IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB9852

Chapter VI

Secure Data Dissemination

Elisa Bertino, Università degli Studi di Milano, Italy

Barbara Carminati, Università degli Studi di Milano, Italy

Elena Ferrari, Università degli Studi dell'Insubria, Italy

ABSTRACT

In this chapter, we present the main security issues related to the selective dissemination of information (SDI system). More precisely, after provided an overview of the work carried out in this field, we have focused on the security properties that a secure SDI system (SSDI system) must satisfy and on some of the strategies and mechanisms that can be used to ensure them. Indeed, since XML is the today emerging standard for data exchange over the Web, we have casted our attention on Secure and Selective XML data dissemination (SSXD). As a result, we have presented a SSXD system providing a comprehensive solution to XML documents. In the proposed chapter, we also consider innovative architecture for the data dissemination, by suggesting a SSXD system exploiting the third-party architecture, since this architecture is receiving growing attention as a new paradigm for data dissemination over the web. 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 user queries. A relevant issue in this architecture is how the Owner can ensure a secure dissemination of its data, even if the data are managed by a third-party. Such scenario requires a redefinition of dissemination mechanisms *developed for the traditional SSXD system. Indeed, the traditional techniques cannot be exploited in a third party scenario. For instance, let us consider the traditional digital signature techniques, used to ensure data integrity and authenticity. In a third party scenario, that is, a scenario where a third party may prune some of the nodes of the original document based on user queries, the traditional digital signature is not applicable, since its correctness is based on the requirement that the signing and verification process are performed on exactly the same bits.*

INTRODUCTION

Companies and organizations are today massively using Internet as the main information distribution means both at internal and external levels. Such a widespread use of the web has sped up the development of a new class of information-centred applications focused on the selective dissemination of information (hereafter called SDI). The obvious purpose of these applications is the delivery of data to a possible large user community. The term selective in this context means that each user should not receive all the data but he/she must receive only specific portions of them. Such portions can be determined according to several factors, such as user interests and needs, or the access control policies that the data source has in place. The chapter focuses on security issues for selective data dissemination services, since such issues represent one of the most novel and promising research directions in the field. A *Secure and Selective Dissemination of Information – SSDI* service– is an SDI service that ensures a set of security properties to the data it manages. In particular, we focus on four of the most important security properties: *authenticity*, *integrity*, *confidentiality*, and *completeness*. Since it is often the case that data managed by an SDI service are highly strategic and sensitive, the scope of SSDI applications is wide and heterogeneous. For instance, a first relevant scenario for such kinds of applications is related to the electronic commerce of information. This is, for instance, the case of digital libraries or electronic news (e.g., stock price, sport news, etc.). In such a case, users subscribe to a source and they can access information on the basis of the fee they have paid. Thus, in a digital library scenario, it is necessary to develop a mechanism ensuring that a user receives all and only those portions of the library he/she is entitled to access, according to the fee he/she has paid and only for the subscription period. Additionally, the service must ensure that these contents are not eavesdropped during their transmission from the library to the intended receiver. Another important scenario for SSDI applications is data dissemination within an organization or community, where the delivery is controlled by security rules defined by Security Administrator(s) (SAs). Consider, for instance, documents containing sensitive information about industrial projects. In such a case, personal data of the enrolled staff should be available only to the authorized secretaries, whereas technical details should be accessible only to the technical staff.

Ensuring security properties is particularly crucial when a *push* dissemination mode is adopted for distributing documents to users. Indeed, under a push dissemination mode, the SSDI service periodically (or whenever some relevant event arises) broadcasts data to the whole (or to selected portions) of the user community. The traditional techniques for data access used in conventional DBMSs are not suitable to enforce information push, since they are based on the conventional on-user demand paradigm (referred also as *information pull*). In fact, since different users may have privileges to see different, selected portions of the same document, supporting the push dissemination mode with traditional techniques may entail generating different physical views of the same document and sending them to the proper users. The number of such views may become rather large and thus such an approach cannot be practically applied. Thus, in the chapter we illustrate some innovative solutions for efficiently supporting information push, based on the use of cryptographic techniques.

