Securing XML Documents with Author-X

This Java-based access-control system supports secure XML document administration at varying levels of granularity.

Elisa Bertino and Silvana Castano
University of Milan

Elena Ferrari
University of Como

The widespread adoption of XML for Web-based information exchange is laying a foundation for flexible granularity in information retrieval. XML can "tag" semantic elements, which can then be directly and independently retrieved through XML query languages. Further, XML can define application-specific document types through the use of document type definitions (DTDs).

Such granularity requires mechanisms to control access at varying levels within documents. In some cases, a single-access control policy may apply to a set of documents; in other cases, different policies may apply to fine-grained portions of the same document. Many other intermediate situations also arise. (For an overview of research and available commercial products, see "The XML Security Page" at http://www.nue.et-inf.uni-siegen.de/~geuer-pollmann/xml_security.html.)

The typical three-tier architecture for accessing an XML document set over the Web consists of a Web client, network servers, and the back-end information system with a suite of data sources. In this framework, shown in Figure 1 (next page), public-key infrastructures (PKIs) represent an important development for addressing security concerns such as user authentication. But such facilities do not provide mechanisms for access control to document contents nor for their release and distribution.

Author-X is a Java-based system, developed at the University of Milan's Department of Information Science, to address the security issues of access control and policy design for XML documents.[1] Author-X supports the specification of policies at varying granularity levels and the specification of user credentials as a way to enforce access control. Access control is available according to both push and pull document distribution policies, and document updates are distributed through a combination of hash functions and digital signature techniques. The Author-X approach to distributed updates allows a user to verify a document's integrity without contacting the document server.

1089-7801/01/$10.00©2001 IEEE

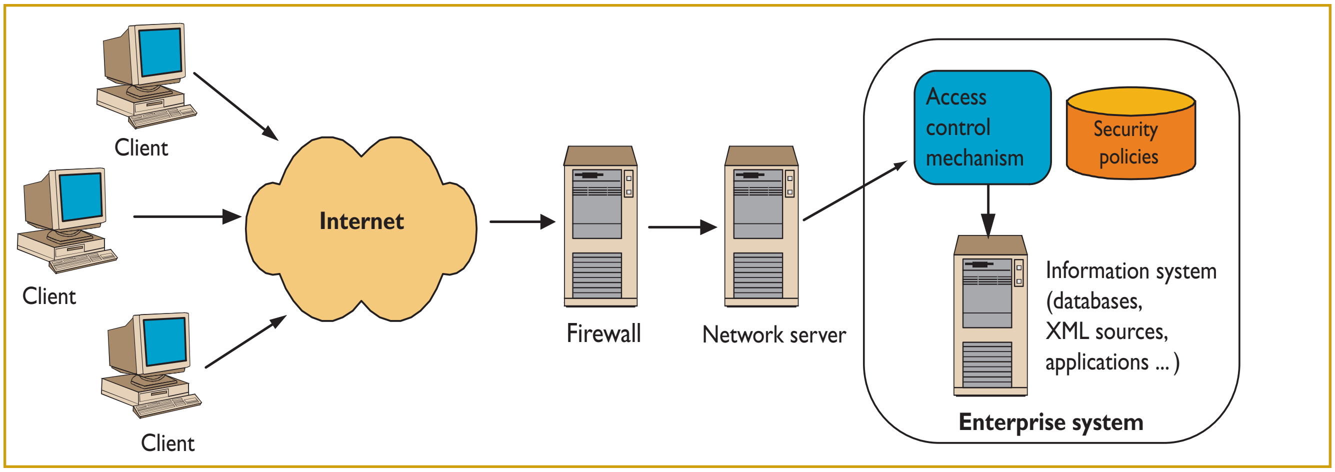


Figure 1. Typical three-tier architecture for access to an XML document source. Public-key infrastructures do not provide mechanisms for access control to document contents nor for their release and distribution.

In this article, we will first illustrate the distinguishing features of credential-based security policies in Author-X, then examine the system's architecture, and conclude with details about its access-control and administration engines.

Credential-Based Security Policies

In general, security policies state who can access enterprise data and under which modalities. Once policies are stated, they are implemented by an access-control mechanism. In Author-X, security policies for XML documents have the following distinguishing features:

- They can be *set-oriented* or *instance-oriented*, reflecting support for both DTD- and document-level protection.
- They can be *positive* or *negative* at different granularity levels, enforcing differentiated protection of XML documents and DTDs.
- They include options for *controlled propagation* of access rights, whereby a policy defined for a document or DTD can be applied to other semantically related documents and DTDs (or portions of them).
- They reflect user profiles through *credential-based qualifications*.

Author-X security policies are implemented through six basic components.

User Credentials

The first component is a user credential — that is, a set of properties about a user that are relevant for security policies (age, position within the organization, current projects, and so on).[2] Credentials allow a system security administrator to express security policies directly. Each user is assigned one or more credentials upon subscribing to the system. To simplify this task, Author-X groups credentials with similar structures into *credential types*. It also provides a language based on XML for encoding credentials and credential types.

Figure 2 shows example credentials for subjects entitled to access an XML purchase order document. Figure 3 represents the document graphically: Blue nodes represent elements, yellow nodes represent XML syntax attributes, and edges represent element-to-subelement and element-to-attribute relationships. Users can be qualified by XPath expressions that set conditions on credentials and credential properties.[3] For example, an XPath expression could be specified to say that employees of carrier company X (CCX) can access customer information in the purchase orders for which CCX is the carrier. Similarly, we could say: All secretaries working in the sales department can access and modify all purchase order documents.

