type hierarchy and all the subjects to which the credential type is associated. Entries corresponding to subjects contain a pair (attribute, value) for each attribute of the credential type to which the subject belongs to (see, e.g., the entry corresponding to Bob in Figure 12). Children of an entry corresponding to a subject denote the documents that have to be released to that subject using the offline mode for key distribution. Each of these entries contains the corresponding keys for that subject encrypted with her/his public key.

The correspondence between the credential type hierarchy and the LDAP directory structure has been designed to speed up updates to the LDAP directory when the keys for a new encrypted document must be placed in the directory. Indeed, we need only to determine the set of access control policies which apply to the document (this can be simply done by querying table *Key_Info*) and then take the credential expression contained in each of this policy. The credential expression can be easily converted into an LDAP query that retrieves all the entities that satisfy the credential expression. These entities can correspond either to a credential type or to a subject. For example, with reference to the directory in Figure 12, the query *LLoC Employee* (referring to a policy which applies to all LLoC employees) returns as answer the entity corresponding to the credential type LLoC Employee, whereas the query *Employee AND nationality = US* returns as answer the entity corresponding to subject Bob. When the query returns as answer an entity corresponding to a subject we simply need to update the subtree rooted at that entity by inserting a subentity corresponding to the considered document (if this entity does not already exist) containing an attribute for each key associated with the policy in table *Key_Info*. If this entry already exists we simply need to update it by inserting the keys. Keys are encrypted with the public key of the subject. By contrast, if the entity returned as answer to the query denotes a credential type the procedure we have just presented is performed for each child of this entity denoting a subject.

Figure 13 reports an algorithm to insert keys for a new document into the LDAP directory.

Example 4.3. Consider the LDAP directory in Figure 12 and the document in Figure 7 whose associated *Key_Info* table is reported in Figure 9. Suppose that the LDAP directory must be updated with the keys corresponding to this document. Algorithm 4 considers in turn each policy contained in table *Key_Info*, namely $\mathtt{P}_1, \ldots, \mathtt{P}_4$. The algorithm first considers policy $\mathtt{P}_1$. The credential expression in this policy is LLoC Employee ∨ European Division Employee (cfr. Example 3.6). Thus, the corresponding query on the portion of the LDAP directory in Figure 12 returns entity LLoC Employee. Thus, the algorithm considers all the children of this entity which denote a subject (in the example the only entity satisfying this condition is the one denoting user Ann) and updates the subtree rooted at this entity by inserting an entity corresponding to the document of Figure 7 containing as attributes keys k_2 and k_3 encrypted with the public key of Ann. Then, the algorithm considers policy $\mathtt{P}_2$ and repeats the same steps we have illustrated for policy $\mathtt{P}_1$. The result is that the entity inserted at

```
ALGORITHM 4. The LDAP Directory Update Algorithm

INPUT:    1. A document $d$ and its associated table $Key\_Info$ returned by Algorithm 2
          2. The LDAP Directory
OUTPUT:   The LDAP Directory updated with the keys for document $d$
METHOD:

(1) Let $P$ be the set of policies in $Key\_Info$
(2)    For each $p \in P$:
          Let $K_p$ be the set of keys associated with $p$ in $Key\_Info$
          Let $ce_p$ be the credential expression in $p$
          Let $E$ be the set of entities in the LDAP directory that satisfies the query
          corresponding to $ce_p$
          For each $e \in E$:
             If $e$ denotes a subject $s$:
                If $\exists$ a children $c$ of $e$ corresponding to $d$:
                   encrypt keys in $K_p$ with the public key of $s$
                   insert the encrypted keys in $c$
                else:
                     create a children $c$ of $e$
                     encrypt keys in $K_p$ with the public key of $s$
                     insert the encrypted keys in $c$
                endif
             endif
             If $e$ denotes a credential type
                Let $C$ be the set of children of $e$ representing subjects
                For each $c \in C$
                   Let $s$ be the subject represented by $c$
                   If $\exists$ a children $c'$ of $c$ corresponding to $d$:
                      encrypt keys in $K_p$ with the public key of $s$
                      insert the encrypted keys in $c'$
                   else:
                      create a children $c'$ of $c$
                      encrypt keys in $K_p$ with the public key of $s$
                      insert the encrypted keys in $c'$
    endfor
```

Fig. 13. An algorithm for LDAP directory update.

the preceding iteration is updated by inserting key k_1 encrypted with the public key of Ann. Policies P_3 and P_4 do not imply any update to the portion of LDAP directory in Figure 12. Thus, the resulting LDAP directory is the one shown in Figure 14. In the figure, KUA denotes the public key of Ann.

Possible redundancy of information that arises when a subject belongs to more than one credential type can be avoided by using embedded references [Srivastava 2000] in the LDAP directory.

4.5 System Architecture

In this section, we present the architecture of our system. We describe only the portion of the architecture supporting information push. Details on information pull support can be found in Bertino et al. [2001c].

