CERIAS Tech Report 2004-69

SELECTIVE AND AUTHENTIC THIRD-PARTY DISTRIBUTION OF XML DOCUMENTS

by E. Bertino, B.Carminati, E.Ferrari, B. Thuraisingham, A. Gupta

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

Selective and Authentic Third-Party Distribution of XML Documents

Elisa Bertino, *Fellow*, *IEEE*, Barbara Carminati, Elena Ferrari, *Member*, *IEEE*, Bhavani Thuraisingham, *Fellow*, *IEEE*, and Amar Gupta, *Senior Member*, *IEEE*

Abstract—Third-party architectures for data publishing over the Internet today are receiving growing attention, due to their scalability properties and to the ability of efficiently managing large number of subjects and great amount of data. In a third-party architecture, there is a distinction between the *Owner* and the *Publisher* of information. The Owner is the producer of information, whereas Publishers are responsible for managing (a portion of) the Owner information and for answering subject queries. A relevant issue in this architecture is how the Owner can ensure a secure and selective publishing of its data, even if the data are managed by a third-party, which can prune some of the nodes of the original document on the basis of subject queries and access control policies. An approach can be that of requiring the Publisher to be trusted with regard to the considered security properties. However, the serious drawback of this solution is that large Web-based systems cannot be easily verified to be secure and can be easily penetrated. For these reasons, in this paper, we propose an alternative approach, based on the use of digital signature techniques, which does not require the Publisher to be trusted. The security properties we consider are authenticity and completeness of a query response, where completeness is intended with regard to the access control policies stated by the information Owner. In particular, we show that, by embedding in the query response one digital signature generated by the Owner and some hash values, a subject is able to locally verify the authenticity of a query response. Moreover, we present an approach that, for a wide range of queries, allows a subject to verify the completeness of query results.

Index Terms—Secure publishing, third-party publication, XML, authentication, completeness.

1 Introduction

XML (*eXtensible Markup Language*) is rapidly becoming a de facto standard for document representation and exchange over the Web [1]. A common requirement for Web applications is thus the need for secure publishing of XML documents. By secure publishing, we mean that the publishing service must ensure some security properties to the data it manages. Third-party architectures for data publishing over the Web are today receiving growing attention, due to their scalability properties and to the ability of efficiently managing a large number of subjects and a great amount of data. In a third-party architecture, there is a distinction between the *Owner* and the *Publisher* of information. The Owner is the producer of the information, whereas Publishers are responsible for managing (a portion of) the Owner information and for answering subject queries. A relevant issue in this architecture is how the Owner can ensure a secure publishing of its data, even if the data are managed by a third-party. The most intuitive solution is that of requiring Publishers to be trusted with regard to the considered security properties. However, this solution could not always be feasible in the Web environment since large Web-based systems cannot be easily verified to be secure and can be easily penetrated. For this reason, in this paper, we propose an alternative approach that does not require the Publisher to be trusted. In the paper, we mainly focus on two of the most relevant security properties: authenticity and completeness. Ensuring document authenticity means that the subject receiving a document is assured that the contents of the document itself actually originated at the claimed source. We refer to this requirement as *document source authenticity*. A second requirement is to ensure the integrity of the document received by a subject with respect to the original document, thus avoiding, for instance, that the Publisher modifies some document portions before returning them to a subject. We refer to such requirements as *document contents authenticity*. Ensuring the completeness of the response means that any subject must be able to verify that he or she has received all the document(s) (or portion(s) of document(s)) that is entitled to access, according to the stated access control policies.

The key point of our approach is that, even though we do not require the Publisher to be trusted with respect to authenticity and completeness properties, we are able to ensure, at the same time, that a subject is able to verify such properties on the answer returned by a Publisher. This capability is obtained by a combination of digital signature and hashing techniques. Let us first consider authenticity. In the scenario, we consider that it is not enough that the

- *E. Bertino is with the CERIAS and CS Department, Purdue University, Recitation Building, 656 Oval Drive, West Lafayette, IN 47907-2086. E-mail: bertino@cerias.purdue.edu.*
- *B. Carminati and E. Ferrari are with the Dipartimento di Scienze della Cultura, Politiche e dell'Informazione, Università dell'Insubria, Como, Via Valleggio 11, 22100 Como, Italy. E-mail: {barbara.carminati, elena.ferrari}@uninsubria.*
- *B. Thuraisingham is with MITRE-Bedford, 202 Burlington Road, Bedford, MA 01730-1420. E-mail: thura@mitre.org.*
- *A. Gupta is with the Eller College of Management, University of Arizona, McClelland Hall 417H, PO Box 210108, Tucson, AZ 85721-0108. E-mail: gupta@eller.arizona.edu.*

Manuscript received 11 Apr. 2002; revised 11 Feb. 2003; accepted 13 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 116338.

Owner signs each document it sends to the Publisher since the Publisher may return to a subject only selected portions of a document, depending on the query the subject submits and on the access control policies in place. For this reason, we propose an alternative solution that requires that the Owner sends the Publisher, in addition to the documents it is entitled to manage, a summary signature for each managed document, generated using a technique based on Merkle hash trees [13]. The idea is that, when a subject submits a query to a Publisher, the Publisher also sends him/her, besides the query result, also the signatures of the documents on which the query is performed. In this way, the subject can locally recompute the same bottom-up hash value signed by the Owner and, by comparing the two values, he/she can verify whether the Publisher has altered the content of the query answer and can be sure of its authenticity. The problem with this approach is that, since the subject may be returned only selected portions of a document, he/she may not be able to recompute the summary signature, which is based on the whole document. For this reason, the Publisher sends the subject a set of additional hash values, referring to the missing portions, which make the subject able to locally perform the computation of the summary signature.

Query answers returned by the Publisher to a subject are filtered according to the access control policies specified by the information Owner. Such policies are specified by means of an expressive credential-based access control model [4] specifically tailored to the protection of XML documents. Thus, additional information sent by the Owner to the Publisher is what policies apply to what portions of the documents it manages.

Completeness verification relies on the use of the *secure structure* of an XML document, which is sent by the Owner to the Publisher and successively returned to subjects together with the answer to a query on the associated document. This additional document contains the hashed tagname and attribute names and values of the original XML document, so that the receiving subject is prevented from accessing information he/she is not allowed to access, being at the same time able to perform the completeness verification. To verify completeness of the received answer, the idea is that the subject performs the same query submitted to the Publisher on the secure structure. Clearly, the query is first transformed to substitute each tag and attribute name, and each attribute value with the corresponding hash value. In the paper, we show that, by comparing the result of the two queries, the subject is able to verify completeness for a wide range of XPath [1] queries.

Finally, to make the Publisher able to verify which access control policies apply to a subject, we require a subject first subscribes to the Owner. As a result of the subscription process, the Owner returns the subject a *policy configuration*, that is, a certificate containing information about the access control policies that apply to the subject. The subject policy configuration is signed with the private key of the Owner to prevent the subject from altering its content. All the security information exchanged between the parties of our architecture is encoded using XML.

In the paper, besides providing a formal foundation for the proposed infrastructure, we formally prove its soundness. To the best of our knowledge, the work presented in this paper is the first one that provides such a comprehensive infrastructure for the selective, authentic, and complete third-party publication of XML documents.

The remainder of this paper is organized as follows: The next section briefly introduces XML, Merkle hash trees, and the access control model we use throughout the paper. Section 3 presents the overall architecture supporting our approach. Sections 4, 5, and 6 describe the information generated during the interactions among the Owner, the Publisher, and the subjects. Section 7 deals with authenticity and completeness verification. In Section 8, we analyze potential attacks which could be conducted by the Publishers or by the subjects, whereas in Section 9, we discuss the overhead implied by our approach. Section 10 presents related work. Section 11 concludes the paper and outlines future research directions. Appendix I contains an example of security enhanced XML document, whereas Appendix II proposes an example of authenticity verification. Finally, Appendix III shows the formal proofs of the theorems presented in this paper. (Appendices can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm.)

2 Basic Concepts

In this section, we start by first reviewing the basic concepts of XML [1]. We then introduce Merkle hash trees [13]. Finally, we summarize the main characteristics of the access control model used by the Owner to specify access control policies.

2.1 Basic Concepts of XML

Building blocks of XML documents are nested, tagged *elements*. Each tagged element has zero or more subelements, zero or more attributes, and may contain textual information (*data content*). Elements can be nested at any depth in the document structure. Attributes can be of different types, allowing one to specify element identifiers (attributes of type `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`(s)/`URI`(s)).

To develop, on a formal basis, our approach for secure publishing of XML documents, we introduce a formal model of XML documents that we use throughout the paper. In the following, we denote with $Label$ a set of element tags and attribute names, and with $Value$ a set of attribute/element values.

Definition 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* $v \in V_d^a$ *has an associated value* $val \in Value$; *each* $v \in V_d^e$ *may have an associated data content.*
- $\bar{v}_d$ *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 the 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)/*`URI`*(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.*

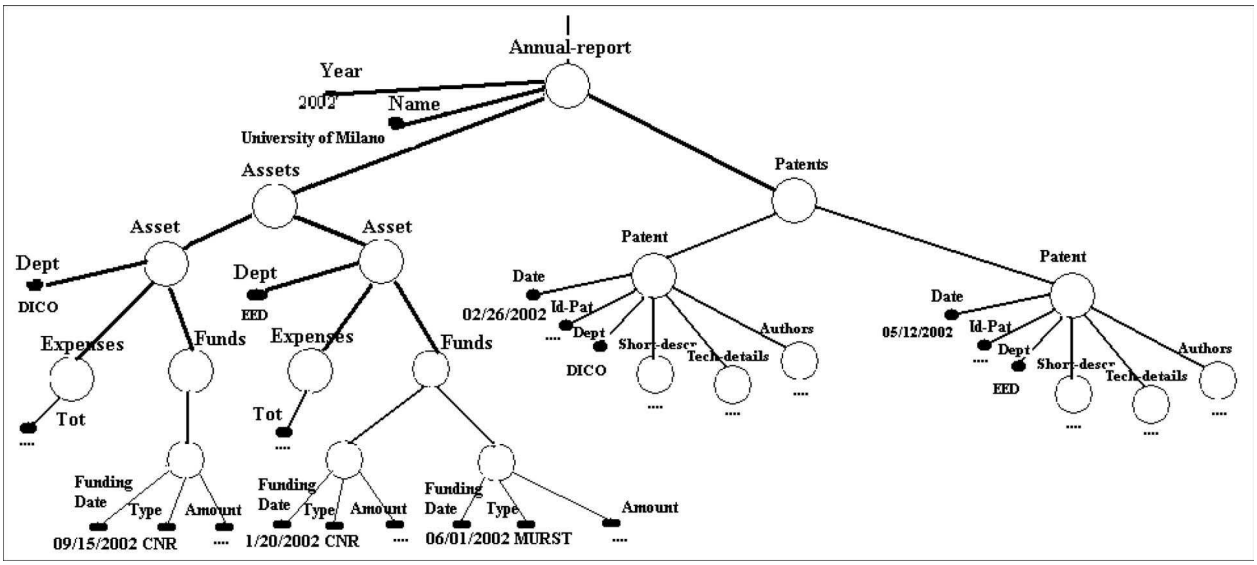

Fig. 1. Graph representation of an XML document.