Protection Objects

Author-X protection objects are documents or DTDs (or portions of them). Policies can be specified for a range of protection objects:

- all instances of a given DTD;
- collections of both well-formed (follows XML grammar rules, but does not have an associated DTD) and valid (well-formed and conforms to a given DTD) XML documents; and
- selected portions within one or more documents (such as an element, attribute, or link — or a set of any of these).

The granularity of protection objects is complemented in Author-X by support for content-based access control, which allows the security administrator to specify security policies based on document content (attribute values). This feature is important because documents with the same structure frequently have different protection requirements with respect to their contents.

Access Modes

Author-X categorizes security policies into two groups: *browsing* and *authoring*. Browsing policies allow a user to read the information in a protection object or possibly to navigate through its link, whereas authoring security policies allow a subject to modify protection objects according to different modes (such as append, write, delete, or insert).

Signs

Author-X security policies can specify either permissions (positive policies) or denials (negative policies). By using this feature, the security administrator can easily specify exceptions (granting access permissions to a whole document except for one attribute, for instance) and thus reduce the number of policies that must be specified to secure an *XML source*, that is, a set of XML documents and DTDs.

```
<carrier_employee credID="154">
   <name>
      <fname> Bob </fname>
      <lname> Watson </lname>
   </name>
   <phone_number> 8005769840 </phone_number>
   <company> UPS </company>
</carrier_employee>


<secretary credID="104", managerID="154">
   <name>
      <fname> Tom </fname>
      <lname> Moore </finame>
   </name>
   <age> 25 </age>
   <department> sales < /department>
   <salary> 2,000 </salary>
   <level> third </level>
   <duty> manager secretary </duty>
</secretary>
```

Figure 2.Two example user credentials. XML tags define properties that Author-X uses for document access control.

Author-X uses a *strongest-policy* principle to solve conflicts that arise between positive and negative policies. This conflict-resolution strategy relies on criteria stating that security policies defined for specific documents prevail over those defined for DTDs (because the latter are considered less specific) and that policies defined at a lower level in a document or DTD hierarchy prevail over those defined at higher levels.

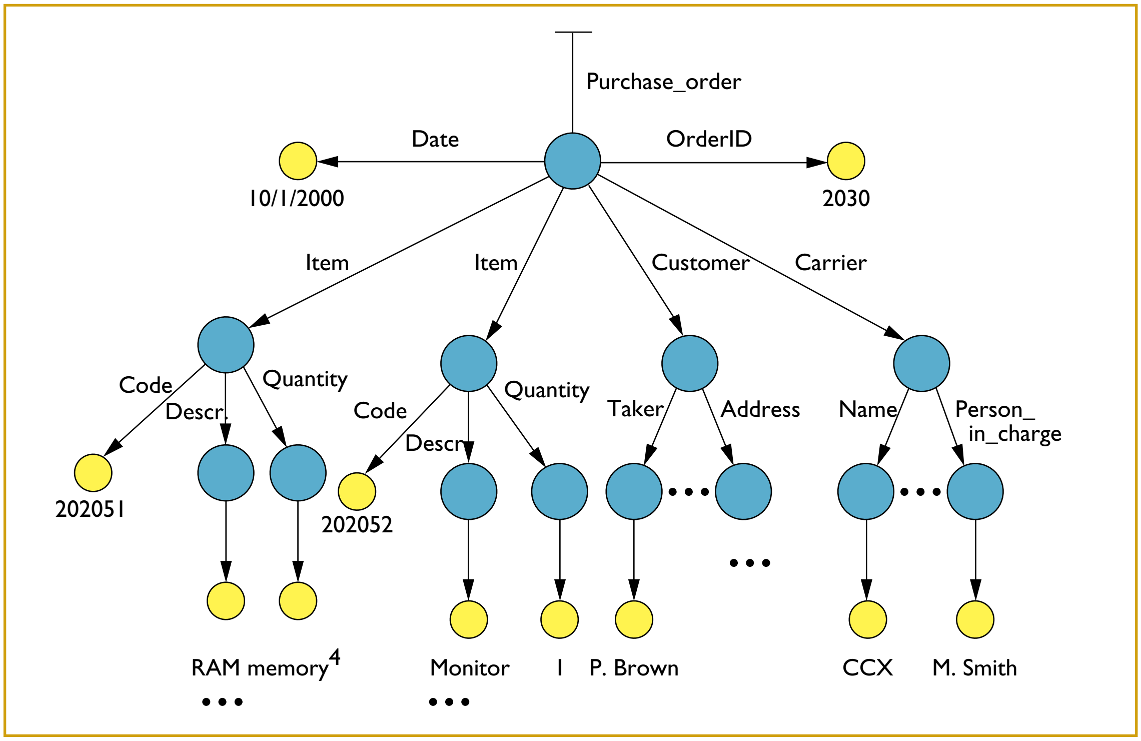


Figure 3. XML document representation. Blue nodes represent elements, yellow nodes represent XML syntax attributes, and edges represent element-to-subelement and element-to-attribute relationships.

```
<policy_base>
   <policy_spec cred_expr="//secretary [department="sales"]" target="Purchase_order.dtd"
     priv="ALL" type="GRANT" prop="CASCADE"/ >
   <policy_spec cred_expr="//carrier_employee[company="CCX"]" target="Purchase_order.xml"
     path = "//Purchase_order[Purchase_order/carrier/name="CCX"]" priv="VIEW" type="GRANT"
     prop="CASCADE"/ >
  <policy_spec cred_expr="//carrier_employee[company="CCX"]" target="Purchase_order.xml"
     path = "//item" priv="VIEW" type="DENY" prop="CASCADE"/ >
   <policy_spec cred_expr="//publicity_agent target="Purchase_order.dtd
      path="//item/description priv="VIEW" type="GRANT" prop="NO_PROP"/ >
   <policy_spec cred_expr="//publicity_agent target="Purchase_order.dtd"
      path="//Purchase_order/order ID" priv="VIEW" type="GRANT" prop="NO_PROP"/ >
</policy_base>
```

Figure 4. Example policy base. Author-X encodes the security policies for a set of XML documents in an XML file called the policy base.This policy base encodes the security policies specified for the document in Figure 3.