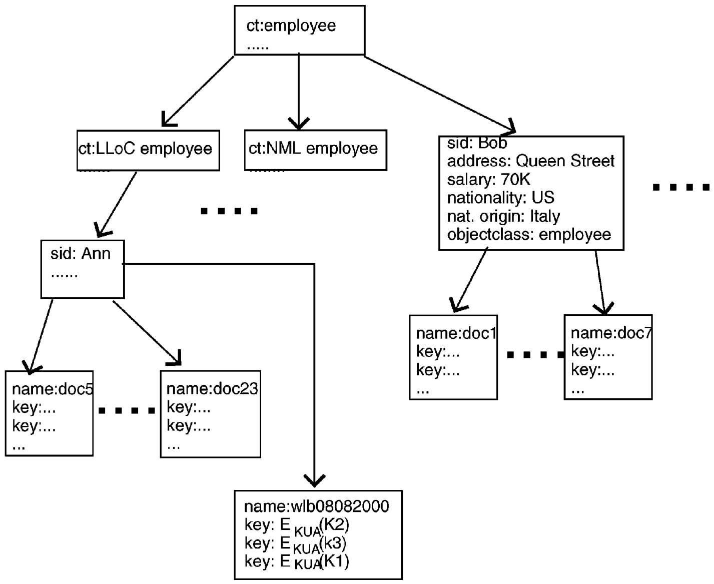

Fig. 14. LDAP directory for Example 4.3.

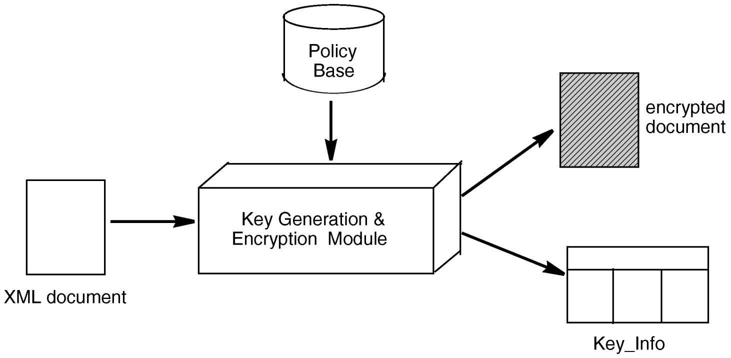

Fig. 15. Key generation and encryption module.

The architecture consists of two main modules: the *Key Generation and Encryption Module* and the *Document and Key Distribution Module*. The *Key Generation and Encryption Module*, illustrated in Figure 15, receives as input an XML document and decides which portions of the document should be encrypted with different keys on the basis of the access control policies in the Policy Base. This task is performed by Algorithm 1 illustrated before. The module then encrypts (by activating Algorithm 2) the document and returns table *Key_Info*, storing information on the generated keys. The *Document Distribution Module* is in charge of broadcasting the encrypted document to a given set of subjects and of managing keys. The operations that this module performs depend on whether the online or the offline strategy for key distribution has been selected by the SA for the considered document.

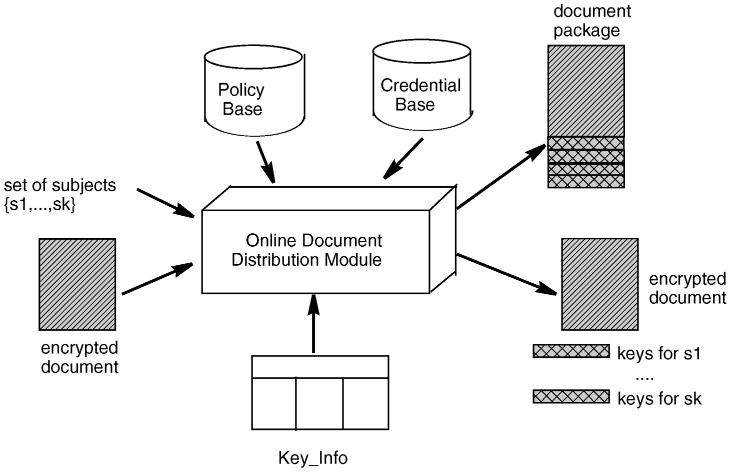

Fig. 16. Document and Key Distribution Module: Online Mode.