Based on the above definition, an XML document can be represented as a graph. In the graph representation adopted, white nodes represent elements, whereas black nodes represent attributes. The graph representation contains edges representing the element-attribute and the element-subelement relationships, and *link edges*, representing links between elements. Solid lines represent edges, whereas dashed lines represent link edges. An example of XML document modeling a university annual report is presented in Fig. 1.[1] Further, XML can define application specific document types using document type definitions (DTDs) or XML Schemas.

For the scope of this paper, given the graph representation of an XML document, we assume the existence of an order among the children of an element. The existence of this order is fundamental when a subject has to verify the authenticity of a query result (see Section 7.1). To define this order, we make the assumption that, given an element e, its attributes precede in the order the subelements of e. Moreover, we assume the existence of a function, denoted as $child$, that, given an XML document $d = (V_d, \overline{v}_d, E_d, \phi_{E_d})$, an element $v \in V_d^e$, and an integer $i \in \{1, \ldots, Nc_v\}$, returns the ith child of v with regard to the defined order, where Nc_v denotes the number of children of node v.

2.2 Merkle Hash Trees for XML Documents

Authenticity is ensured by using the Merkle tree authentication mechanism proposed in [13] and adapting it to XML. The method we propose allows a subject to prove source authenticity as well as the authenticity of both the schema and the content of a document. To accomplish this goal, the idea is to associate an hash value with each node in the graph representation of an XML document. More precisely, the hash value associated with an attribute is obtained by applying a hash function over the concatenation of the attribute value and the attribute name. By contrast, the hash value associated with an element is the result of the same hash function computed over the concatenation of the element content, the element tag name, and the hash values associated with its children nodes, both attributes and elements. Hash values associated with the nodes of an XML document are computed by the *Merkle hash function*, defined in what follows.

Definition 2 (Merkle hash function). *Let $d = (V_d, \overline{v}_d, E_d, \phi_{E_d})$ be an XML document. Let $h()$ be a collision-resistant hash function, and HS be the codomain of $h()$. The Merkle hash function associated with d, denoted as MhX_d, is a function: $V_d \rightarrow HS$ such that, for each $v \in V_d$:*[2]

$$MhX_d(v) = \begin{cases} h(h(v.val) \| h(v.name)) & \text{if } v \in V_d^a \\ h(h(v.content) \| h(v.tagname) \| \\ MhX_d(child(1,v)) \| \ldots \| MhX_d(child(Nc_v, v))), & \text{if } v \in V_d^e, \end{cases}$$

where "$\|$" denotes the concatenation operator, and $child()$ is the function defined in Section 2.1.

Throughout the paper, we use the expression *Merkle hash value of v* to denote the value returned by the Merkle hash function associated with a document d when its input is one of its nodes v, that is, $MhX_d(v)$. The key point is that, if a subject knows the correct Merkle hash value of a node, the Publisher cannot forge the value of the descendant children. Thus, to make a subject able to verify the authenticity of a query result on a document d, the subject must possess the Merkle hash value of the root of d. The Publisher returns such Merkle hash value to a subject together with the query result. The Publisher in turn receives this value from the Owner. Since the Merkle hash value of the root of a document is fundamental for the subject verification

1. Note that no dashed lines are contained in the graph in Fig. 1 because the corresponding document does not contain any `IDREF/URI` attribute.

2. Given an element e, we use the notation $e.content$ and $e.tagname$ to denote the data content and the tagname of e, respectively. Note that, according to the XML specification [1], the data content of an XML element can consist of textual data mixed with subelements. In this case, with the term $e.content$, we mean the concatenation of the different portions of textual data contained in e. Finally, given an attribute a, the notation $a.val$ and $a.name$ is used to denote the value and the name of attribute a, respectively.

```
<Professor credID="9" sbjID="16" CIssuer="2" >
    <name> Alice Brown </name>
    <university> University of Milan </university>
    <department> DICO </department>
    <research-group> DB </research-group>
</Professor>
```

```
<secretary credID="12" sbjID="4" issuerID="2">
    <name> Tom Moore </name>
    <university> University of Milan </university>
    <department> DICO </department>
    <level> senior </level>
</secretary>
```

Fig. 2. Examples of $\mathcal{X}$-Sec subject credentials.

process, there is the need of ensuring that a Publisher cannot alter it. For this reason, we impose that the Owner must sign it, and we refer to this signature as the *Merkle Signature* of a document. Usually, two approaches can be used to generate a digital signature [15]. The first implies the computation of the digest of the data being signed, which is then encrypted with a secret key. In this case, to make the receiver able to verify the authenticity of the received data, it is required that the Owner and the receiver share the secret key. Since the management of shared keys in a highly dynamic scenario like the Web is very costly, we adopt the second approach, which is based on asymmetric encryption. Asymmetric encryption relies on the definition of a pair keys, a public and a private key. These keys are defined in such a way that a message encrypted with the public key can be only decrypted using the private key, and vice versa. According to this schema, a subject receiving the digest encrypted with the private key of the Owner is able to verify the authenticity, without the need of costly management of shared secret keys. All he/she has to do is to decrypt it with the Owner public key. Note also that, since the data to be signed are already digested (see Definition 2), we can directly encrypt the Merkle hash value of the root of the document with the Owner's private key. Thus, with the term Merkle Signature of document d, we simply mean the encryption with the Owner's private key of the Merkle hash value of the root of document d. The correctness of a digital signature is based on the requirement that the signing and verification process are performed on exactly the same bits. Thus, even if a single bit of the data being signed is modified, for instance, the order of the attributes is inverted, the digital signature cannot be verified. To avoid this problem, we impose that the Merkle hash function is applied on a canonical form [1] of the corresponding XML document.

2.3 An Access Control Model for XML Documents

Access control policies are specified by the Owner according to the access control model presented in [4], whose main characteristics are summarized in what follows. In this model, subjects are qualified by means of *credentials*. A credential is a set of properties concerning a subject that are relevant for security purposes (for example, age, position within an organization). Credentials are encoded using an XML-based language, called $\mathcal{X}$-Sec [3], as illustrated in Fig. 2. Access control policies specify conditions on the credentials and properties of the credentials, using an XPath-compliant language.

The access control model provides varying access granularity levels, and can express policies that apply to: 1) all the instances of a DTD/XML Schema, 2) collections of documents not necessarily instances of the same DTD/XML Schema, and 3) selected portions within a document(s), or a link (or a set of links). This set of granularity levels is complemented with the possibility of specifying access control policies based on the document content, in addition to the document structure.

Like credentials, access control policies are also encoded using the $\mathcal{X}$-Sec language. We use the term *Policy Base* ($\mathcal{PB}$) to denote the XML file encoding the access control policies of the Owner.

Example 1. Fig. 3 shows a $\mathcal{PB}$ for the XML document in Fig. 1. The first two access control policies authorize DICO professors to access all the patents of their department, whereas they are entitled to see only the authors and the short description of patents of EED. By contrast, the third and fourth policies state that EED professors are entitled to access all the patents of their department, whereas they are entitled to see only the authors and the short description of patents of DICO. The fifth access control policy authorizes DICO junior

```
<?xml version="1.0" encoding="UTF-8" ?>
- <policy_base>
    <policy_spec Id="P1" cred_expr="//Professor[department='DICO']" target="annual_report.xml" path="//Patent
      [@Dept='DICO']//node()" priv="VIEW" />
    <policy_spec Id="P2" cred_expr="//Professor[department='DICO']" target="annual_report.xml" path="//Patent
      [@Dept='EED']/Short-descr/node() and //Patent[@Dept='EED']/authors" priv="VIEW" />
    <policy_spec Id="P3" cred_expr="//Professor[department='EED']" target="annual_report.xml" path="//Patent
      [@Dept='EED']//node()" priv="VIEW" />
    <policy_spec Id="P4" cred_expr="//Professor[department='EED']" target="annual_report.xml" path="//Patent
      [@Dept='DICO']/Short-descr/node() and //Patent[@Dept='DICO']/authors" priv="VIEW" />
    <policy_spec Id="P5" cred_expr="//secretary[department='DICO' and level='junior']"
      target="annual_report.xml" path="//Asset[@Dept='DICO']/node()" priv="VIEW" />
    <policy_spec Id="P6" cred_expr="//secretary[department='DICO' and level='senior']"
      target="annual_report.xml" path="//Asset[@Dept='EED']/Funds/@Type and //Asset
      [@Dept='EED']/Funds/@Funding-Date" priv="VIEW" />
    <policy_spec Id="P7" cred_expr="//secretary[department='EED' and level='junior']" target="annual_report.xml"
      path="//Asset[@Dept='EDD']/node()" priv="VIEW" />
  </policy_base>
```

Fig. 3. An example of Policy Base.

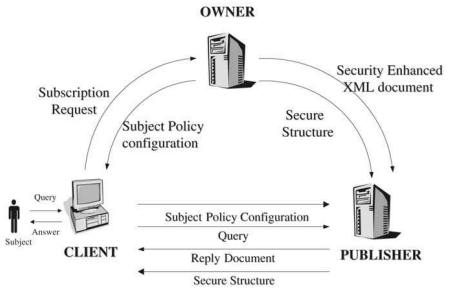

Fig. 4. System architecture.

secretaries to access the department assets. Similarly, the last policy states that EED junior secretaries can access the assets of EED. By contrast, the sixth access control policy makes the DICO senior secretaries able to access assets of EED.