Propagation Options

The Author-X access-control model provides a set of propagation options. These options specify how and whether a security policy for a given protection object propagates to another. This feature thus offers a means to concise expression for a set of security requirements.

There are two types of propagation: *implicit* and *explicit*. Implicit propagation is applied by default to all specified policies and is based on the two principles:

- DTD-level policies automatically propagate to all DTD instances, and
- policies specified on a given document or DTD element automatically propagate to all associated attributes and links.

The security administrator can, when necessary, overwrite these default propagation principles by specifying explicit positive and negative security policies. There are three explicit propagation options available in Author-X:

- NO_PROP. The security policy applies only to the protection objects defined in its specification; no propagation is enacted.
- FIRST_LEVEL. The policy propagates to all direct child elements of the elements in the policy specification.
- CASCADE. The policy propagates to all direct and indirect child elements of the elements in the policy specification.

The security administrator can thus state if and how a given policy propagates to protection objects at lower levels in the document or DTD hierarchy.

Policy Base

All security policies for an XML source are encoded in an XML file called the policy base. Figure 4 shows an example policy base for the XML document in Figure 3. The first policy allows the secretaries of the sales department to modify and browse all purchase order documents (enforced through a policy specified at the DTD level). The next two policies give CCX employees access to information about the customer, carrier, order id, and date on purchase orders for which CCX is the carrier:

- a positive policy referring to orders for which CCX is the carrier applies to all purchase order documents, and
- a negative policy applies only to the portions of documents that are unavailable to CCX employees (that is, the item elements).

The last two policies authorize publicity agents to access the order id and description of purchase order items.

System Architecture

Author-X has a security policy engine for system specification, validation, and maintenance; the engine itself is based on XML to enforce access control. Figure 5 shows the system's general architecture, which consists of the XML source and X-bases repositories and two Java components, X-Access and X-Admin.

Repositories

The XML source repository contains the XML documents and DTDs, if defined, which are to be protected.

X-bases repositories are the following:

- The *policy base* contains the security policies for the documents and DTDs in the XML source.
- The *credential base* contains the user credentials and credential types.

- The *encrypted document base* contains an encrypted copy of portions of the documents in the XML source.

Expressing credentials, credential types, and security policies using XML makes them fully interoperable with one another. Using XML to specify credentials also facilitates secure submission and distribution because we can apply the digital signature and encryption mechanisms we have developed to protect XML documents. Moreover, expressing security policies in XML simplifies information exportation when a document migrates from one source to another.

Java Components

X-Access implements access control over the XML source by using security policies and credentials stored in the corresponding X-bases. In response to each access request, X-Access returns a view with only those portions of the requested documents for which the policy base authorizes access. X-Access supports the controlled release of XML documents according to the pull and push dissemination modes.

X-Admin provides an environment and graphical tool set to assist the security administrator in administration activities related to policy and credential management.

Because they constitute the core components of XML document protection, we will discuss these Java components in more detail.

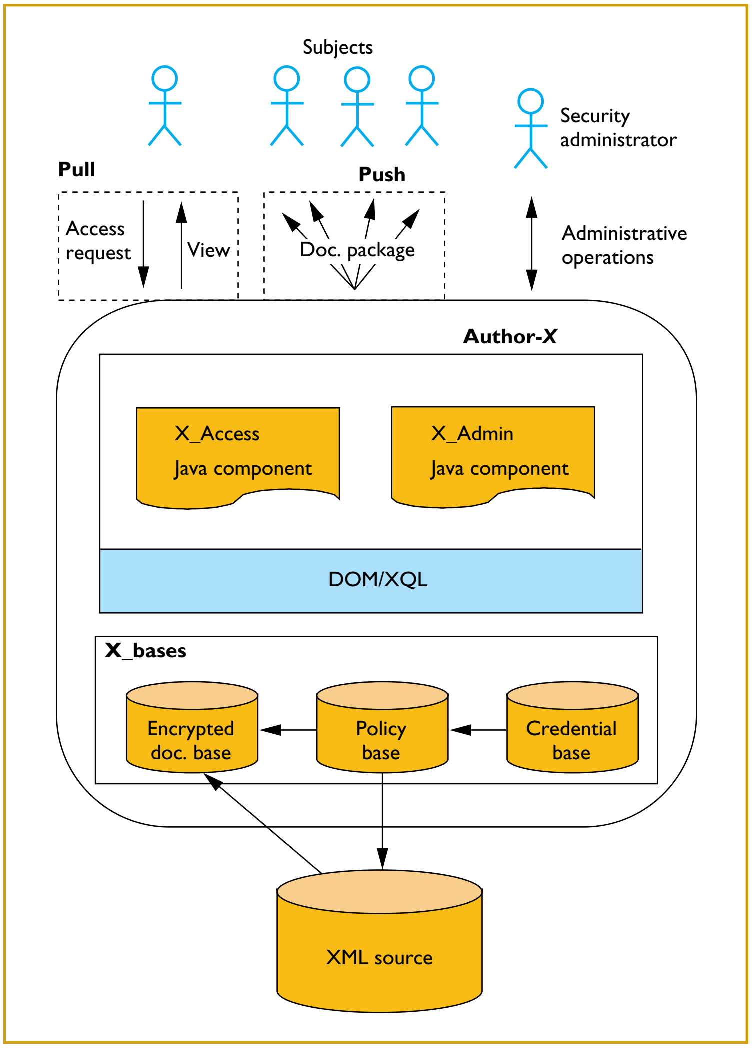


Figure 5. Author-X system architecture. Author-X components are the XML source and X-bases repositories and the X-Access and X-Admin Java modules.