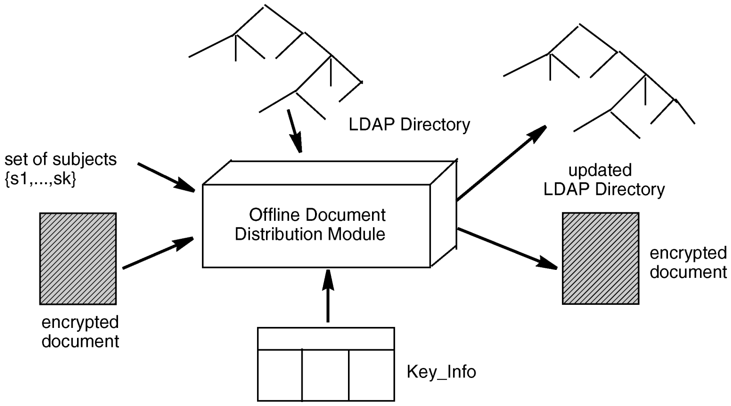

Fig. 17. Document and Key Distribution Module: Offline Mode.

Operations performed for the online mode are illustrated in Figure 16. In this case, the module receives as input the encrypted document, generated by the Key Generation and Encryption Module, and a set of subjects to which the document should be released. On the basis of the access control policies in the Policy Base, on the subject credentials, and on the content of table *Key_Info* for the input document, the module determines which keys are needed by the various subjects to access the correct view of the document, and, based on that, it builds the package to be sent to these subjects (by executing Algorithm 3) or sends the keys to each subject in a separate message, according to the strategy that has been selected.

The other mode of operation of the Document and Key Distribution Module (illustrated in Figure 17) is used when the offline mode for key distribution has been selected. In such case, the system sends to all subjects only the encrypted copy of the document, whereas the corresponding keys are stored into the

LDAP directory (this operation is done by Algorithm 4) and further retrieved by the subjects.

5. RELATED WORK

The work presented in this article has some relationships with access control models and mechanisms developed for object-oriented DBMSs [Fernandez et al. 1994; Rabitti et al. 1991], HTML documents [Samarati et al. 1996], and XML documents [Damiani et al. 2000].

The models proposed in Fernandez et al. [1994] and Rabitti et al. [1991] 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. Those models support concepts such as positive and negative authorizations, and authorization propagation. Our model also supports such concepts, even though it has a larger variety of authorization propagation options. Three different options are supported by which the SA can specify (i) that an authorization defined at a given level in the hierarchy propagates to all lower levels, (ii) that the propagation stops at a specified level down in the hierarchy, or (iii) that no propagation has to be enforced. By contrast, ORION [Rabitti et al. 1991] has only one propagation policy, which is equivalent to option (i). Moreover, none of the above-mentioned models provide support for secure information push mechanisms. This is the most innovative feature of our access control model, which is not found in any access control model previously proposed for object-oriented DBMSs. An access control model for WWW documents has been proposed in Samarati et al. [1996]. In this model, HTML documents are considered, organized as unstructured pages connected by links. Authorizations can be given either to the whole document or to selected portions within the document. Although we borrow from Samarati et al. [1996], 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 to an XML document, describing its structure. Such aspects require the definition and enforcement of more sophisticated access control policies, than the ones devised for HTML documents. The access control model proposed in Samarati et al. [1996] 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, administering authorizations is very difficult. In particular, if one wants to give access to portions of a document, he/she has to manually split the page into different *slots* on which different authorizations are given. This problem is completely overcome in 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 Damiani et al. [2000]. This 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 one is that it does not consider the problem of a secure massive distribution of XML documents and thus considers only the information pull mode. Second, the model proposed in Damiani et al. [2000] 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 user to read the information in an element and/or to navigate through its links, or to modify/delete the content of an element/attribute.

Because of the widespread use of XML and due the relevance of XML security, the World Wide Web Consortium (W3C) has set up several working groups to address the various security aspects related to XML. For example, The XML Working Groups of the W3C are working on standards for both an XML representation of digital signatures (W3C XML Signature Working Group) and encrypted contents (W3C XML Encryption Working Group). The goal of the OASIS Consortium [OASIS Consortium] is the design and development of industry standard specifications for XML-based interoperability. In this framework, the XACML Technical Committee is studying the definition of a standard model for XML-based security policies. However, the draft proposal for XACML is based on very simple access control policies, in that notions such as credentials, positive/negative policies, conflict management, and dissemination strategies are not taken into account.

Other related work deals with approaches proposing more flexible methods to qualify subjects with respect to traditional identity-based schemes for access control. One of the most relevant research efforts in this area are role-based access control (RBAC) models. In particular, note that the concept of credential has some similarity with that of role [Osborn 2000; Sandhu et al. 1996]. Roles can be seen as a set of actions or responsibilities associated with a particular working activity. Under role-based models, all authorizations needed to perform a certain activity are granted to the role associated with that activity, rather than being granted directly to users. Users are then made members of roles, thereby acquiring the roles' authorizations. User access to data is mediated by roles; each user is authorized to play certain roles and, on the basis of the role, he/she can perform accesses on the data. Whenever a user needs to perform a certain activity, the user only needs to be granted the authorization of playing the proper role, rather than being directly assigned the required authorizations. A basic distinction between roles and credentials is that credentials are characterized by a set of attributes, and this allows us to grant access authorizations only to users whose credentials satisfy certain conditions (e.g., access to a document can be granted to all the users with a given age or with a given nationality). This can of course be done also through roles but it requires the creation of a distinct role for each condition we would like to enforce (e.g., enforcing the access control policy of the previous example requires the creation of two distinct roles, one corresponding to the users with the specified age, and the other corresponding to the users with the specified nationality). This makes the specification and management of authorizations very difficult, given also the large variety of users that typically access XML documents.