In explaining such techniques, and the architectures on which SSDI services are based, we cast our discussion in the framework of XML documents (Bray, 1998). The reason is that XML represents today a standard for data exchange over the Web. However, the approaches we present are general enough to be applied to web documents expressed according to other languages as well. More precisely, in the second part of the chapter we present few innovative solutions to efficiently implement secure and selective XML data dissemination services. We also discuss techniques that can be used to improve scalability for SSXD services, by focusing on the use of third-party architectures.

The remainder of the chapter is organized as follows. First we provide an overview of the works carried out in SDI field. Then we deal with the security issues related the SDI, and then we focus our attention to XML data, introducing SSXD services exploiting the access control paradigm, and the subscription paradigm. Finally, we propose a new architecture for SSXD service, which improves the scalability.

BACKGROUND: APPROACHES FOR SDI SERVICES

Since the concept of SDI was introduced by H.P. Luhn (Luhn, 1958; Luhn, 1961), different approaches for implementing SDI services have been proposed. Moreover, due to the increasing improvements of hardware, communication bandwidth and ubiquity capability, the evolution and proliferation of SDI systems have been amazing in these last years, thus making the task of reviewing the related work difficult. Just to have an idea of the SDI evolution, at the beginning, that is, around the mid-1970s, SDI systems were implemented only for university libraries [see (Housman, 1973) for an overview], with the aim to deliver updates of bibliographic information on technical journals to users. In these systems, managing only textual data, user interests were modeled by Boolean expressions, which state the keywords and the conditions that documents should have or satisfy in order to be interesting for the user. By contrast, today SDI systems are able to manage several data types: structured, unstructured, semi-structured (Belkin, 1987; Salton, 1983), and multimedia data (Alert, 2003). Furthermore, the use of the Web as means for data exchange has sped up the evolution of SDI systems. Typical and widespread examples of SDI systems exploiting the Web are the personal portal pages, that is, the possibility that nowadays the major portal sites (i.e., like Excite, Yahoo, etc.) allow users to customize their own portal pages by simply selecting topics (financial news, horoscope, society news, etc.) they are interested in to launch when their browsers open the portal site web pages.

Thus, instead of describing the main features of some commercial or academic SDI systems, we prefer to give in this section an overview of the most important issues related to SDI. We start by recalling that the common goal of all SDI systems is to collect new data items from several data sources, filter them according to some criteria (i.e., user profile, subscription, or access control policies) and then deliver them to interested users. This means that a key issue is that the SDI service is be able to effectively and efficiently filter the data to deliver on the basis of user profile[1] — that is, to send all and only the information interesting for user, avoiding the delivery of unrequested data or the loss of information. Thus, in such a context, there exist at least two main issues to be faced according to which SDI systems can be classified. The first issue is the representation of dynamic user profile, whereas the second is the filtering of the information according to user profiles. Indeed, the two problems are closely related in that the user profiles representation greatly influences the filtering algorithm. In general, we can say that an SDI system converts the user profiles in queries expressed in a language understood by the system, so that the matching of them on the documents gives the information to deliver to the users. Thus, in the following we mainly focus on the first issue.

In particular, the research groups that have mostly investigated these issues are the Information Filtering (Loeb, 1972) and the Information Retrieval communities (Salton, 1983). Indeed, the main goal of these groups is to develop systems that select among large collections of information all and only the ones that are of interest for a requesting user. The results of the efforts carried out by these research communities are three different models, namely, the Boolean model, the Vector Space model, and the Probabilistic model, which represent the retrieval model most widely adopted by SDI systems.

The Boolean model represents user profile by means of set of words, and Boolean predicates applied to them. These predicates are matched on the documents to determine the information to deliver to users. Several SDI services exploiting the Boolean model exist, such as, for instance, the IBM's Gryphon (Strom, 1998), and SIENA (Carzaniga, 2000). These SDI systems, basically, differ on the basis of the index structure adopted to implement efficient and effective information filtering [see (Yan, 1994a) for an overview of the index structures suitable for the Boolean model].

The major problem with the Boolean model is that it does not provide the possibility of ranking documents by their relevance for the submitted query. Indeed, according to this model all the documents satisfying the boolean expression specified in the query are delivered to the users, without any other kind of refining. Best-match retrieval models have been devised to cope with this limitation. In particular, the Vector Space model (Salton, 1983; Salton, 1989; Yan, 1999), which is widely known, represents the texts and the queries (i.e., the user profile) as weighted vectors in a multidimensional space. Then, the information filtering is based on a comparison between the text vector and the query vector. The closer the two vectors the more relevant will be the text query. The assumption, which is the base of the Vector Space model, is that the more similar the vector representing a text is to a vector representing the user description, the more likely is that the text is relevant to that user. Similarly to the Boolean model, the SDI systems exploiting a Vector Space model may differ on the adopted index structure [for an overview of the index structures for SDI adopting a Vector Space model see (Yan, 1994b)].