3 Overall Architecture

The architecture we propose for secure publishing of XML documents, shown in Fig. 4, relies on the distinction between the *Owner* of the information and one or more *Publishers*, that are entitled to publish the information (or portions of it) of the Owner and to answer subjects queries.

The Owner specifies access control policies according to the model summarized in Section 2, and sends the Publisher the documents it is entitled to manage. For each document, the Owner sends the Publisher also information on which subjects can access which portions of the document, according to the access control policies it has specified. Additionally, the Owner sends the Publisher the Merkle Signature of each document it is entitled to manage. All this additional information is encoded in XML and attached to the original document, forming the so-called *security enhanced XML document*. The information contained into the security enhanced XML document allows a subject to verify the authenticity of the information returned by the Publisher, but it is not sufficient to make a subject able to verify the completeness of a query result. For this reason, the Owner supplies the Publisher with some additional information about the structure of the original XML document. This information is encoded into an XML document, called *secure structure*, which contains the structure of the XML document, transformed through the use of an hash function. Moreover, the secure structure is complemented with its Merkle Signature, to prevent alterations by the Publisher.

Subjects are required to register with the Owner, during a mandatory *subscription phase*. During this phase, a subject is assigned by the Owner one or more credentials, which are stored at the Owner site.[3] As a result of the subscription process, the Owner returns the subject the *subject policy configuration*, which stores information on the access control policies that apply to the subject. The subject policy configuration is signed with the private key of the Owner to prevent the subject from altering its content.

After the subscription phase has been completed, the subject can submit queries to a Publisher. When a subject submits a query, it also sends the Publisher its policy configuration, to enable the Publisher to determine which access control policies apply to the subject. On the basis of the policy configuration and the submitted query, the Publisher computes a *view* of the requested document(s), which contains all and only those portions of the requested document(s) for which the subject has an authorization according to the access control policies in place at the Owner site. In order to verify the authenticity of the answer, the subject must be able to locally recompute the same bottom-up hash value signed by the Owner (i.e., the Merkle Signature), and to compare it with the Merkle Signature generated by the Owner and inserted by the Publisher in the answer. Since the view computed by the Publisher may not contain all the nodes of the requested documents, the subject may not be able to compute the bottom-up hash value over the whole document by considering only the nodes in the view. Thus, the Publisher complements the view with additional information (e.g., hash values computed over the parts of documents not contained in the view). The Publisher locally computes both the view and the additional information by considering only the security enhanced version(s) of the requested document(s). The result is an XML document, called *reply document*. Upon receiving the reply document, the subject can verify, by using only the information in the reply document itself, the authenticity of the answer. Additionally, the subject can make some verification on the completeness of the query result by using the information contained in the secure structure associated with the document on which the query applies.

In the next sections, we describe in more details the interactions among subjects, Publishers, and the Owner.

4 Subject-Owner Interaction

When a subject subscribes to the Owner, it receives back an object called *subject policy configuration* (cfr. Fig. 4), providing information on the access control policies that the subject satisfies.

Definition 3 (Subject policy configuration). *Let $Policies$ be the set of identifiers of the access control policies[4] specified by the Owner. Let s be a subject subscribing to the Owner, and let $PC(s)$ be the subset of $Policies$ which contains all and only the identifiers of policies that apply to s. The* policy configuration of s *is the signature of $PC(s)$ with the Owner's private key.*

As all the other security information, also the *subject policy configuration* is encoded using XML.

Example 2. According to the subject credentials illustrated in Fig. 2, the access control policies which apply to Alice Brown are P_1 and P_2. By contrast, the credential of Tom Moore satisfies only policy P_7. Thus, the policy configuration of Alice Brown and Tom Moore are, respectively, the signatures of the identifiers 1, 2, and 7 with the Owner's private key.

3. Alternatively, subject credentials can be issued by a trusted third-party Credential Authority, and presented to the Owner during the subscription phase.

4. We assume that each access control policy is uniquely identified by an identifier generated by the system when the policy is specified.

5 Owner-Publisher Interaction

For each document the Publisher is entitled to manage, the Owner sends the corresponding security enhanced document and the corresponding secure structure, which are described in the following sections.

5.1 Security Enhanced XML Document

The first information the security enhanced document contains is which access control policies apply to the corresponding document. Policy information is specified at the element level since different access control policies can apply to different elements and/or attributes of the same document. The idea is to encode information about the set of policies that apply to a specific element into a string of hexadecimal values, called *policy configuration*, and to store this string as an additional attribute of the corresponding element within the security enhanced version of the document (this attribute is called `PC`). In the following, given an XML document d, we denote with $Policies(d) = \{P_i, \ldots, P_p\}$ the identifiers of the access control policies that apply to d. With each element e in d, we associate a binary string of length equal to the cardinality of $Policies(d)$, denoted $ps(e)$, where, starting from the left side, the value of the ith bit is: 1, if the ith policy in $Policies(d)$[5] applies to e; 0, otherwise. Moreover, in order to encode this string into an hexadecimal value, we split $ps(e)$ into m blocks of four bits[6] and translate each block into the corresponding hexadecimal representation. Thus, the *policy configuration* of e, denoted as $PC(e)$, is a string $a_1 \ldots a_m$, such that a_i is the hexadecimal representation of b_i, $i = 1 \ldots m$. To store information about policies that apply to e's attributes, we propose a slightly different approach. The idea is to store into a unique attribute, added to e and called $\mathrm{PC}_{\mathrm{Attr}}$, the concatenation of the policy configurations of all the attributes of e.

Finally, to make policy configurations meaningful to Publishers, it is necessary to insert an additional element into the security enhanced version of a document d. This element, called `Policy`, contains the identifiers of the policies in $Policies(d)$. These identifiers are ordered according to their values.

Example 3. Let d be the XML document in Fig. 1, and let $Policies(d) = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$ be the access control policies applying to d. The `Policy` element that will be inserted into the security enhanced version of d is: $<$`Policy`$>1,2,3,4,5,6,7<$`/Policy`$>$. Thus, consider the element `Short-descr` of the DICO patent, whose corresponding policy configuration has value "90." The binary value corresponding to "9" is 1,001, this means that the policies that apply to this node are those whose identifiers are in the first and the fourth position in the `Policy` element, that is, P_1 and P_4.

We are now ready to formally define the security enhanced XML document.

Definition 4 (Security enhanced XML document). *Let $d = (V_d, \bar{v}_d, E_d, \phi_{E_d})$ be an XML document, the security enhanced (shortly SE-XML) version of d is an XML document $d' = (V'_d, \bar{v}'_d, E'_d, \phi'_{E'_d})$, such that:*

1. $V'_d = V'^{e}_d \bigcup V'^{a}_d$, *where:*
   - $V'^{e}_d = V^e_d \bigcup v_{pe}$, *where v_{pe} is a node representing a* `Policy` *element. v_{pe} is a direct child of $\bar{v}'_d$; and* $V'^{a}_d = V^a_d \bigcup V^a_{new} \bigcup V^a{}_{Sign}$, *where:*
     - V^a_{new} *contains* $\forall v \in V^e_d$:
       - * $\mathrm{PC}(v)$, *the policy configuration of v if: 1) there exists at least an access control policy in $\mathcal{PB}$ that applies to v, and 2) $v.content$ is not empty;* $\mathrm{PC}_{\mathrm{Attr}}(v)$, *the policy configuration of v's attributes, if element v contains at least an attribute to which an access control policy in $\mathcal{PB}$ applies.*
     - V^a_{Sign} *is an attribute containing the Merkle Signature of d, that is, the signature of $MhX_d(\bar{v}_d)$ with the Owner private key. V^a_{Sign} is a direct child of $\bar{v}'_d$.*
2. $\bar{v}'_d = \bar{v}_d$.
3. $E'_d = E_d \bigcup E'^{e}_d \bigcup E'^{a}_d$, *where:*
   - E'^{e}_d *contains the edge connecting $\bar{v}'_d$ to v_{pe}, whereas E'^{a}_d contains the edges connecting each node $v \in V^e_d$ to the nodes storing* $\mathrm{PC}(v)$, *and* $\mathrm{PC}_{\mathrm{Attr}}(v)$. *$E'^{a}_d$ also contains the edge connecting $\bar{v}'_d$ to the node storing the Merkle Signature of d.*
4. $\phi_{E'_d} : E_d \rightarrow Label^*$ *is the edge labeling function, where* $Label^* = Label(d) \bigcup \{$`Policy`, PC, $\mathrm{PC}_{\mathrm{Attr}}$, `Sign`$\}$, *and $Label(d)$ is the set of element tags and attribute names of document d.*

It is important to note that, in our approach, the Owner sends the Merkle Signature to the Publisher by inserting it inside the SE-XML version of the corresponding document (i.e, attribute `Sign`). Then, the Publisher returns it to subjects in the reply documents. An alternative solution for the distribution of the Merkle signatures is to release them directly to subjects during the subscription phase. This solution implies to send each subject the Merkle signatures of all the XML documents at the Owner source that the subject can access. This implies that a modification of the Owner source requires the recomputation and distribution of the Merkle signatures of all the documents affected by the modification to all the subjects entitled to access these documents. By contrast, the insertion of the Merkle Signature of a document into its SE-XML version drastically reduces the overhead necessary to manage the modification of the Owner source. Indeed, in this case, the Owner sends the Merkle signatures only to the Publishers entitled to manage the modified documents, along with the update versions of the corresponding SE-XML documents. Appendix I (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm) reports the SE-XML version of the document presented in Fig. 1, with respect to the access control policies in Fig. 3.