The concept of subject credential was first presented by Winslett et al. [1997], whereas the access control model proposed in Adam et al. [2000] provides a formalization of the concept of subject credential by proposing the credential logic-based specification language that we use in this article. Other related work is the work on credential specification for stranger parties. In particular, the work by IBM on Trust Policy Language (TPL) [Herzberg and Mass 2001] is devoted to the enforcement of an XML-based framework for specifying and managing role-based access control in a distributed context. This framework has been extended for mapping subject certificates to a role, based on policies defined by the owner of the resource and on the roles of the issuers of the certificates [Herzberg et al. 2000].

6. CONCLUSIONS

In this article, we have presented an access control system supporting selective distribution of XML documents among possibly large user communities. Our system supports the formulation of high-level access control policies. Such policies take into account both user characteristics, and document contents and structure. Some of the main results of our work include an efficient approach for secure document distribution. A prevalent feature of such approach is that it requires a minimum number of keys in order to encrypt the various document components. Additionally, the system provides a range of key distribution methods thus allowing one to apply different key distribution strategies to different documents. This variety of methods is provided as part of our system because the choice of the optimal key distribution mode varies depending on several heterogeneous factors. For instance, the online mode is more appropriate for minimizing the number of message exchanges between subjects and the XML source, whereas the offline mode can be preferable in the case of documents of large size and to which many different access control policies apply (because in such case the dimension of the document package can be quite large). Other factors influencing the key distribution mode are explicit preferences stated by users, the degree of confidentiality to be guaranteed for the transmitted information, and the frequency of document and access control policy updates. An interesting research direction we are currently investigating is thus an evaluation of the performance, both at the theoretical and at the experimental level, in order to determine which key distribution strategy is better and under which circumstances.

Another extension we are currently working on is the development of a user friendly interface for access control administration. Such an interface already exists for credential management [Bertino and Ferrari 2000]. We are also investigating issues related to the mediation of security policies when integrating heterogeneous document sources, as well as content-based authorizations for XML documents containing multimedia data. Additionally, we are extending our system to support also document integrity [Bertino et al. 2002]. Another interesting research direction is related to access control policies for distributed document modifications. An important issue in this context is related to the fact that sometimes a document must follow a specific path during its update.

For instance, the document must be first modified by a secretary then signed by a manager, and so on. Techniques are therefore needed to specify such update paths and to verify that they are followed by XML documents. To support this facility, we plan to investigate the use of roles and document signatures.

APPENDIX

A. NOTATIONS USED IN THE ARTICLE

Table V lists, for the convenience of the reader, the symbols more frequently used in this article. For each symbol, the table reports a brief explanation of its meaning.

B. PROOFS

PROOF OF PROPOSITION 4.1. Let d^e be a well-formed encryption of a document d. By Definition 4.4, proving the thesis is equivalent to prove:

(1) each portion of d^e is encrypted with one and only one key;

(2) $\forall s \in S$, $\exists K' \subseteq Keys$ such that the document obtained by decrypting d^e with the keys in K' is equal to $V_d(s)$.

By hypothesis the encryption is well formed. Thus, by Definition 4.5, each portion of d^e is encrypted with one and only one key. Thus, we are left to prove point 2. Let us consider a subject s and let $\Pi_s(\mathcal{PB}) = \{\texttt{acp} \mid \texttt{acp} \in \mathcal{PB}$ and $s \in \psi(\texttt{subject-spec(acp)})\}$, that is, $\Pi_s(\mathcal{PB})$ is the set of policies which apply to subject s. By hypothesis, for each $\texttt{acp} \in \Pi_s(\mathcal{PB})$, $\exists K' \subseteq Keys$ such that the document obtained by decrypting d^e with the keys in K' is equal to $V_d(\texttt{acp})$. By Definition 4.3, $V_d(s) = \bigcup V_d(\texttt{acp}_i)$, $\forall\ \texttt{acp}_i$ such that $s \in \psi(\texttt{subject-spec(acp}_i\texttt{))}$. Thus, by applying to d^e the union of the sets of keys corresponding to policies in $\Pi_s(\mathcal{PB})$ we obtain $V_s(d)$, which proves the thesis. □

PROOF OF THEOREM 4.1. Let d^e be the encryption generated by Algorithms 1 and 2 for a document d. Let $Keys(\texttt{acp})$ be the keys associated with the access control policy acp in the table *Key_Info* of document d, $\forall \texttt{acp} \in \mathcal{PB}$. We have to prove that $\forall \texttt{acp} \in \mathcal{PB}$, the document obtained by applying to d^e the keys in $Keys(\texttt{acp})$ is equal to $V_d(\texttt{acp})$.