Finally, the probabilistic model is based on the Probability Ranking Principle (Robertson, 1977), which later became known as the binary independence retrieval (BIR). The fundamental idea of the probabilistic model is based on the concept of idea answer set. This set represents the set of documents consisting of all and only those documents that are relevant for a predefined user query. Thus, given a user query if the description of the ideal answer set corresponding to is known, we would not have problems in retrieving its document. According to this basic concept, the main issue in the probabilistic model is how to describe the ideal answer set, which implies associating a set of properties to it. In order to devise these properties, the probabilistic model implies a first guessing phase, where a preliminary probabilistic description of the ideal answer set is provided. According to this description is retrieved an initial set of documents. In order to improve the probabilistic description of the ideal answer set, the probabilistic model implies iterations with the user, with the goal of refining the initial set of retrieved documents. Thus, the probabilistic model implies the reiterating of the iteration phase, in order to make the description much closer as possible to the ideal answer set.

It is important to note the nature of the data being filtered strongly affects the filtering model. Thus, since the IR and IF communities are mainly related to textual data, the database community has investigated the data filtering algorithms for structured data. More precisely, the most significant research efforts have been done in the area of Continual Queries. A continual query is a standing query that monitors updates of the source, thus to return the interested information to users as soon as it is acquired by the source. Early work on CQ for relational databases was done by Terry et al. (Terry, 1992). More recently, OpenCQ (Liu, 1999) and NiagaraCQ (Chen, 2000) have been proposed for information delivery over the Internet.

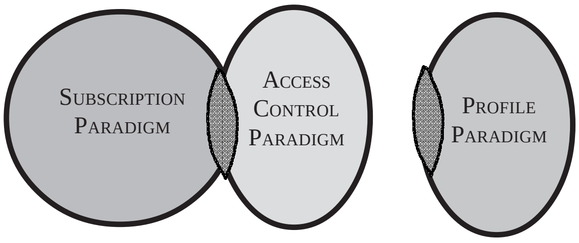
Moreover, due the rapid diffusion of XML as a standard for data exchange over Internet, in the last couple of years several researchers have investigated efficient filtering strategies for XML documents on the basis of user preferences (Altinel, 2000; Diao, 2003; Pereira, 2001). In the proposed approaches, user preferences are specified using some XML pattern specification language (e.g., XPath, 1999). The goals of these research activities are to define algorithms and data structures to implement an effective, efficient and scalable filtering of XML data.

SECURITY ISSUES IN DATA DISSEMINATION SERVICES

In the previous section we have summarized the main research efforts carried out in the area of information dissemination, and we have given an overview of some of the proposed approaches. From this overview it easy to note that security issues have not been so far widely investigated. However, today companies and organizations are massively adopting the information dissemination paradigm for theirs businesses, and this makes security issues very relevant and highly strategic. The aim of this section is thus to introduce some security properties that a secure and selective dissemination of information system must ensure, by sketching some of the possible solutions, which will be carefully presented in the rest of this chapter. However, since these security properties and the mechanisms to ensure them are strictly correlated to the data dissemination paradigms adopted by the SSDI system, we need first to introduce them. Indeed, in designing an SSDI system, it is necessary to take into consideration that there does not exist a unique paradigm to filter the data before its delivery. Indeed, it is possible to devise at least three different paradigms, namely the access control paradigm, the profile-based paradigm, and the subscription-based paradigm, which are briefly introduced in the following.

The access control paradigm implies the existence of an access control model whereby a SA specifies authorization rules stating which portions of the source can be accessible by which users. This is the case, for instance, of a source containing confidential data (e.g., high strategic industrial information), which must be accessible in a selective manner. In this case, the selective delivery of data is regulated by the specified access control policies. This means that each user receives only the data portions for which he/she has an authorization according to the specified policies.