5.2 Secure Structure

To prove completeness of XML queries, a subject receives from the Publisher the secure structure of the XML documents on which the query is performed. In this section, we show how the secure structure is generated. The basic

5. The order is given by the policy identifier values.
6. If $ps(e)$ is not a half-byte-multiple, its last block is padded with 0s.

```
<?xml version="1.0" encoding="UTF-8" ?>
<x-82592553 x-4639474="rVR5DQ" x-67303774="QTQXS" Sign="OD2mc9aVV/tP4g3TG+1kr4sFhdio=">
  <Policy>1,2,3,4,5,6,7</Policy>
  <x-1915689488>
    <x-785490824 x-40276037="PlcZUo">
      <x-590292021 x-57205665="...." PC_ATTR="08" />
      <x-13947931>
        <x-1037159472 x-122584813="fENhtL" x-0260379="hgKID" x-93640287="..." PC_ATTR="080808" />
      </x-13947931>
    </x-785490824>
    <x-785490824 x-40276037="pKGEs">
      <x-590292021 x-57205665="...." PC_ATTR="02" />
      <x-13947931>
        <x-1037159472 x-122584813="gPd39" x-0260379="hgKID" x-93640287="..." PC_ATTR="060602" />
        <x-1037159472 x-122584813="o4GpM" x-0260379="yr0QjJ" x-93640287="..." PC_ATTR="060602" />
      </x-13947931>
    </x-785490824>
  </x-1915689488>
  <x-300913858>
    <x-873964128 x-0477503="PcXNh" x-74083499="...." x-96869452="PlcZUo" PC_ATTR="808080">
      <x-11915608 PC="90">.....</x-11915608>
      <x-1113276544 PC="80">.....</x-1113276544>
      <x-72048309 PC="90">....</x-72048309>
    </x-873964128>
    <x-873964128 x-0477503="WQloBY" x-74083499="...." x-96869452="pKGEs" PC_ATTR="202020">
      <x-11915608 PC="60">.....</x-11915608>
      <x-1113276544 PC="20">.....</x-1113276544>
      <x-72048309 PC="60">....</x-72048309>
    </x-873964128>
  </x-300913858>
</x-82592553>
```

Fig. 5. The secure structure of the XML document in Fig. 1.

idea is to supply the subject with the structure of the XML document on which the query is submitted where, with the term structure, we mean the XML document without data contents, that is, containing only the names of the tags and attributes of the XML document. The subject is then able to locally perform on the structure all queries whose conditions are against the document structure of the original document. Thus, the subject can match the result with the answer sent by the Publisher. Obviously, in such a basic approach, the subject is able to view the tag and attribute names of the whole document and thus also those referring to portions he/she may not be authorized to access. To overcome this drawback, the secure structure of the XML document is generated by hashing with a standard hash function each tag and attribute name. Since the value returned by the hash function may contain characters not allowed in an XML well-formed document[7] (e.g., "<","/"), during the generation of the secure structure the resulting hash values are encoded into an integer-based representation. Moreover, to be compliant with the XML syntax, we insert symbol "x" as a prefix of the integer before storing it as a tagname. Additionally, to extend the set of queries for which it is possible to prove completeness, we also insert the hashed attribute values of the XML document in the secure structure. The node-set returned by evaluating a query on the secure structure could be a superset of the nodes the subject is entitled to see according to the Owner access control policies. Thus, in order to verify the completeness, the subject must also consider the access control policies specified on the document. For this reason, the secure structure contains also the `Policy`, `PC`, and $\mathrm{PC}_{\mathrm{ATTR}}$ elements, whose tagname and content are not hashed. Additionally, in order to prevent alterations by the Publisher, the Owner computes the Merkle Signature of the secure structure. This signature is sent to Publishers together with the corresponding secure structure. Fig. 5 shows the secure structure of the XML document in Fig. 1.

7. A well-formed XML document is a document that follows the grammar rules of XML.

6 Subject-Publisher Interaction

When a subject s submits a query to a Publisher, the Publisher first determines the set of nodes that need to be returned to s. Such nodes are determined by evaluating the query on the SE-XML version of the requested document(s) and by pruning from the set of nodes returned by the evaluation, those nodes corresponding to portions for which s does not possess appropriate authorizations. Information on access control policies that apply to s are transmitted by s to the Publisher when submitting the access request. More precisely, the subject sends the Publisher his/her policy configuration together with the submitted query. Then, the set of nodes to be returned to s is complemented with additional information, that are used by s to authenticate the answer and to verify its completeness. In particular, all the additional information needed to verify the authenticity, as well as the nodes of the requested document(s) to be returned to the subject, are organized into an XML document, called *reply document*, described in the following section.

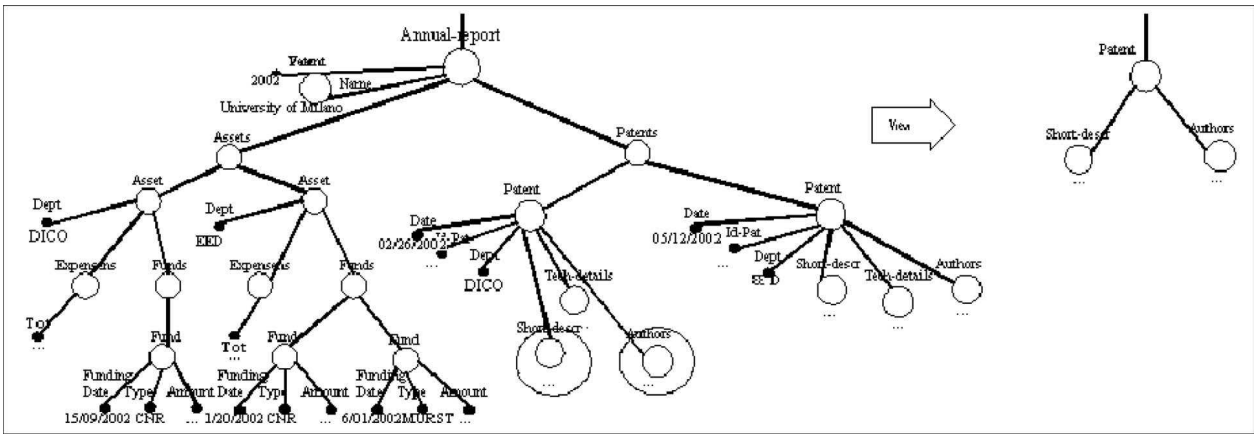

Fig. 6. An example of XML document returned by function $View()$.

6.1 Reply Document

For the sake of simplicity, in the following, we consider only queries on a single document. The methods we present, however, can be easily extended to queries referring to multiple documents. In general, an answer to a query may contain only selected portions of a given XML document. For instance, consider the XML document in Fig. 1, and suppose that an EED professor submits a query q asking for all DICO patents. According to the access control policies in Fig. 3, the nodes answering q and for which an EED professor has the necessary authorizations are the `Short-descr` and `Authors` elements. We assume the existence of a function, called $View()$, which takes as input a query q and the policy configuration of a subject s, and turns the set of nodes answer to q and for which s has the necessary authorizations into a well-formed XML document. The output of function $View()$ for the example above is presented in the right part of Fig. 6. In this document, the `Short-descr` and `Authors` elements are inserted as direct children of the node `Patent` from which all the other attributes/elements are pruned. Since all the other nodes are removed from the document, the subject is not able starting from the document returned by $View()$ to recompute the Merkle hash value of the root, and then compare it with the received Merkle Signature. It is thus necessary to return the subject some additional information together with the view that makes the subject able to perform the comparison.

This information is represented by the *relative Merkle hash path* defined in the following. In the definition, given a node w, we denote with $Path(w)$ the set of nodes connecting w to the root of the corresponding document. Moreover, given a node $v \in Path(w)$, we denote with $Path(w, v)$ the subset of nodes in $Path(w)$ connecting w to v.

Definition 5 (Relative Merkle hash path). *Let $g = (V_g, \bar{v}_g, E_g, \phi_{E_g})$ be an SE-XML document. Let $v \in V_g^e$ and $w \in V_g$ such that $v \in path(w)$. The relative Merkle hash path of w with regard to v, denoted as $MhPath(w, v)$, is a list of hash values containing all the elements in the set*

$$\{h(f.content), h(f.tagname) | f \in Path(w, v) \backslash w\} \bigcup$$
$$\{MhX_d(e) | e \in sib(f), f \in Path(w, v) \backslash v\},$$

where $sib()$ is a function that, given a node $f \in V_g$, returns f's siblings. The elements in the list are ordered with regard to the order defined in Section 2.1.

Intuitively, given two nodes v, w such that $v \in Path(w)$, $MhPath(w, v)$ is the set of Merkle hash values necessary to compute the Merkle hash value of v having the Merkle hash value of w. We now introduce the formal definition of reply document.

Definition 6 (Reply document). *Let $g = (V_g, \bar{v}_g, E_g, \phi_{E_g})$ be the SE-XML version of an XML document d, let s be a subject, and q be a query on d submitted by s to a Publisher. Let $View(q, s) = (V_q, \bar{v_q}, E_q, \phi_{E_q})$ be the XML document answer to q, according to the policy configuration of s. The* reply document of query q with respect to s *is an XML document $r = (V_r, \bar{v}_r, E_r, \phi_{E_r})$ such that:*

1. $V_r^e = V_q^e \bigcup V_{ATT}^e$, *where V_{ATT}^e contains a node, called* `AttributeElement`, *for each attribute $a \in V_q^a$. This node represents an element whose data content is the value of a. The name of a is stored into an additional attribute of* `AttributeElement`, *called* `AttrName`. *The node is a direct child of the node in V_r^e corresponding to the element in $View(q)$ to which a belongs to.*
2. $V_r^a = V^a{}_{Sign} \bigcup V_{MhPath}^a$, *where:*
   - $V^a{}_{Sign}$ *is an attribute which contains the value of attribute* `Sign` *in g.*
   - V_{MhPath}^a *contains an attribute, called* `MhPath`, *for each $e \in V_q^e$, with value:*
     - $\{MhX_d(child(j, w^g)) \mid j = 1, \ldots, Nc_{w^g}\} \bigcup MhPath(w^g, v^g)$, *if e is a terminal node in V_q; $MhPath(w^g, v^g)$, otherwise, with $v^g, w^g \in V_g$, such that w^g is the node corresponding to e in document g and:*
       - *if $e = \bar{v}_q$, then v^g is the root of document d; otherwise, let $f \in V_q^e$ be the father of e in document $View(q)$, then v^g is the node corresponding to f in document g.*
3. $\bar{v}_r = \bar{v}_q$.

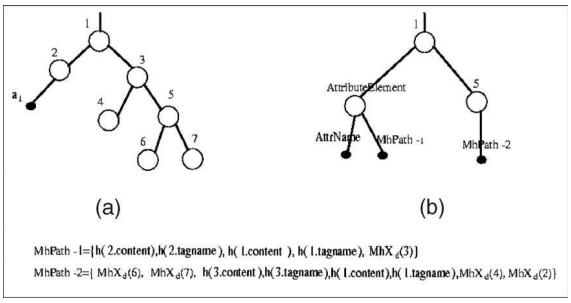

Fig. 7. An example of reply document.