Let us first suppose that acp does not apply to document d, that is, that $\nexists(\texttt{s,o,p}) \in \mathcal{E}(\texttt{acp})$ such that the identifier of d appears in o. In such case, by Definition 4.2, $V_d(\texttt{acp})$ is empty. Let us consider Algorithm 1. If $\nexists(\texttt{s,o,p}) \in \mathcal{E}(\texttt{acp})$ such that the identifier of d appears in o, then acp is not included into set $\Pi_d(\mathcal{PB})$ by Step (1) of the algorithm. Thus, at the end of the execution of Algorithm 1, $\nexists(el, ID_p) \in M_d$, such that $\texttt{acp} \in ID_p$. Algorithm 2 adds an entry $(\texttt{acp}', K')$ to *Key_Info*, $\texttt{acp}' \in \mathcal{PB}$, only if $\exists(el, ID_p) \in M_d$ such that $\texttt{acp}' \in ID_p$. Thus, at the end of the execution of Algorithm 2, table *Key_Info* does not contain any entry for acp, which proves the thesis.

Suppose now that acp applies to document d, that is, $\exists(\texttt{s,o,p}) \in \mathcal{E}(\texttt{acp})$ such that the identifier of d appears in o. In such case, set $\Pi_d(\mathcal{PB})$ computed by Step (1) of Algorithm 1 includes acp. Thus, acp is considered during the

Table V. Notations and Terminology.

Symbol	Meaning
$\mathcal{I}_{\mathcal{E}}, \mathcal{I}_{\mathcal{T}_{\mathcal{E}}}$ *Label*, *Value*	a set of element/DTD element identifiers a set of element tags and attribute names /element values
*Label**	strings obtained by the concatenation of names in *Label* and a symbol in {*|+| ?}
Type	a set of types to be associated with names in *Label**
$\mathcal{S}$	an XML source
$Inst(dtd\_id)$	the identifiers of instances of DTD dtd_id
$\mathcal{AN}, \mathcal{T}, \mathcal{V}$	a set of attribute names/types/values
$\mathcal{CT}, \mathcal{CI}$	a set of credential type/credential identifiers
$\prec_{\mathcal{CT}}$	the credential type hierarchy
$A(ct)$	the set of attributes of instances of the credential type *ct*
$\mathcal{CE}, \mathcal{PE}$	the set of credential/path expressions
$\mathcal{C}, \mathcal{P}, \mathcal{S}$	the set of conditions/ privileges/subjects
$\mathcal{PO}, \mathcal{DS}$	the set of protection object/document specifications
$\mathcal{CB}, \mathcal{PB}$	the credential/policy base
$v(a)$	the value of attribute a
$Eval(pe)$	the set of element identifiers denoted by the path expression *pe*
`subject-spec(acp)`, `document-spec (acp) priv(acp)`, `prop-opt(acp)`	the subject/document specification, privilege, and propagation option of the access control policy `acp`
`s(A)`, `o(A)`, `p(A)`	the subject, object, and privilege of authorization `A`
$\mathcal{ID}(d)$	it returns d, if d is a DTD root identifier; it returns the identifiers of DTD roots in $\mathcal{S}$, if d = `#`.
$\psi(ce)$	the subjects denoted by the credential expression *ce*
$\tau(po)$	the set of documents, elements, and/or attributes denoted by the protection object specification *po*
$Succ^n(e\_id)$	if $n = *$, it returns the identifiers of all the subelements of e_id, if $n = 0$, it returns the empty set, if $n \in \mathbb{N}$, it returns the identifiers of the subelements of element e_id which are at most n level down in the hierarchy with respect to e_id;
$\delta(ds)$	the set et of documents, elements, and/or attributes denoted by the document specification ds
$\mathcal{E}$(`acp`)	the set of authorizations entailed by the access control policy `acp`
V_d(`A`)	view of document d with respect to authorization `A`
V_d(`acp`)	view of document d with respect to access control policy `acp`
$V_d(s)$	view of document d with respect to subject s

execution of Step (3) of the algorithm. Several cases can arise based on the form of `document-spec(acp)`. Let us consider all these cases in turn:

(1) `document-spec(acp)` denotes a set of attributes of document d. Let A be the set of such attributes. In this case, by Definition 4.2, V_d(`acp`) is equal to the set of elements that contain at least an attribute in A, from which all the attributes that do not appear in A have been removed. Since, `document-spec(acp)` denotes a set of attributes, Step (3)(c) of Algorithm 1 is executed. At the end of this step, for all and only the attributes $a \in A$,