Figure 1: A taxonomy of data filtering paradigms



A different approach for data filtering exploits the profile-based paradigm. According to such a paradigm, the service releases data to users on the basis of their profiles (i.e., user interests, and needs). More precisely, a user profile contains information that could be directly specified by the users (i.e., during an subscription phase), or could be collected indirectly by the SSDI service (for instance, by analysing previous data requests made by the same user), and which are used by the SSDI system to better customize the service for their users. Unlike the access control paradigm that is mainly driven by security issues, in a profile-based scenario the main goal of data filtering is to avoid the delivering of superfluous information to users.

The last data filtering paradigm is the subscription-based paradigm, according to which users have to register to the data source by a mandatory subscription phase. More precisely, during this subscription, a user specifies directly to the SSDI service the portions of the source that he/she is interested in receiving. In this case, the selective delivery of data is regulated by this selection, in that users will receive all and only the subscribed portions. It is interesting to note that in this scenario the user profiles are not taken into account, in that the data filtering mechanism does not care about the user interests and needs, but it simply sends to a user the subscribed portions. Due this reason, this paradigm is particularly tailored for pay-per-view services, which are characterized by the fact that SDI services release to users all and only those portions for which the users have paid a subscription fee.

As reported in *Figure 1*, the three paradigms are not exclusive. This means that there may exist SDI services exploiting, for instance, both the subscription and the access control paradigm. Starting from the left, the first circle represents SSDI systems exploiting the subscription paradigm, the second circle represents those systems exploiting the access control paradigm, whereas the last circle represents the SSDI systems adopting the profile paradigm. As depicted in *Figure 1*, an SSDI service could exploit both the access control and the subscription paradigms, as well as both the access control and the profile paradigms. The decision to adopt the combination of two paradigms depends on the scenario in which the service operates. Consider for instance a newspaper company exploiting an SSDI system to securely disseminate articles and news. In this scenario, it is possible to adopt both the subscription and the profile-based paradigm. However, the newspaper company may want to ensure additional filtering criteria, for instance to avoid sending articles containing offensive content to underage people. In such a case, the SSDI system should adopt also the access control paradigm in order to satisfy these additional secure requirements.

We are now ready to start our discussion by focusing on three of the most important security properties: authenticity, integrity, and confidentiality.

Ensuring document authenticity means that the user receiving a document is assured that the document contents come from the source they claim to be from. Ensuring document integrity means ensuring that the contents of the documents are not altered during their transmission from the source to the intended recipient. Finally, ensuring document confidentiality means that the document contents can only be disclosed to users, authorized according to the specified access control rules stated on the source.

In order to ensure these properties it is necessary to integrate the SDI architecture with an additional mechanism whose goal is to securely disseminate the information. The mechanism must take into consideration the delivery mode adopted by the service. There are two main data delivery modes that an SDI service can adopt: pull mode and push mode. According to the information pull mode, when a user submits an access request, the SSDI mechanism checks which authorizations the user has on the requested document. Based on the information contained in the profile, the user may be returned a *view* of the requested document that contains all and only those portions that match the profile. Confidentiality requirements are thus guaranteed since the profile also contains information on the access control policies satisfied by the user. To ensure also the answer integrity and the authenticity, the SSDI mechanism can adopt standard cryptographic techniques (i.e., digital signature, message authentication code, symmetric or asymmetric encryption, etc.). Besides the traditional information pull mode, a push mode also can be adopted. According to the push mode the SSDI system periodically, or when some pre-defined events happen (i.e., an update on the source), sends (portions of) its documents to interested users, without the need of an explicit access request by the users. Supporting push information in SSDI system is much more complicated, because standard techniques used for the pull mode are not efficient when applied to the push mode. In fact, since different users may be returned different, selected portions of the same document, supporting a push dissemination approach may entail generating different physical views of the same document and sending them to the proper users. The number of such views may become rather large and, thus, such an approach cannot be practically applied.

To avoid this situation a solution is that of exploiting broadcast encryption techniques (Fiat, 1994) to efficiently support information push. Broadcast encryption implies the encryption of different portions of the same document with different encryption keys based on the specified access rules. Thus, the same encrypted copy of the document is then broadcasted to all users, whereas each user only receives the key(s) of the portion(s) he/she is enabled to access.