4. $E_r = E_q^e \bigcup E_{\text{AttributeElement}} \bigcup E_{\text{MhPath}} \bigcup E_{\text{Sign}}$, *where* $E_{\text{AttributeElement}}$ *is the set of edges connecting* `AttributeElement` *elements with their parent,* E_{MhPath} *is the set of edges connecting* `MhPath` *attributes with the nodes in which they are contained, and* E_{Sign} *is the edge connecting attribute* `Sign` *with* $\bar{v}_r$.
5. $\phi_{E_r} : E_r \rightarrow \overline{Label}$ *is the edge labeling function, where*

$$\overline{Label} = Label(g) \bigcup \{\texttt{AttributeElement}, \texttt{MhPath}, \texttt{Sign}\}.$$

Note that a node belonging to the view returned by function $View(q)$ may not have as father the same node as in the original document because such node could be not answering the query q. In such a case, the reply document contains also the information regarding the pruned nodes, that is, the hash values belonging to the relative Merkle hash path of the considered node with respect to its father in the view. For instance, with reference to Fig. 6, consider the element representing the short description. The father of this node is the `Patent` element. Thus, the `MhPath` attribute associated with `Short-descr` contains all the hash values needed to compute the Merkle hash value of `Patent` starting from `Short-descr`. Since `Short-descr` element does not have children, by Definition 5, the `MhPath` attribute contains the Merkle hash values of its siblings, that is, the Merkle hash values of elements: `Tech-details`, `Authors`, and of attributes: `Date`, `Id-Pat`, and `Dept`. By Definition 2, with these values and with the `Short-descr` element, it is possible to compute the Merkle hash value of `Patent`. Moreover, if e is the root of the reply document, the related `MhPath` attribute contains the Merkle hash values necessary to compute the Merkle hash value of the root of document d starting from node e. Thus, referring to the above example, the `MhPath` attributes associated with `Patent` contain all the Merkle hash values necessary to compute the Merkle hash value of the root element (i.e., `Annual-report` element).

Since we support access control policies at the attribute level, all attributes in $View(q)$ are replaced in the reply document by particular elements, called `AttributeElement`, whose data content is the value of the attribute itself and whose `AttrName` attribute contains the name of the attribute. This operation allows one to associate also with the attributes in a query response the additional information needed for subject verification (i.e., `MhPath` attributes). Indeed, when different access control policies are applied to an element and to its attributes, a query may return an attribute, but not the element in which it is contained. As an example, consider the XML document presented in Fig. 7a, and suppose to submit a query returning attribute a_1 and elements 1 and 5. The corresponding reply document is presented in Fig. 7b. It is easy to verify that, by using the hash values contained into the `MhPath` attribute of the `AttributeElement` node corresponding to a_1, it is possible to compute the Merkle hash value of element 1 (i.e., $MhX_d(1)$). If access control policies are specified at the element-level only, there is no need to replace attributes in $View(q)$ by `AttributeElement` elements since, in such a case, each time a query is performed asking for an attribute, also the element in which the attribute is contained is returned.

6.2 Reply Document Generation

An algorithm for generating the reply document is presented in Fig. 8. The algorithm is executed by the Publisher and takes as input the query q submitted by a subject s, the SE-XML version of the document d on which the query is submitted, and the policy configuration of s. The algorithm consists of three main steps, during which it uses two functions, called $Evaluate()$ and $ReBuild()$. Function $Evaluate()$ takes as input a query q submitted on document d, and returns a well-formed XML document r containing all and only the nodes satisfying q. Step 3 determines which access control policies apply to each node satisfying the query q. Then, it removes from the node-set satisfying query q the nodes that s is not authorized to access, on the basis of the information in the SE-XML version of d. Then, all the attributes in the resulting document are replaced with an `AttributeElement` element (Step 4). Moreover, an additional attribute, called `MhPath`, is inserted in each node to be returned to s (Step 5). The value of this attribute is set according to Definition 6. The last information needed by s to verify authenticity and completeness of query result is the Merkle Signature of document d. This value is inserted in Step 6. Finally, in Step 7, function $ReBuild()$ takes as input the obtained set of nodes and transforms them into a well-formed XML document.

Fig. 9 shows the reply document returned by a Publisher to an EED professor, who requests all the patents of DICO.

It is important to note that for queries returning a complete subtree, the Publisher can generate a simplified version of the reply document (with regard to the one generated by Algorithm 1). Indeed, by Definition 2, the Merkle hash value of a node depends on all and only its descendants. Thus, by having all the nodes belonging to a subtree, it is possible to compute the Merkle hash value of the root of the subtree. For this reason, in this simplified version, all `MhPath` attributes are omitted, with the exception of the one in the root $\bar{v}_q$ of the subtree returned by the query.

7 Subject Verification

In this section, we illustrate how a subject, upon receiving a reply document and a secure structure, can verify the authenticity and the completeness of the corresponding query answer.

7.1 Authenticity Verification

Before presenting an algorithm that allows a subject to authenticate the answer to a query, we need to introduce the concept of authenticable element.

Algorithm 1: The Reply Generator Algorithm

INPUT: 1. The SE-XML version $g = (V_g, \overline{v}_g, E_g, \phi_{E_g})$ of an XML document d;
2. A query q on d submitted by s; 3. $PC(s)$ the policy configuration of s

OUTPUT: The reply document r

1. $PolConf$ is initialized to be empty
2. Let $r = (V_r, \overline{v}_r, E_r, \phi_{E_r})$ be the XML document returned by function $Evaluate(q)$
3. **For** each $v \in V_r$:
   A Let $v^g \in V_g$ be the node corresponding to v in document g
   B **If** $v^g \in V_g^e$: **For** each $c \in v.$`PC`: $PolConf := PolConf || HexToBin(c)$
   else:
   a Let w^g be the father of v^g
   b Let j be the relative order of v^g wrt its siblings
   c Let $PolString$ be the j-th policy string extracted from $w^g.PC_{Attr}$
   d **For** each $c \in PolString$: $PolConf := PolConf || HexToBin(c)$
   endif
   C $Find := 0$
   D **For** $n = 1$ to $length(PolConf)$:
   a $Bit := nextBit(PolConf)$
   b **If** $Bit := 1$:
   i Let y be the n-th identifier stored in `Policy` element
   ii **If** y appears in $PC(s)$: $Find := 1$, Break
   endfor
   E **If** $Find = 0$: Remove v from V_r
   F $PolConf$ is initialized to be empty
   endfor
4. **For** each $a \in V_r^a$:
   A Create a new node e with tag `AttributeElement`
   B $e.content := a.val$
   C Create a new attribute in e with name `AttrName`
   D $e.$`AttrName`$.val := a.name$
   E Add e to V_r, remove a from V_r
   endfor
5. **For** each $w \in V_r$:
   A Let $w^g \in V_g$ be the node corresponding to w in document g
   B **If** $w = \overline{v}_r$: $v^g = \overline{v}_g$
   else:
   a Let v be the father of w in document r
   b Let $v^g \in V_g$ be the node corresponding to v in document g
   endif
   C Add attribute `MhPath` to node w
   D $w.$`MhPath` $:= MhPath(w^g, v^g)$
   endfor
6. Add the attribute `Sign` to $\overline{v}_r$, and set its value equal to attribute `Sign` in document g
7. $\overline{r}$:=ReBuild(r)
8. **Return**($\overline{r}$)

Fig. 8. The Reply Generator Algorithm.

Definition 7 (Authenticable element). *Let $d = (V_d, \overline{v}_d, E_d, \phi_{E_d})$ be an XML document, let $g = (V_g, \overline{v}_g, E_g, \phi_{E_g})$ be the SE-XML version of d, and $r = (V_r, \overline{v}_r, E_r, \phi_{E_r})$ be the reply document corresponding to a query submitted on d by a subject s. Let V_T be the set of terminal nodes of r. For each $v \in V_r^e$, v is* authenticable by s, *iff there exists $v_t \in V_T$, with $v \in path(v_t)$, such that it is possible, through a recursive bottom-up computation, to compute the Merkle hash value of $\overline{v}_d$ using only the values in $\{w.$`MhPath`$|w \in path(v_t)\}$.*

Note that authenticability is required only for the nodes of the reply document that represent elements. Indeed, attribute nodes in the reply document (i.e., `MhPath` attributes) are inserted only to store values needed to check the authenticity of the answer. The following theorem states the authenticability of each element of a reply document.

Theorem 1. *Let $g = (V_g, \overline{v}_g, E_g, \phi_{E_g})$ be the SE-XML version of an XML document d, and $r = (V_r, \overline{v}_r, E_r, \phi_{E_r})$ be the reply document corresponding to a query submitted on d by a subject s. Each node in V_r^e is authenticable by s.*

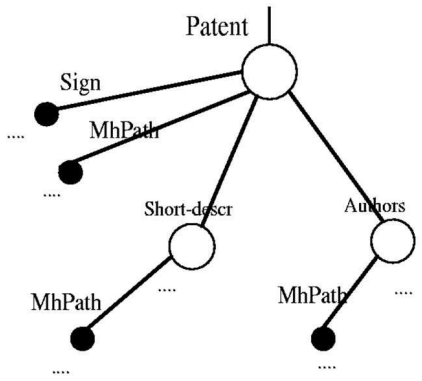

Fig. 9. An example of reply document.

The following definition states that a node in a reply document is authentic if it is authenticable by the subject receiving the document, and the recursive bottom-up computation returns a value matching the Merkle Signature of the requested document.