X-Access Dissemination Modes

The pull and push dissemination modes in Author-X are complementary, and the choice of which to use depends on many factors. The security administrator can use document characteristics and user profiles to decide which mode to employ for all documents in the source. *Information pull* mode is commonly used in traditional database management systems; it is suitable for documents that are not distributed as regularly or that are usually requested individually. *Information push* can be useful for sending documents such as monthly newsletters to a set of subscribers; it could also be appropriate for a company that distributes a monthly report containing sections that everyone can access and others (financial information, for example) that only selected users can access.

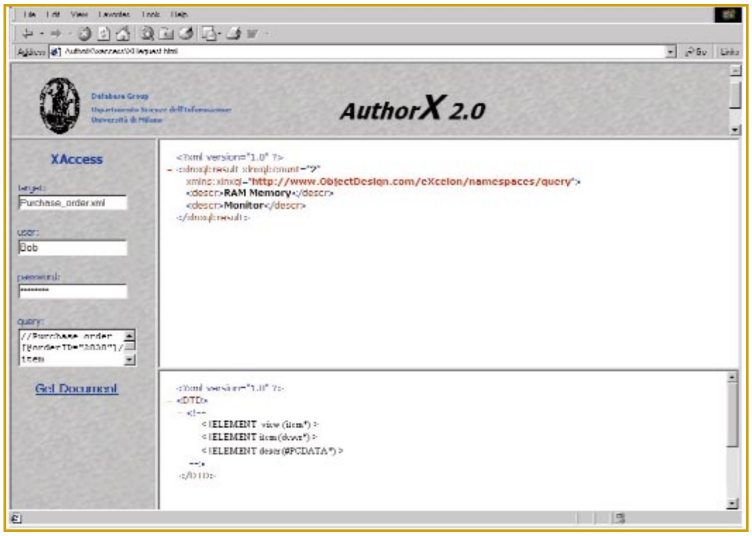
Pull-Mode Operations

Under information pull, a user submits an explicit document access request, which is represented as a tuple:

```
r = <subject, target, path,
    acc_modality>
```

where `subject` is the user requesting access, target corresponds to the requested XML documents, path is an XPath expression that eventually selects specific portions of the requested documents, and acc_modality is an element of the set {browsing, authoring} that specifies which type of access is requested.

Upon receiving an access request , X-Access checks which authorizations (both positive and negative) the `subject` has on the `target` documents — either directly by the security policies or implicitly by propagation options. The system then



Purchase_order.xml

Bob

****

```
//Purchase_order
[@orderID="2030"]/
item
```

```
<?xml version="1.0" ?>
- <dxxql:result xlnxql:count="2"
    xmlns:xlnxql="http://www.ObjectDesign.com/eXcelon/namespaces/query">
    <descr>RAM Memory</descr>
    <descr>Monitor</descr>
  </dxxql:result>
```

```
<?xml version="1.0" ?>
- <DTD>
  - <!--
      <!ELEMENT view (item*)>
      <!ELEMENT item(descr*)>
      <!ELEMENT descr(#PCDATA*)>
    -->
  </DTD>
```

Figure 6. Example access request and returned document view in Author-X. The request appears in the left-hand side of the screen; the returned view, top right; and the associated DTD, bottom right.

presents the `subject` a view of the authorized portions of the requested documents.

The first step in building the subject view is the *pruning phase*, during which the system removes all unauthorized elements and attributes from the requested document or documents. During this phase, X-Access first queries the policy base to extract all browsing or authoring policies (depending on the type of access request) specified for the target documents — including those specified on the associated DTD. Access is denied if the query result is empty. Otherwise, the system uses the conflict-resolution rule to order the policies in decreasing priority.[4]

The pruning algorithm iteratively considers each extracted policy and marks the elements and attributes with a "+" to specify permission or a "–" for denial. All attributes marked with a minus sign or left unmarked are pruned from the `target` view, and the `path` expression is evaluated against this pruned document. If the user requests access to a whole document (that is, the `path` component is null), the system returns the pruned file. The resulting node set is turned back into a well-formed XML document for transfer and display.

For example, suppose that Bob, a publicity agent, submits the following access request:

```
<Bob, Purchase_order.xml,
//Purchase_order[@orderID='2030']/
item,browsing>
```

as shown in the left-hand frame in Figure 6. Following the security policy stored in the policy base, Bob receives a view that contains only the description of each item in the order (that is, RAM and monitor). The right-hand frame in Figure 6 shows the purchase order view returned to Bob (top area) and its associated DTD (bottom area).

Push-Mode Operation

Under information push, the system periodically broadcasts documents to users, rather than sending them upon request. Because different users may have different viewing rights for portions of the same document, support for the push mode could entail generating different views of each document. However, information push is largely designed to distribute documents to large communities, so this approach is not practically scalable.