$\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, $e\_id \in \mathcal{I}_{\mathcal{E}}$, `acp` $\in ID_p$, where e_id is the identifier of the element that contains attribute a. Step (4) of the algorithm is executed is executed since the condition in the **If** statement is satisfied. It is easy to verify that, at the end of the execution of Step (4), for all and only the attributes $a \in A$, $\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, `acp` $\in ID_p$, where e_id is the identifier of the element that contains a. Moreover, for each element that contains an attribute in A, $\exists(el'.tag, ID'_p)$ such that el' is the identifier of the element, and `acp` $\in ID'_p$. Let us consider Algorithm 2. Step (4) encrypts each attribute in A and adds the corresponding key to the entry associated with `acp`. Moreover, it encrypts the start and end tags of the elements that contain an attribute in A and adds the corresponding keys to the entry associated with `acp` in *Key_Info*. No other element of the marking contains `acp` and thus no further updates are performed on the entry of *Key_Info* corresponding to `acp`. Thus, by decrypting d^e with the keys associated with `acp` in *Key_Info*, V_d(`acp`) is obtained, which proves the thesis.

(2) `document-spec(acp)` denotes a set of elements in d. Let E be the set of the identifiers of the elements of d denoted by `document-spec(acp)`. In this case, Step (3)(b) of Algorithm 1 is executed. Several cases can arise, depending on the privilege in `acp`:

—`priv(acp)` = `browse_all`. By Definition 4.2, V_d(`acp`) depends on the propagation option in `acp`. If `prop-opt(acp)` = `*`, then the view contains all the elements in E and all the subelements of elements in E. If `prop-opt(acp)` = `0`, then the view contains all the elements in E, whereas if `prop-opt(acp)` = `n`, `n` ≥ 1, then the view contains all the elements in E and all the subelements of elements in E from the direct subelements going down `n` levels in the document hierarchy. In all the above cases, the set of identifiers of the elements belonging to the view, is equal to set $E' \cup E^*$, where E^* and E' are the sets computed at the beginning of Step (3)(b). Thus, at the end of Step (3)(b), for all and only $e\_id \in E' \cup E^*$, $\exists(el, ID_p) \in M_d$ such that $el = e\_id$ and `acp` $\in ID_p$. If Step (4) of Algorithm 1 is not executed elements in $E' \cup E^*$ are entirely encrypted by Step (4) of Algorithm 2, and the corresponding keys are added to the entry corresponding to `acp` in table *Key_Info*. No other element of the marking contains `acp`, thus no further update to the entry corresponding to `acp` in *Key_Info* is performed. Thus, the thesis holds. By contrast if Step (4) of Algorithm 1 is executed, then at the end of Step (4) for all and only $e\_id \in E' \cup E^*$ and for all and only the attributes a in the element whose identifier is e_id, $\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, `acp` $\in ID_p$, and $\exists(el', I D'_p) \in M_d$ such that $el' = e\_id$.TAG, `acp` $\in ID'_p$. Thus, similar to point 1, it can be proved that at the end of the execution of Algorithm 2, the entry corresponding to `acp` in *Key_Info* contains all and only the keys that, when applied to d^e, return V_d(`acp`). Thus, the thesis holds.

—`priv(acp)` = `view`. By Definition 4.2, V_d(`acp`) depends on the propagation option in `acp`. If `prop-opt(acp)` = `*`, then V_d(`acp`) contains all the elements in E and all the subelements of elements in E which do not contain `IDREF(s)` attributes . If an element contains `IDREF(s)` attributes, then such attributes are removed from the view. If `prop-opt(acp) = 0`, then V_d(`acp`) contains all the elements in E which do not contain `IDREF(s)` attributes. If an element in E contains `IDREF(s)` attributes, then such attributes are removed from the view. Finally, if `prop-opt(acp) = n`, `n` $\geq$ 1, then the view contains all the elements in E and all the subelements of elements in E from the direct subelements going down `n` levels in the hierarchy, which do not contain `IDREF(s)` attributes . If an element contains `IDREF(s)` attributes, then such attributes are removed from the view. In all the above cases, the set of identifiers of the elements belonging to the view is equal to the set $E' \cup E^*$ computed at the beginning of Step (3)(b). At the end of Step (3)(b) for all and only the elements whose identifiers are in $E' \cup E^*$ and such that they do not contain `IDREF(s)` attributes, $\exists(el, ID_p) \in M_d$ such that $el = e\_id$, and `acp` $\in ID_p$. By contrast, for all and only the elements whose identifiers are in $E' \cup E^*$ and such that they contain `IDREF(s)` attributes, $\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, a is a non `IDREF(s)` attribute in e_id and `acp` $\in ID_p$, $\forall$ non `IDREF(s)` attribute in the element whose identifier is e_id. Thus, similar to the previous case, it is easy to prove that at the end of Algorithm 2, the entry associated with `acp` in *Key_Info* contains all and only the keys to obtain V_d(`acp`) from d^e.