Definition 8 (Authentic element). *Let $g = (V_g, \overline{v}_g, E_g, \phi_{E_g})$ be the SE-XML version of an XML document $d = (V_d, \overline{v}_d, E_d, \phi_{E_d})$, and $r = (V_r, \overline{v}_r, E_r, \phi_{E_r})$ be the reply document corresponding to a query submitted on d by a subject s. Each node $v \in V_r^e$ is authentic iff: 1) v is authenticable by s and 2) the computed Merkle hash value of $\overline{v}_d$ is equal to the decryption of* `Sign`$.val$ *using the Owner public key.*

An algorithm for verifying the authenticity of a query answer is presented in Fig. 10. The algorithm takes as input a reply document r, returned as answer to a query on document d, and returns *True* if all the elements in the reply document are authentic. It returns *False*, otherwise. Starting from each terminal node in the reply document, the algorithm recomputes the Merkle hash value of the root of d through a bottom-up computation that uses the values of attributes `MhPath` of each node belonging to the path connecting the terminal node to the root of the reply document. In the algorithm, there is the need of knowing where the considered element is located with regard to its siblings, in order to correctly compute its Merkle hash value. Due to this reason, in the Merkle hash path, we store this information also. To do that, we use symbol "/" to denote a subsequent level in the hierarchy, and symbol "#" to denote the position of the element with regard to its siblings. Then, the obtained value is compared with the decryption of the Merkle Signature of d, using the Owner public key[8] (the Merkle Signature of d is stored into attribute `Sign`): If the two values coincide, then all the nodes belonging to the path are authentic. Otherwise, the algorithm terminates and returns $False$.

To clarify how the algorithm works, consider a terminal node v_t and suppose that $path(v_t) = \{v_1, \ldots v_n\}$, where v_{j+1} is the father of v_j, $j = 1, \ldots, n-1$. According to Definition 7.1, to compute the Merkle hash value of node v_j ($j = 1, \ldots, n-1$), the algorithm uses the `MhPath` attributes of the nodes belonging to $path(v_t)$. By Definition 6, the `MhPath` attribute of a node v_{j-1} contains $MhPath(v_{j-1}^g, v_j^g)$, where v_{j-1}^g, v_j^g are the nodes corresponding to v_{j-1} and v_j in SE-XML document g, respectively. Between v_{j-1}^g and v_j^g, there could exist some intermediate nodes in document g, which could be removed from the reply document r by Algorithm 1. To correctly compute the Merkle hash value of node v_j, it is necessary to also consider the Merkle hash values of these removed nodes. We denote by $level_k$ the subset of Merkle hash values belonging to $MhPath(v_{j-1}^g, v_j^g)$, which are necessary to compute the Merkle hash value of node v_k^g, where $v_k^g \in path(v_{j-1}^g, v_j^g)$. Step $2.B.a$ is iterated for each node v_k belonging to $path(v_{j-1}^g, v_j^g)$. At each iteration function, *Extract-next-level()* extracts $level_k$ from v_{j-1}.`MhPath`, for each $v_k \in path(v_{j-1}^g, v_j^g)$, starting from the father of v_{j-1}^g and going up in the tree till node v_j^g is reached. The inner while cycle extracts each Merkle hash value from $level_k$ (using function *Extract-Next-Merkle()*) and concatenates all the extracted values into variable $Merkle$. When cycle $2.B.a$ terminates, variable $Merkle$ stores the Merkle hash value of node v_j. The cycle is iterated for each node in $path(v_t)$. Thus, at the end of cycle $2.B$, variable $Merkle$ stores the Merkle hash value of the root of d. Then, Step $2.C$ compares the computed value against the value generated from the decryption with the public key of the Owner of the Merkle signature of d, stored in the `Sign` attribute in the root of the reply document. If the two values are equal then another path is considered. Otherwise, the algorithm returns $False$. If, for all the paths in the reply document, the condition in step $3.C$ is not verified, then the algorithm returns $True$. An example of authenticity verification is reported in Appendix II (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm). The following theorem states that Algorithm 2 can be used to verify the authenticity of an answer returned by a Publisher.

Theorem 2. *Let s be a subject, q be a query submitted by s, and r be the reply document received by s as answer to q. Algorithm 2 returns $True$ iff each $v \in V_r^e$ is authentic.*

7.2 Completeness Verification

Completeness verification is a difficult issue and greatly depends on the kinds of queries that are submitted to a Publisher.[9] We need, thus, to point out for which kind of queries the subject is able to verify the completeness of the answer. In general, queries on an XML document can be classified into two groups: queries depending on the structure of the XML document, and queries depending on the content. The first kind of queries, to which we refer as *structure queries*, impose conditions only on the structure of the XML document. According to the XPath syntax, structure queries are defined by means of XPath expressions that specify only the location path. Consider the XML document presented in Fig. 1, an example of structure query is: "`//Patent//*`", which returns the set of all the descendent elements of the `Patent` element. By contrast, in the second type of queries, that we call *content-dependent queries*, besides the structure, it is also possible to specify conditions on the content of the elements and/or attributes in the path. Among content-dependent queries, we are able to verify completeness only for those queries that state equality conditions on

8. In the algorithm, $D_{KU_{Owner}}$[`Sign`$.val$] denotes the decryption of the Merkle Signature of d with the public key of the Owner.

9. We assume that subjects submit queries to the Publishers by means of XPath expressions.

INPUT: A reply document $r = (V_r, \bar{v}_r, E_r, \phi_{E_r})$
OUTPUT: $True$ if all the nodes in r are authentic, $False$ otherwise
1 Let $V_T = \{v | v \in V_r^e$ AND each children of v belongs to $V_r^a\}$
2 **For** each $v_t \in V_T$:
A **If** $v_t.tagname =$ `AttributeElement`: $MyMerkle := h(h(v_t.content)||h(v_t.\texttt{AttName}.val))$
else: $MyMerkle := h(v_t.content)||h(v_t.tagname)$
B **For** each $w \in Path(v_t)$:
a **While** ((*curr-level*:=*Extract-next-level*(w.`MhPath`))$\neq 0$):
i) **While** ((*MhX*:=*Extract-next-Merkle*(curr-level))$\neq 0$):
a **If** $MhX =$'#': $Mhx := MyMerkle$
b $Merkle := Merkle||MhX$
endwhile
ii) $Merkle := h(Merkle)$
endfor
C **If** $(Merkle \neq D_{KU_{Owner}}[\texttt{Sign}.val])$: **Return**($False$)
endfor
3 **Return**($True$)

Fig. 10. The Subject Verification Algorithm.

the attribute values (hereafter, we denote this kind of queries with the term *content-dependent** queries). If we consider again the XML document in Fig. 1, an example of content-dependent* query is: "`//Asset[@Dept="DICO"]//*`", that selects a set of subtrees rooted at any descendent of the `Asset` element, whose `Dept` attribute has value "DICO."

Now, we can show how it is possible to use the secure structure to prove the completeness of structure and content-dependent* queries. Both kinds of queries specify their conditions by using only the tag/attribute names, and the attribute value, which are information all contained in the secure structure as hash values. The proposed solution for completeness verification implies the translation of the query q, by substituting the tag/attribute names and attribute values with the corresponding hash values. Afterward, the translated query can be evaluated on the secure structure. In such a way, under the assumption of a collision-resistant hash function, the node-set resulting by the evaluation of the query on the secure structure corresponds to all and only the nodes of the requested document, answering query q. This node-set can be a superset of the nodes the subject is entitled to see according to the Owner access control policies. For this reason, in order to verify the completeness of the answer, the subject must consider also the access control policies specified on the document. The secure structure contains the `Policy` element, which contains the identifiers of the access control policies that apply to the document. Therefore, by considering the `Policy` element and the values of the `PC` and PC_{ATTR} attributes of each node in the secure structure, and by matching these values with his/her subject policy configuration, the subject is able to verify which are the document nodes for which he/she has the appropriate authorizations. According to this approach, the subject is able to determine the set of nodes that should be returned by the Publisher.

Before verifying completeness of the answer with the above-mentioned approach, a preliminary required step is that the subject verifies the authenticity of the secure structure itself using the received signature and the strategy for authenticity verification we have illustrated in Section 7.1. In this way, he/she is assured that the Publisher has not modified the secure structure given by the Owner.

We do not report here the algorithm for completeness verification since it is similar to the algorithm for reply document generation presented in Section 6.2. The main difference is that, instead of evaluating the submitted query on the secure enhanced document, the completeness verification algorithm evaluates the translated query on the secure structure.

Example 4. Consider the secure structure presented in Fig. 5. Suppose that an EED junior secretary submits a query asking for all the Fund items contained in the annual-report. Suppose, moreover, that an untrusted Publisher sends her a reply document containing only the first `Fund` element. The completeness verification process executed by the secretary first verifies the authenticity and integrity of the secure structure. Then, it considers the query. In this example, the query is "`/Asset[@Dept='EED']/Funds/*`", to which, according to the selected hash function, the following translation corresponds: "`//x-785490824[@x-40276037='pKGEs']/x-13947931/*`." The evaluation of this query on the secure structure presented in Fig. 5 returns two nodes with tagname `x-1037159472`. Since both the nodes have no content-element, they have associated only the policy configuration of their attributes (contained in attribute PC_{ATTR}), whose value is "060602." The only policy applied to the junior secretary is the one with 7 as the identifier. Thus, considering the value of the `Policy` element, the nodes for which the secretary has an authorization are those whose policy configuration has the seventh bit set equal to 1. This implies that the binary value of the second character of PC_{ATTR} value must have the third bit equal to 1. The binary string corresponding to 06 is 0110. This implies that the junior secretary should access both the

elements. Thus, the completeness verification returns a node-set containing the two nodes with tagname `x-1037159472`. Finally, the last step of completeness verification is to hash the answer received by the Publisher, and match it with the obtained node-set. The junior secretary verifies that, in the answer received by the Publisher an element is omitted, more precisely, a `Fund` element.

8 Attack Analysis

In this section, we analyze the potential attacks that could be conducted by the Publishers or by the subjects.

8.1 Subject Attacks

In our architecture, subjects interact both with the Publishers and with the Owner. We need, thus, to consider the possible attacks carried out in both interactions. In the Owner-subject interaction, there exist two different kinds of attacks. The first kind is an authentication attack, and it implies the existence of a malicious subject S_m, which eavesdrops during the subscription phase of a subject S_i, the subject policy configuration. Thus, S_m can submit queries to the Publishers exploiting the authorizations granted to S_i. The solution for this kind of attacks relies on the adoption of a standard authentication protocol [15] during the subscription phase.

Another possible kind of attack carried out by a subject implies that a malicious subject makes available his/her subject policy configuration to other subjects. To avoid this attack, we can impose that the submitted queries, as well as the subject policy configurations, must be signed by the subject submitting the query.