For this reason, X-Access adopts an approach that essentially allows the security administrator to send the same document to all subjects, while still enforcing stated security policies. Extending Gladney and Lotspiech's Cryptolope approach,[5] our system encrypts portions of a document and uses different encryption keys for different security policies. X-Access selectively distributes the keys necessary to let each authorized user access appropriate portions of the document. We have developed mechanisms to support information push for both browsing and authoring access.

Browsing access. Document encryption is driven by the specified security policies: All the elements and attributes to which a single policy (or combination of policies) applies are encrypted with a single key. The encrypted copy of the document is then stored in the encrypted document base (shown in Figure 4).

To illustrate the document encryption strategy, suppose the purchase order document in Figure 3 should be "pushed" according to the policy base in Figure 4. As shown in Figure 7, encryption generates four keys (K_1, K_2, K_3, K_4) for different portions of the document. According to the security policies, secretaries will receive all the keys, all company employees will receive K_1, and publicity agents will receive K_3 and K_4.

X-Access supports two different distribution methods for delivering the encrypted document and keys to a set of users:

- *online* mode delivers the keys and document together, whereas
- *offline* mode requires users to retrieve the keys through further interactions with the system.

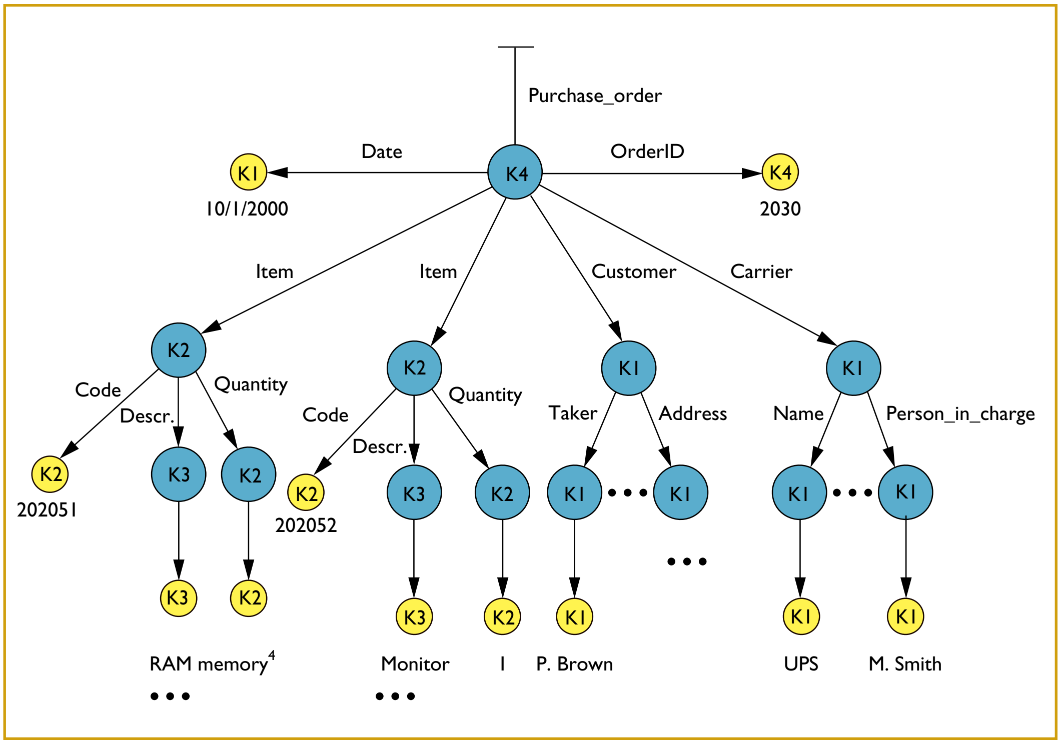


Figure 7. Key generation. Different portions of a document are encrypted with different keys based on the security policies holding for the document. This document has been encrypted with four different keys based on the security policies specified for the document in Figure 3.

There are two different options for online mode. Under the first, the encrypted document is encapsulated into a package that includes an entry with appropriate keys for each recipient. Upon receiving the package, the user decrypts the entry with a private key, and then uses the keys here contained to decrypt the document. This approach lets us send the same package to all subjects and protect the content because each portion can be decrypted only with the corresponding private key.

The drawback is that the package can reach considerable size with a large number of users who each need multiple keys. Moreover, this approach can be vulnerable to attack because all the keys are in the same package. A malicious subject could launch a denial-of-service attack by deleting another user's keys, for example, and making it impossible to decrypt document content. For all these reasons, X-Access also supports an online distribution mode in which keys are sent to each subject in a separate message using secure e-mail techniques.[6]

In offline mode, on the other hand, X-Access sends only the encrypted copy of the document to all subjects; keys are stored at the server site using the Lightweight Directory Access Protocol,[7] and subjects query the LDAP directory to obtain the appropriate keys.

Choosing whether to use online or offline key distribution for a specific document depends on issues such as the number of recipients, the number of keys generated during document encryption, the sensitivity level of the information, and the users' behavior (whether they are always connected to the network, for example, or are seldom-connected mobile users). With so many influential factors, we designed our system to support a spectrum of key distribution methods; security administrators can select the most appropriate method for each document.

Authoring access. We designed the push mode for authoring access to handle XML documents that are subject to predefined distributed and cooperative updates (usually at explicit intervals along a specified update path) during which users in different organizational roles must modify possibly different portions of the same document. A monthly report might first be modified by a secretary, for example, then signed by a manager, and passed along up the chain of command.

Using traditional pull mode, Author-X would have to mediate and manage each update request by returning only the portions of the document that each user was authorized to modify. Moreover, the system would have to verify whether each user's updates obeyed the security policy restrictions for the document. Under information push, on the other hand, the server generates and distributes a single encrypted copy of the document, which flows from one subject to another along an attached update path.