—`priv(acp)` = `navigate`. We omit the prove because it is similar to the proof for `priv(acp) = view`.

(3) `document-spec(acp)` denotes the whole document d. The proof is analogous to the one for the case in which `document-spec(acp)` denotes a set of elements in d. □

Proof of Theorem 4.2. Let d^e be the encryption generated by Algorithms 1 and 2 for a document d. By Definition 4.5, proving the thesis is equivalent to prove that:

(1) each portion of d^e is encrypted with one and only one key, and

(2) $\forall$`acp` $\in \mathcal{PB}$, $\exists K' \subseteq Keys$ such that the document obtained by decrypting d^e with the keys in K' is equal to V_d(`acp`).

Proving point (2) is straightforward. Indeed, by Theorem 4.1, $\forall$`acp` $\in \mathcal{PB}$, V_d(`acp`) can be obtained by applying to the encryption returned by Algorithm 2 the keys associated with `acp` in table *Key_Info*. Thus, we are left to prove that Algorithm 2 encrypts each portion of d^e with one and only one key.

Let us first prove that there does not exist a portion of d^e which is not encrypted. Step (4) of Algorithm 2 encrypts each element/attribute in d for which there exists a corresponding tuple in M_d. Moreover, Step (6) encrypts each portion of d^e which has not been encrypted during Step (4). Thus, after the execution of Step (7) no un-encrypted portions of d^e exist.

Let us now prove that there does not exist a portion of d^e which is encrypted with more than one key. Portions of d^e which are encrypted by Algorithm 2 are:

(1) a whole element (i.e., an element and all its attributes);
(2) an attribute;
(3) the start and end tag of an element.

Let us consider all the above cases.

(1) Suppose that there exists a whole element e in d which is encrypted with two different keys. This element has been encrypted either by Step (4) or (6) of Algorithm 2. Step (6) of Algorithm 2 encrypts with a unique key only those portions of d^e which have not been encrypted by Step (4). Thus, if e is encrypted by Step (6), it is encrypted with only one key, that contradicts the assumption. It is easy to prove that, if the whole element is encrypted with the same key by Step (4), then $\exists(el, ID_p) \in M_d$, such that el is the identifier of e. Let us consider Algorithm 1. Updates to the marking M_d are only performed by calling function Add(). By the definition of function Add(), there cannot exist two elements $(el, ID_p), (el', ID'_p) \in M_d$ such that $el = el'$. Thus, M_d contains only one element for e. This means that e is considered only once during Step (4) and thus is encrypted with only one key, that contradicts the assumption.

(2) Suppose that there exists an attribute a which is encrypted with two keys. If there does not exist an element in M_d for attribute a, then a is encrypted by Step (6) of Algorithm 2 with only one key, which contradicts the assumption. Let us suppose that $\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, where e_id is the identifier of the element which contains a. Using the same considerations we have used for the previous case it can be proved that there cannot exist $(el', ID'_p) \in M_d$ such that $el' = e\_id.a$. Thus, the only possibility for encrypting a with two different keys, is that Algorithm 2 encrypts the whole element in which a is contained with a key, and then it encrypts a with another key. However, by Step (4) of Algorithm 1, if $\exists(el, ID_p) \in M_d$ such that $el = e\_id.a$, then there exists one and only one element $(el'.\mathrm{TAG}, ID'_p) \in M_d$ such that $el' = e\_id$. This implies that only the start and end tag of the element are encrypted by Step (4) of Algorithm 2, that contradicts the assumptions.

(3) Suppose that there exists the start and/or the end tag of an element which is encrypted with more than one key. In such a case, using a reasoning similar to the one we have used for the first case, it can be proved that a contradiction arises. □

Proof of Theorem 4.3. We suppose that the thesis does not hold and derive a contradiction. Suppose that there exists a well-formed encryption $d^{e'}$ of document d such that the number of keys used to obtain $d^{e'}$ is less that the number of keys used to obtain d^e, where d^e is the encryption of d returned by Algorithm 2. Let $Keys(d^{e'})$ be the set of keys used to obtain $d^{e'}$ and let $Keys(d^e)$ be the set of keys used to obtain d^e. Since, by hypothesis, d^e and $d^{e'}$ are well formed, it means that $\exists$ acp $\in \mathcal{PB}$ such that the number of keys in $Keys(d^{e'})$ that have to be used to obtain $V_d(\texttt{acp})$ is less than the number of keys in $Keys(d^e)$ that have to be used to obtain $V_d(\texttt{acp})$. This means that two portions of $V_d(\texttt{acp})$ are encrypted with the same key in $d^{e'}$ and with different keys in d^e. It is easy to prove that if Algorithm 2 encrypts two portions of $V_d(\texttt{acp})$ with different keys, then $\exists \texttt{acp}' \in \mathcal{PB}$ such that $\texttt{acp}'$ applies to a portion and not to the other. Thus, if the two portions are encrypted with the same key k in $d^{e'}$, this means that there does not exist a subset of $Keys(d^{e'})$ that allows one to obtain exactly $V_d(\texttt{acp}')$. Indeed, if k is contained in this set, then a portion of $d^{e'}$ which is not in $V_d(\texttt{acp}')$ is decrypted. Similarly, if k is not contained in this set, then a portion of $V_d(\texttt{acp}')$ is not decrypted. In both cases, a contradiction arises. □