We need to also consider the case in which one or more authorizations are revoked from a subject. In this case, the Owner regenerates the subject policy configuration, which is then sent to the subject. In such a scenario, it is necessary to avoid that a subject continues to use the old subject policy configuration, exploiting also the revoked authorizations. To this purpose, we assume that the Owner inserts into the subject policy configuration also a subject ID and the timestamp of the (re)generation. Then, the Owner stores in a public repository each subject ID and the timestamp of his/her latest subject policy configuration. Moreover, when a subject is revoked, the Owner removes the relative entry from the public repository. In such a way, the Publisher can verify the validity of the subject policy configuration received by the subject.

Finally, another kind of subject attack is a passive inference of information from the secure structure. Indeed, during completeness verification, the subject computes the hash values corresponding to the names of the nodes belonging to its authorized view. Then, by using such hash values, the subject is able to deduct the existence and the cardinality of these nodes in the secure structure. To avoid information inference, from the secure structure, it is possible to apply cryptographic techniques (see Section 11).

8.2 Publisher Attacks

In this paper, we have shown that authenticity verification relies on the comparison between the Merkle hash values locally computed by a subject by using the information contained in the `MhPath` attributes, and the Merkle Signature contained in the `Sign` attribute, where all these attributes are contained into the Reply document. Thus, a possible attack conducted by a Publisher is the replacement of the `Sign` element associated with an SE-XML document. In this case, even if the query result is authentic, the authenticity verification will always fail.[10] To prevent this attack, we can associate with each XML document a unique ID, by inserting it as an attribute of its root. Thus, the Merkle hash value of the root of an XML document is computed also on the hash value of its ID. Furthermore, the Owner must also sign, in addition to the Merkle hash value of the root of the XML document, the corresponding ID value. The resulting signature is then stored in the `Sign` element. The Publisher then inserts in the Reply document, together with the `Sign` element, also the attribute containing the corresponding ID. In such a way, the subject can verify at first the authenticity of the answer by using the Merkle hash value of the root. If the authentication process fails, he/she can verify if the ID value signed by the Owner matches the one in the Reply document and, thus, he/she can verify whether the above attack has occurred.

9 Performance Issues

In this section, we discuss the overhead implied by our approach.

9.1 Update Management

Since, in the proposed framework, a modification on a document also implies an update of the security-enhanced version and of the secure structure of the document, a key issue is the efficient management of updates. To this purpose, we discuss, for each kind of update that could occur over the Owner's XML source or the Policy Base, the corresponding updates that the Owner has to perform on the security-enhanced version and on the secure structure of the involved documents. There are two main kinds of updates that need to be considered. The first is an update of the $\mathcal{PB}$, such as, for instance, an access control policy insertion, deletion, or update. In the case of policy insertion, it is necessary to update the SE-XML version and the secure structure of all the documents to which the new policy applies. However, this update does not require too much overhead. Indeed, for each document to which the policy applies, it is only necessary to insert the identifier of the new policy in the `Policy` element, and to update all and only the policy configurations of those nodes to which the new access control policy applies. Management of policy updates is simpler since it is not necessary to update the `Policy` element, but only the policy configurations of those nodes that are influenced by the policy update. Moreover, in case of policy deletion, it is only necessary to set equal to zero the identifier of the deleted policy in the `Policy` element of the SE-XML version and of the secure structure of all the documents to which the revoked policy applied. All $\mathcal{PB}$ updates do not require to recompute the Merkle Signature of the involved documents since this value is computed over the root of the original XML document, thus it does not include the policy configuration values. Thus, upon a modification of $\mathcal{PB}$, the Owner sends the Publishers only the `Policy` element of all the documents affected by the policy updates and, only in case of policy insertion and

10. Note that the same attack can also be applied to the Merkle Signature of the secure structure.

modification, a set of policy configurations, one for each node affected by the policy update.

Note, moreover, that our credential-based access control model makes the management of subject subscriptions very efficient. Indeed, each time a new subject subscribes to the service or an existing subject is removed from the system, it is not necessary to update the SE-XML version and the secure structure of the documents on which the subject has access, since policies are not specified according to an identity-based mechanism.

Another kind of modification that implies an update of the SE-XML version and of the secure structure of a document is the update of the documents belonging to the Owner's source. These modifications can result in the insertion and deletion of a node of a document or the update of the data content of an element or of the value of an attribute. All these updates imply the recomputation of the Merkle Signature of the document that is sent to the Publishers along with the updated nodes of the corresponding SE-XML document and of the secure structure.

9.2 Storage Complexity of Security Related Information

Another relevant issue is the size of the SE-XML document, the secure structure, and the reply document with regard to the size of the original XML document.

Let us first consider the SE-XML document. Let $d = (V_d, \bar{v_d}, E_d, \phi_{E_d})$ be an XML document, and N_e be the cardinality of V_d^e. To create the SE-XML version of d, the Owner adds to d one attribute for the Merkle Signature, and at most $2N_e$ attributes representing the `PC`, and $\mathrm{PC}_{\mathrm{Attr}}$ attributes, respectively. Moreover, the SE-XML version of d contains an additional element, i.e., the `Policy` element. Thus, the information, which more influences the size of the SE-XML document, is the element/attribute policy configuration. More precisely, let N_p be the number of access control policies that apply to document d. The size of the policy configuration of an element e in d is $\frac{N_p}{4}$ characters. The policy configuration of attributes of an element e is given by the concatenation of the policy configurations of each attribute of e. Thus, let N_a be the number of attributes associated with an element e, the size of the policy configuration of attributes of e is $N_a \times \frac{N_p}{4}$ characters. Thus, in the worst case, that is, when for each element/attribute at least a policy is specified, the size of the original document is increased of $N_e(\frac{N_p}{4} + N_{Att} \times \frac{N_p}{4})$ bytes, where N_{Att} is the maximum number of attributes associated with an element of the document. It should be noted, however, that we expect that, in most cases, both N_{Att} and N_p be very limited because the number of different policies that usually apply to a document, even in the case of documents with heterogeneous security requirements, is not very high nor the number of attributes associated with an element. Moreover, the increased document size is compensated by the fact that, once the Publisher has received the SE-XML version of a document, it can locally answer subject queries, filtering query answers according to the access control policies of the Owner, and making a subject able to verify query authenticity by considering only the information in the SE-XML document and without interacting with the Owner. Therefore, the additional information appended by the Owner to documents are instrumental in reducing the number of interactions with the Owner.

Let us consider now the storage overhead of the secure structure. Similar to the SE-XML document, the secure structure also contains policy information and the Merkle Signature. This means that, in addition to the `Sign` element, it contains a unique `Policy` element, and at most $2N_e$ additional attributes representing the `PC`, and $\mathrm{PC}_{\mathrm{Attr}}$ attributes, respectively. The storage increase due to this information is thus equal to that calculated for the SE-XML document. We need also to make two considerations. The first is that, according to its definition, the secure structure does not contain data content. And, the second is that the Publisher sends the secure structure to a subject only the first time a subject submits a query on the corresponding document.

Finally, we consider the storage overhead in the reply document. In the proposed framework, in order to make a subject able to verify the authenticity of a query result, the Publisher adds to the query result some additional information, i.e., the `MhPath` attributes. These attributes contain a set of hash values. All these hash values have the same size, denoted in the following as $HashSize$.[11] Thus, according to Definition 6, given a terminal node e, the maximum size of `MhPath` is equal to $HashSize * Nc_e + sum_{f \in path(e)} HashSize * (2 + |sib(f)|)$, where $|sib(f)|$ denotes the number of f's siblings. It is, however, important to note that it is possible that the sets of Merkle hash values stored into different `MhPath` attributes be not disjoint. In order to reduce this redundancy, in the prototype system we have developed, we have adopted an approach that makes us able to store each hash value only in one `MhPath` attribute of the reply document. According to this approach, in the generation phase of the `MhPath` attributes before inserting an hash value, we verify if this value is already stored into another previously computed `MhPath` attribute. If this is the case, we insert in the new `MhPath` attribute only a reference to such an existing value. By this approach, the number of hash values stored in all the `MhPath` attributes of a reply document is, in the worst case, equal to the number of nodes of the original document d. Thus, in the worst case, the size of the information added to a query answer is $N_d \times HashSize$, where N_d is the number of nodes of d on which the query is performed.

10 Related Work

Merkle hash trees are a well-known mechanism used in several computer areas for certified query processing. For instance, it has been exploited by Naor and Nissim in [14] to deal with the problem of creating and maintaining efficient authenticated data structures holding information about the validity of certificates. More precisely, the paper proposes as data structure a sorted hash tree scheme, such that tree leaves correspond to revoked certificates. Thus, verifying that a certificate is revoked or not implies verifying the existence of certain leaves in the tree. A similar approach has been proposed by Devanbu et al. [8] to prove the completeness and authenticity of queries on relational data. Similar schemes have also been used for micropayments [5], where Merkle hash trees are used to minimize the number of public key signatures that are required in issuing or authenticating a sequence of certificates. By contrast, the use of such trees for handling XML documents is still a novel aspect, which, to the best of our knowledge, has been so far

11. The size of the hash value depends on the algorithm used. It could be 128 or 160 bits.

investigated only by Devanbu et al. in [9]. In this work, the authors have developed a scheme, based on Merkle hash trees, allowing clients to validate the answers to certain type of queries against XML sources managed by untrusted publishers. The method developed in [9] is based on the definition of a data structure, called "xtrie," which stores the set of possible paths that can be specified on a given DTD. However, the work presented in [9] has many differences with regard to our proposal. A first difference is the type of XML documents supported by the two approaches. In our approach, we have no limitation on the structure of XML documents, whereas the approach presented in [9] does not consider attributes and it imposes that data content be only present in leaf nodes. Another important difference is the kinds of queries for which the subject is able to verify authenticity. In our approach, we can certify the authenticity for each possible kind of XPath queries, whereas the approach presented in [9] considers only queries returning whole subtrees. A further distinction is that we also consider completeness with regard to access control rights, besides data authentication, and we provide a comprehensive architecture and related mechanisms to support data authentication and completeness services.