As Figure 8 (next page) shows, each user separately receives decryption keys for appropriate portions of the document. Unlike browsing access,

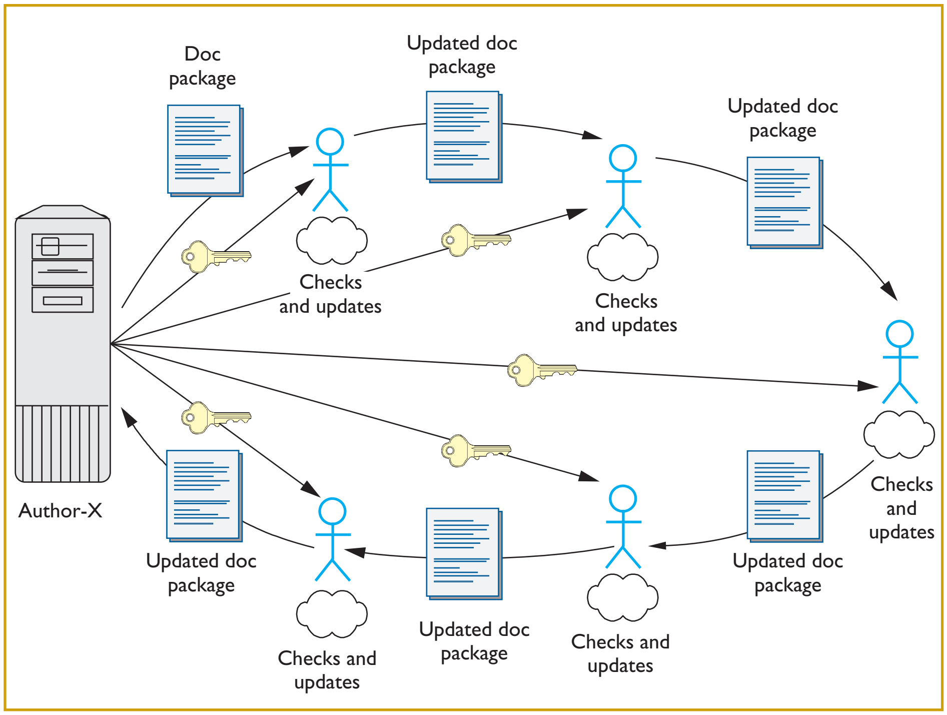


Figure 8. Push mode for authoring access. Encrypted documents with respective keys are sent to all users. Users exploit the keys to decrypt received documents and can modify only the authorized portions.

authoring access also requires a mechanism that lets each user verify whether the source's authoring policies allow the previous users' authoring operations and whether the specified update path was followed. By attaching additional control information to the encrypted document, we create a distributed environment that enables users, in most cases, to perform this verification without interacting with the document server. To generate the so-called *document package* (the encrypted document together with the control information), we use hash functions and digital signature techniques.

X-Admin Facilities

Figure 9 shows the pool of security administration tools in the X-Admin architecture. We have implemented two components — the *policy manager* and the *credential manager* — on top of five basic viewer facilities:

- The *document/DTD viewer* uses a graphical notation similar to the one adopted by conventional XML editing and parsing tools, such as the Apache group's Crimson and Xerces, to display a target document or DTD.
- The *policy viewer* displays the users and XML documents related to a given policy.
- The *propagation viewer* uses both explicit and implicit propagation principles to display all policies for the target document or DTD. In particular, it adopts different graphical styles to represent protection objects by policy type (explicit or propagated, positive or negative, and so on).
- The *conflict viewer* displays all conflicts arising among security policies defined for the target document or DTD. Moreover, the viewer shows the default conflict-resolution choice for each conflict, determined according to the strongest-policy principle.
- The *credential viewer* displays the structure of credential types and user credentials; it also provides a form-based graphical editing environment for specifying credential expressions.

The security administrator can invoke these basic facilities interactively within an X-Admin working session. Propagation and conflict viewer facilities can also be invoked on the whole policy base to visualize propagation and conflicts for all stored security policies on a per-document or DTD basis. Because documents and security-related information are specified in XML syntax, viewer facilities work internally on document object model (DOM) representations[8] and exploit the XML query language (XQL), as shown in Figure 9.

Credential Manager

The credential manager component supports the security administrator in the specification and maintenance of credential types and associated user credentials. To define a new credential type, the security administrator specifies its name and structure in forms provided through the credential viewer. Once the credential-type structure has been defined, the credential manager generates the corresponding XML-based syntax for storage in the credential base.

It is also possible to add new properties to existing credential types through the credential viewer. Deleting a credential type also removes all user credentials defined for it. Interactive menus list credential types, properties, constant values, and comparison and logic operators for graphically defining expressions on credential types. The credential viewer-editor then generates the corresponding XPath-based syntax for the credential expressions.

Policy Manager

The security administrator can specify security policies for a target document or DTD through the policy manager's specification forms. To limit the input required for securing a target document or DTD, the policy manager derives as much information as possible from the policy base and from the documents and DTDs in the XML source, through implicit and explicit propagations. The policy manager proposes any automatically derived candidate policies to the security administrator for validation, at which point the policies can be customized as necessary or accepted "as-is." The policy manager also supports maintenance operations for updating or revoking policies in the policy base.

Figure 10 shows the X-Admin policy specification menu. In a working session, the security administrator first selects the policy specification modality — *in-site* or *external* — for the document to be protected.