REFERENCES

Adam, N., Atluri, V., Bertino, E., and Ferrari, E. 2002. A content-based authorization model for digital libraries. *IEEE Trans. Knowl. Data Eng. 14*, 2, 296–315.

Bertino, E., Carminati, B., Ferrari, E., Thuraisingam, B., and Gupta, A. 2002. Selective and authentic third-party distribution of XML documents. MIT Sloan Working Paper No. 4343-02.

Bertino, E., Castano, S., and Ferrari, E. 2001a. Author-X: A comprehensive system for securing XML documents. *IEEE Internet Comput. 5*, 3, 21–31.

Bertino, E., Castano, S., and Ferrari, E. 2001b. On specifying security policies for web documents with an XML-based language. In *Proceedings of the 1st ACM Symposium on Access Control Models and Technologies* (*SACMAT'01*) (Chantilly, Va.). ACM, New York.

Bertino, E., Castano, S., Ferrari, E., and Mesiti, M. 2001c. Specifying and enforcing access control policies for XML document sources. *WWW J. 3*, 3, 139–151.

Bertino, E., and Ferrari, E. 2000. Secure and Selective Dissemination of XML Documents. Technical Report, Department of Computer Science, University of Milano (Extended version of this article.)

Carminati, E. and Ferrari, E. 2002. Access control policy management for XML documents. Tech. Rep. Department of Computer Science, University of Milano, Milano, Italy, submitted for publication.

Damiani, E., de Capitani di Vimercati, S., Paraboschi, S., and Samarati, P. 2000. Securing XML Documents. In *Proceedings of the 6th International Conference on Extending Database Technology* (Konstanz, Germany), pp. 121–135.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. 1999. Securing XML documents. In *Proceedings of the International Conference on World Wide Web*, available at: http://www.research.att.com/suciu.

Fernandez, E., Gudes, E., and Song, H. 1994. A model for evaluation and administration of security in object-oriented databases. *IEEE Trans. Knowl. Data Eng. 6*, 275–292.

Gladney, H. and Lotspiech, J. 1997. Safeguarding digital library contents and users: Assuring convenient security and data quality. *D-lib Mag*.

Herzberg, A. and Mass, Y. 2001. Relying party credentials framework. In *Proceedings of the RSA Conference* (San Francisco, Calif.).

Herzberg, A., Mass, Y., and Mihaeli, J. 2000. Access control meets public key infrastructure, or: assigning roles to strangers. *In Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, Calif.). IEEE Computer Society Press, Los Alamitos, Calif.

Milo, T. and Zohar, S. 1998. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the 24th International Conference on Very Large Data Bases* (*VLDB'98*). pp. 122–133.

OASIS Consortium. http://www.oasis-open.org.

Osborn, S. ed. 2000. *Proceedings of the 5th ACM Workshop on Role-Based Access Control* (Berlin, Germany). ACM, New York.

Rabitti, F., Bertino, E., Kim, W., and Woelk, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Datab. Syst. 16*, 1, 88–131.

SAMARATI, P., BERTINO, E., AND JAJODIA, S. 1996. An authorization model for a distributed hypertext system. *IEEE Trans. Knowl. Data Eng. 8*, 4, 555–562.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *IEEE Comput. 29*, 2, 38–47.

STALLINGS, W. 2000. Network security essentials: Applications and standars. Prentice-Hall, Englewood Cliffs, N.J.

SRIVASTAVA, D. 2000. Directories: Managing Data for Networked Applications. Tutorial presented at the *16th IEEE International Conference on Database Engineering* (*ICDE'00*) (San Diego, Calif.). IEEE, Computer Society Press, Los Alamitos, Calif.

SUMMERS, R. C. 1997. Secure Computing: Threats and Safeguards. McGraw-Hill, New York.

WINSLETT, M., CHING, N., JONES, V., AND SLEPCHIN, I. 1997. Using digital credentials on the world wide web. *J. Comput. Secu. 7*.

WORLD WIDE WEB CONSORTIUM 1998. Extensible Markup Language (XML) 1.0. Available at: http://www.w3.org/TR/REC-xml.

WORLD WIDE WEB CONSORTIUM 2000. XML Encryption Syntax and Processing. Available at: http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/att-0001/01-xmlencoverview.html.

Received September 2000; revised March 2002; accepted March 2002