11 Conclusions

In this paper, we have presented an approach for secure Web publishing of XML documents. With a set of digital signatures generated by the Owner and no trust required for the Publisher, we have shown that a subject can verify the authenticity of a query response. Additionally, for a wide range of XPath queries, a subject is also able to verify the completeness of a query result, with respect to the access control policies stated by the information Owner. This is obtained using secure structures. Making a distinction between the Owner and the Publisher offers two benefits. First, in any decentralized architecture, such a solution offers the advantage of being scalable and of reducing the risk that the Owner becomes the bottleneck of the entire system. Second, this architecture does not require the Publisher to be trusted, with respect to document authenticity and completeness. We have already developed a Java-based prototype implementation of the architecture proposed in this paper, where we make use of DOMHASH [12] for the computation of the Merkle hash values.

We plan to extend the research described in this paper along several directions. First, we plan to develop a more comprehensive service for confidentiality. Such a task requires addressing two main issues. First, the Owner must be sure that each access granted by the Publisher agrees with the access control policies it specified. Second, a subject must be able to verify that a Publisher does not prevent him/her to access a document portion for which the subject has a proper authorization according to the policies specified by the Owner. In the work presented in this paper, we mainly focus on client-side verification, thus we deal only with the second issue. However, we plan to also investigate the first issue. The idea is to use a solution based on cryptographic techniques by applying an approach similar to that proposed by us in [2] for client/server architectures. According to such a paradigm, different portions of the same XML document are encrypted with different encryption keys, according to the access control policies applied on them, then the same encrypted copy of the XML document is released to clients. In the third party scenario, this means that a Publisher receives an encrypted copy of the data it is allowed to manage. Moreover, in order to allow a Publisher to perform queries on encrypted data, without having the encryption keys, we plan to adopt polymorphic cryptography techniques, such as the one adopted in [10], [11].

Then, we plan to perform an extensive performance analysis of the proposed framework. Additionally, we intend to integrate the Owner-Publisher system into [4]—an XML document server providing a comprehensive environment for securing XML documents. In this context, an interesting issue is how to detect and prevent inference of unauthorized data from the data source. This is the case, for instance, of queries stating conditions on unauthorized attributes/elements, whose answers make the subjects able to inference avoided information. Literature and commercial DBMSs propose several solutions to deal with this issue, such as, for instance, the approaches presented in [16], [17]. Thus, we plan to investigate how to adopt these approaches in our framework. Moreover, we plan to investigate the problem of completeness verification for more generalized classes of queries. We also plan to integrate our system with platforms for privacy preferences [6] to ensure privacy of subjects credential, and with anonymity services [7], as well as with evolving XML standards.

Acknowledgments

This work has been partially supported by a grant from Microsoft Research.

References

[1] World Wide Web Consortium, Xml, http://www.w3.org/XML, 2004.
[2] E. Bertino, B. Carminati, and E. Ferrari, "A Temporal Key Management Scheme for Secure Broadcasting of XML Documents," *Proc. Ninth ACM Conf. Computer and Comm. Security,* pp. 31-40, 2002.
[3] E. Bertino, S. Castano, and E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-Based Language," *Proc. Sixth ACM Symp. Access Control Models and Technologies,* pp. 57-65, 2001.
[4] E. Bertino, S. Castano, and E. Ferrari, "Authorx: A Comprehensive System for Securing XML Documents," *IEEE Internet Computing,* vol. 5, no. 3, pp. 21-31, May/June 2001.
[5] S. Charanjit and M. Yung, "Paytree: Amortized Signature for Flexible Micropayments," *Proc. Second Usenix Workshop Electronic Commerce,* 1996.
[6] L. Cranor and J. Reagle, "The Platform for Privacy Preferences," *Comm. ACM,* vol. 42, no. 2, pp. 48-55, 1999.
[7] L. Cranor and P. Resnick, "Protocols for Automated Negotiations with Buyer Anonymity and Seller Reputations," *Proc. Telecomm. Policy Research Conf.,* Sept. 1997.
[8] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine, "Authentic Third-Party Data Publication," *Proc. 14th Ann. IFIP WG 11.3 Working Conf. Database Security,* Aug. 2000.
[9] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine, "Flexible Authentication of XML Documents," *Proc. Eighth ACM Conf. Computer and Comm. Security,* 2001.
[10] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database Service Provider Model," *Proc. SIGMOD Conf.,* 2002.
[11] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing Database as a Service," *Proc. Int'l Conf. Data Eng.,* 2002.
[12] H. Maruyama, K. Tamura, and N. Uramoto, "Digest Values for Dom (Domhash)," Network Working Group, http://www.ietf.org/rfc/rfc2803.txt, 2004.
[13] R. Merkle, "A Certified Digital Signature," *Proc. Conf. Advances in Cryptology (Crypto '89),* 1989.

[14] M. Naor and K. Nissim, "Certificate Revocation and Certificate Update," *Proc. Seventh USENIX Security Symp.*, 1998.
[15] W. Stallings, *Network Security Essentials: Applications and Standars.* 2000.
[16] B. Thuraisingham, "The Use of Conceptual Structures for Handling the Inference Problem, and Cover Stories for Database Security," *Proc. Fifth IFIP WG 11.3 Working Conf. Database Security,* 1991.
[17] B. Thuraisingham, "Security Checking in Relational Database Management Systems Augmented with Inference Engines," *Computers and Security,* vol. 6, pp. 479-492, 1987.

Elisa Bertino is a professor of database systems in the Department of Computer Science and Communication at the University of Milan, where she is currently the chair of the department and the director of the DB&SEC Laboratory. She has been a visiting researcher at the IBM Research Laboratory (now, Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Purdue University, and at Telcordia Technologies. Her main research interests include security, privacy, database systems, object-oriented technology, and multimedia systems. In those areas, Professor Bertino has published more than 200 papers in all major refereed journals, and in proceedings of international conferences and symposia. She is a coauthor of the books *Object-Oriented Database Systems—Concepts and Architectures* (1993, Addison-Wesley), *Indexing Techniques for Advanced Database Systems* (1997, Kluwer Academic), and *Intelligent Database Systems* (2001, Addison-Wesley). She is a coeditor-in-chief of the *Very Large Database Systems (VLDB) Journal* and a member of the advisory board of the *IEEE Transactions on Knowledge and Data Engineering*. She serves also on the editorial boards of several scientific journals, incuding *IEEE Internet Computing*, *ACM Transactions on Information and System Security*, *Acta Informatica*, the *Parallel and Distributed Database Journal*, the *Journal of Computer Security*, *Data & Knowledge Engineering*, the *International Journal of Cooperative Information Systems*, and *Science of Computer Programming*. She has been a consultant to several Italian companies on data management systems and applications and has given several courses to industries. She is involved in several projects sponsored by the European Union (EU). She is a fellow of the IEEE and a member of the ACM, and has been been named a Golden Core Member for her service to the IEEE Computer Society. She has served as a program committee member of several international conferences, such as ACM SIGMOD, VLDB, ACM OOPSLA, as program cochair of the 1998 IEEE International Conference on Data Engineering (ICDE), as program chair of 2000 European Conference on Object-Oriented Programming (ECOOP 2000), and as program chair of the Seventh ACM Symposium of Access Control Models and Technologies (SACMAT 2002). She will be serving as program chair of the 2004 EDBT Conference.

Barbara Carminati received the MS degree in computer sciences in 2000, and the PhD degree in computer science from the University of Milano, in 2004. She is an assistant professor of computer science at the University of Insubria at Como, Italy. Her main research interests include: database and Web security, XML, secure information dissemination, and publishing. She is also a lecturer at the Computer Science School of the University of Milano and University of Insubria at Como, and she has taught industrial courses on topics such as database systems and security. She has served as a program committee member for the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT '04).

Elena Ferrari received the PhD degree in computer science from the University of Milano, in 1997. She is a professor of database systems at the University of Insubria at Como, Italy. She has also been on the faculty in the Department of Computer Science at the University of Milano, Italy, from 1998 to March 2001. She has been a visiting researcher at George Mason University, Fairfax (Virginia), and at Rutgers University, Newark (New Jersey). Her main research interests include database and Web security, temporal and multimedia databases. In those areas, Professor Ferrari has published several papers in all major refereed journals, and in proceedings of international conferences and symposia. She is on the editorial board of the *VLDB Journal* and the *International Journal of Information Technology (IJIT)*. She has served as program chair of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT '04), COMPSAC '02 Workshop on Web Security and Semantic Web, the first ECOOP Workshop on XML and Object Technology, and the first ECOOP Workshop on Object-oriented Databases. Dr. Ferrari was also the general chair of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT '03) and the Software Demonstration Chair of the 10th International Conference on Extending Database Technologies (EDBT '04). She has also served as program committee member for several international conferences. She is a member of the ACM and the IEEE.

Bhavani Thuraisingham is the Program Director in Data and Applications Security at the US National Science Foundation (NSF). She has been with the MITRE Corporation since January 1989 and is currently on IPA to NSF. She has worked in secure databases for more than 17 years and is the recipient of the IEEE Computer Society's 1997 Technical Achievement Award for outstanding and innovative contributions to secure distributed data management and IEEE's 2003 Fellow Award for contributions to secure systems involving database systems, distributed systems, and the Web. Recently, she received the AAAS 2003 Fellow Award for her research in secure Web information management. She has published more than 400 technical papers and reports, including more than 60 journal articles in secure data management and information technology. She is the inventor of three patents for MITRE on Database Inference Control. She has written six books on data management and data mining for technical managers. Her recent book is on Web data management technologies and their applications to counter-terrorism based on her keynote presentations on the subject at the White House and at the United Nations in 2002. Her current research interests are in secure semantic Web, privacy constraints processing, and secure sensor information management. She is a fellow of the IEEE.

Amar Gupta received the bachelor's degree in electrical engineering and the master's degree in management from the Massachusetts Institute of Technology (MIT), and a doctorate in computer science. He holds the Thomas R. Brown Chair in Management and Technology at the University of Arizona in Tucson. He is also a professor of enterpreneurship and MIS, and the senior director for research and business development at the Eller College of Management at this university. Prior to accepting this endowed position in 2004, he worked in various capacities at MIT for 25 years, including as codirector of the Productivity from Information Technology Initiative for half of this tenure. He has published more than 100 papers in prestigious journals and is currently the associate editor of the *ACM Transactions on Internet Technology*. His areas of research includes knowledge acquisition, knowledge discovery, knowledge management, and knowledge dissemination. He is a senior member of the IEEE and the IEEE Computer Society.