Specifying in-site-document policies. Under in-site mode, the security administrator specifies explicit security policies for protection objects for a target document stored in the XML source (in-site-document). Security policies can be specified either at the DTD level or at the document level. Author-X supplies a form with all the fields for defining a security policy: `subject`, `protection object`, `access mode`, `sign`, and `propagation option`.

The policy manager automatically fills in the form's protection object field with the XPath expression for a protection object selected within the graphical representation of the target document. To specify the users to whom a security policy applies, the security administrator invokes the credential viewer editor. Each of the other fields has an associated multiple-choice menu that lists admissible values and predefined default values (`read`, `grant`, and `cascade`), which can be modified as necessary. As policy definition proceeds, the security administrator can also use the propagation viewer to interactively analyze security policies. Once the security requirements have been satisfied for the target document, the policy manager generates the XML syntax for storing the validated policy in the policy base.

Specifying external document policies. Under external mode, the security administrator can use the policy manager to specify security policies for new documents coming from outside. If the external document is valid, the system displays the associated DTD graphically, and the security administrator can specify the DTD-level security

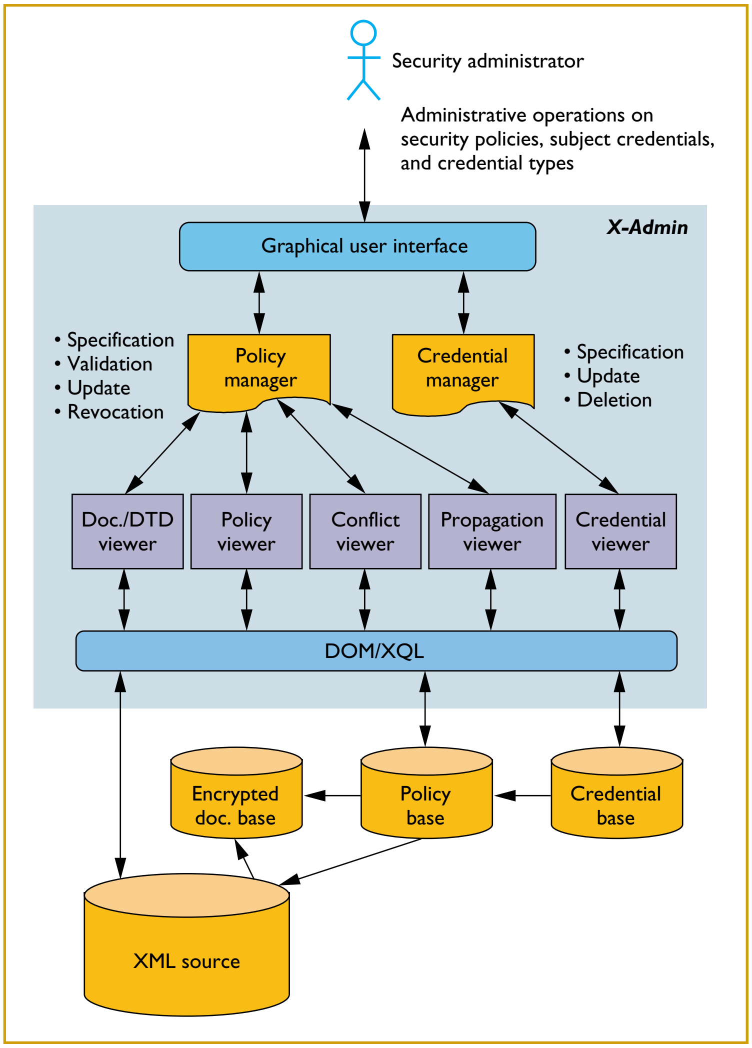


Figure 9. X-Admin architecture. A pool of security administration tools is based on two components — the policy manager and the credential manager — on top of five basic viewer facilities.

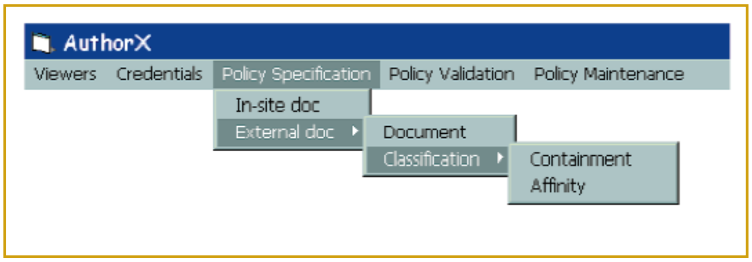


Figure 10. X-Admin policy specification menu. Different options can be invoked for securing documents stored in the XML source and those coming from outside.

policies in the same way as for in-site documents. If the external document is well formed, the policy manager offers both a classification-based and a document-based mechanism that help automate the policy specification process.

The *classification-based* option exploits the propagation principle to derive candidate policies from a DTD that is already secured in the XML source. The policy manager compares the external document against all secured DTDs to find the one with the greatest number of matching protection objects. The classification procedure implements an algorithm that analyzes graph-based structures to compare the structures of all secured DTDs against the document. This procedure identifies a DTD D as a full or partial match depending on the number of its elements that correspond to the external document. If *D* is a complete match — all the elements specified in the external document are present in *D*, and the document in turn contains every mandatory element specified in *D* — the policy manager propagates all security policies defined for *D* to the external document.

If *D* provides a partial match, the policy manager enforces two different mechanisms for propagating policies to a nonmatching protection object of the external document.

We plan to extend protection toward secure access to Web pages and compliance with XML schemas.

- With the *containment-based* mechanism, the policy manager checks whether the protection object is a subelement of an element for which a policy is defined in *D*. If so, the policy for this element is propagated to the protection object.
- With the *affinity-based* mechanism, the policy manager checks *D* to see whether it contains an element or attribute that is semantically related to the protection object (based on a domain ontology and the semantic meaning of the labels).[9,10] If so, the element or attribute is exploited to derive a policy for the protection object.

Author-X then presents policies propagated according to these two mechanisms to the security administrator for validation. Thus, security policies need only be created from scratch for nonmatching protection objects for which no propagated policy can be derived.

If the classification procedure finds no matching DTDs, or upon the security administrator's explicit request, the policy manager invokes the *document-based* option to begin a policy specification session for the external document. In this mode, the security administrator uses the policy manager forms to define security policies from scratch for protecting the external document.

Once the policy specification process is complete, the system checks which secured documents to distribute under information push. Author-X then executes an algorithm that modifies any encrypted copies of these documents stored in the encrypted document base to bring them in line with the newly specified policy. This algorithm incrementally maintains document encryption and changes only the portions to which the specified policy applies. If no encrypted copy of the document exists, the algorithm generates one according to the strategy described earlier under "Pull-Mode Operations," and stores it in the encrypted document base.

Conclusion

The Web community generally regards XML as the most important standardization tool for information exchange and interoperability, and we believe that XML access control will constitute the core security mechanism of Web-based enterprise architectures. The current Author-X prototype is built on top of the eXcelon XML server[11] and supports browsing and updating of DTD-based XML sources.

We plan to extend protection toward secure access to Web pages and compliance with XML schemas (http://www.w3.org/XML/Schema/). Additionally, we will experiment with incorporating Author-X within Web-based enterprise information system architectures by focusing on performance issues. In particular, we will study XML-based solutions for certifying user credentials as well as access-control schemes and architectures for securely disseminating information. We have proposed a preliminary set of XML-based access-control schemes for distributed architectures,[12] and we are working on a prototype extending the Author-X functionalities accordingly.

Acknowledgments

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

References

1. E. Bertino, S. Castano, and E. Ferrari, "Securing XML Documents: The Author-X Project Demonstration," *Proc. ACM Int'l Conf. Management of Data* (SIGMOD 2001), ACM Press, New York, 2000.

2. M. Winslett et al., “Using Digital Credentials on the World Wide Web,” *J. Computer Security*, vol. 7, 1997.
3. *XML Path Language (XPath), Recommendation 1.0*, World Wide Web Consortium (W3C), 1999; available at http://www.w3.org/TR/xpath.
4. E. Bertino et al., “Specifying and Enforcing Access Control Policies for XML Document Sources,” *World Wide Web*, Baltzer Science Publishers, Netherlands, vol. 3, no. 3, 2000; available online from http://www.baltzer.nl/www/contents/2000/3-3.html.
5. H. Gladney and J. Lotspiech, “Safeguarding Digital Library Contents and Users: Assuring Convenient Security and Data Quality,” *D-Lib Magazine*, Corp. for National Research Initiatives, CITY, Va., May 1997; available at http://www.dlib.org/dlib/may97/ibm/05gladney.html.
6. W. Stallings, *Network Security Essentials: Applications and Standards*, Prentice Hall, Upper Saddle River, N. J., 2000.
7. D. Srivastava, “Directories: Managing Data for Networked Applications,” tutorial, *Proc. 16th IEEE Int’l Conf. Database Engineering* (ICDE 2000), IEEE CS Press, Los Alamitos, Calif., 2000.
8. *Document Object Model* (DOM) Level 1, W3C Recommendation; available at http://www.w3.org/TR/REC-DOM-Level-1/.
9. S. Castano and V. DeAntonellis, “A Discovery-Based Approach to Database Ontology Design,” *Distributed and Parallel Databases*, Kluwer Academic Publishers, Netherlands, vol. 7, no.1, Jan. 1999, pp.67-98.
10. A. Miller, “Wordnet: A Lexical Database for English.” *Comm. ACM*, vol. 38, no. 11, Nov. 1995, pp. 39-41.
11. “An XML Data Server for Building Enterprise Web Applications,” white paper, Object Design Inc., 1998; available at http://www.odi.com/whitepapers/XMLEnterprisewWebApps.pdf.
12. E. Bertino, S. Castano, and E. Ferrari, “On Specifying Security Policies for Web Documents with an XML-Based Language,” to appear in *Proc. Symp. Access Control Models and Technologies 2001*, ACM Press, New York, 2001.

Elisa Bertino is a professor of computer science and the chair of the Department of Computer Science at the University of Milan. Her research interests include security, object-oriented databases, distributed databases, multimedia databases, interoperability of heterogeneous systems, and integration of artificial intelligence. She recently co-authored *Intelligent Database Systems* (Addison Wesley). She also served as program chair of the 14th European Conference on Object-Oriented Programming (ECOOP 2000).

Silvana Castano is a professor of computer science at the University of Milan. Her research interests include database and XML security, intelligent information integration and ontologies, Web-based information systems, process reengineering, and workflows. She received a PhD in computer and automation engineering from Politecnico di Milano, in 1993. She co-authored *Database Security* (Addison-Wesley, 1995). She is currently chair of the Associazione Italiana per l’Informatica ed il Calcolo Automatico (AICA) working group on databases.

Elena Ferrari is a professor of computer science at the University of Como, Italy. Her research interests include security, multimedia databases, and temporal object-oriented data models. She received a PhD in computer science from the University of Milano in 1997. She served as program chair of the first ECOOP Workshop on XML and Object Technology (XOT00).

Readers may contact the authors via e-mail at {bertino, castano,ferrarie}@dsi.unimi.it.