It is important to note that the definition of the mechanisms ensuring the above introduced security requirements depends also by the architecture adopted by the SSDI system. Indeed, the mechanisms become more cumbersome if a third-party architecture is adopted. In fact, the basic idea of third-party architectures is the distinction between the Owner of the information, and a third-party entity managing this information. Such a scenario requires a redefinition of dissemination mechanisms developed for the traditional SDI system. Indeed, the traditional techniques cannot be exploited in a third-party scenario. For instance, let us consider the traditional digital signature techniques, used to ensure data integrity and authenticity. In a third-party scenario, that is, a scenario where a third-party may prune some of the nodes of the original document based on user queries, the traditional digital signature is not applicable, since its correctness is based on the requirement that the signing and verification process are performed on exactly the same bits.

Moreover, in a third-party scenario, in addition to the traditional security requirements (i.e., integrity, authenticity, and confidentiality), a further important requirement is the completeness of the answer. Ensuring completeness implies that a user receiving an answer to an access request must be able to verify that he/she receives all the document(s) (or portion(s) of document(s)) that he/she is entitled to access, according to the stated access control policies and the submitted request. To cope with these security requirements, later in the chapter we propose an approach ensuring the completeness and the authenticity of the answer in a third-party architectures.

A COMPREHENSIVE FRAMEWORK FOR AN SSXD SYSTEM

In this section, we present the overall architecture of the proposed SSXD, whereas the descriptions of its internal architecture are presented in later sections. We cast our discussion in the framework of XML documents. For this, reason, before presenting the details of our architecture, we need to briefly introduce the basic notions of XML.

Basic Concepts of XML

The eXtensible Markup Language (XML) (Bray, 1998) is currently the most relevant standardization effort in the area of document representation through markup languages and it is rapidly becoming a standard for data representation and exchange over the Web. The motivation pushing the large development

Figure 2: An example of XML document

```
<?xml version="1.0" encoding="UTF-8" ?>
<Newspaper Title="MyNewspaper" Date="....">
     <Frontpage>
        <Leading_Article Author="…"Title="…">
           ......
        </Leading_Article>
        <Paragraphs>
           <Paragraph Author="…" Title="...">
              ......
           <Paragraph>
           ......
        </Paragraphs>
     </Frontpage>
     <Politic_page>
        <Politic topic="USA" Author="…" Title="...">
           .......
        </Politic>
        <Politic topic="EUROPE" Author="…" Title="…">
           .......
        </Politic>
        .......
     </Politic_page>
     <Literary_page>
        <Article topic="Books" Author=‟..." Title=‟...">
          .......
        </Article>
        <Article topic="Movies" Author="..." Title="...">
         .......
        </Article>
        .......
     </Literary_page >
     <Sport_page>
        <News topic="Soccer" Author="…" Title="...">
           .......
        </News>
        <News topic="Basket" Author="..." Title="...">
           .......
        </News>
        .......
     </Sport_page>
 </Newspaper>
```

efforts concerning XML is the need to introduce also for Web documents the usual separation between the *structure* and *contents* of documents and their *presentation*, and to describe the semantics content of the various document portions. By separating document structure and contents from presentation, the same source document can be visualized according to different modes. Therefore, a document coded according to XML can be displayed at various devices, even at devices not foreseen at document preparation time. Therefore, even though XML has been initially developed for the Web, it can be used in any application or environment where one needs to organize and describe data, independently from the storage formats and transmission means.

The basic building blocks of an XML document are tagged elements. Elements can be nested at any depth and can contain other elements (*subelements*) in turn originating a hierarchical structure. An element contains a portion of the document delimited by two *tags*: the *start tag*, at the beginning of the element, with the form <tag-name>, and the *end tag*, at the end of the element, with the form </tag-name>, where *tag-name* indicates the type of the element (*markup*). Additionally, an element can contain attributes of different types allowing one to specify element identifiers, additional information about the element, or links to other elements of the document (attribute of type IDREF(s)/URI(s)). An important characteristic of XML is the possibility of attaching to a document an intentional description of its structure, i.e., the *Document Type Definition* (DTD) or an *XMLschema*.

An example of XML document, modeling a newspaper, is given in *Figure 2*. In particular, the newspaper consists of a Frontpage (modeled through the Frontpage element) containing a leading article (the Leading_Artiche element) and one or more additional short articles (the Paragraph elements). As traditional paper-based newspaper, the newspaper contains also additional pages related to several topics. Each page is modeled by a different XML element (i.e., Literary_page, Sport_page and Politic_page elements).

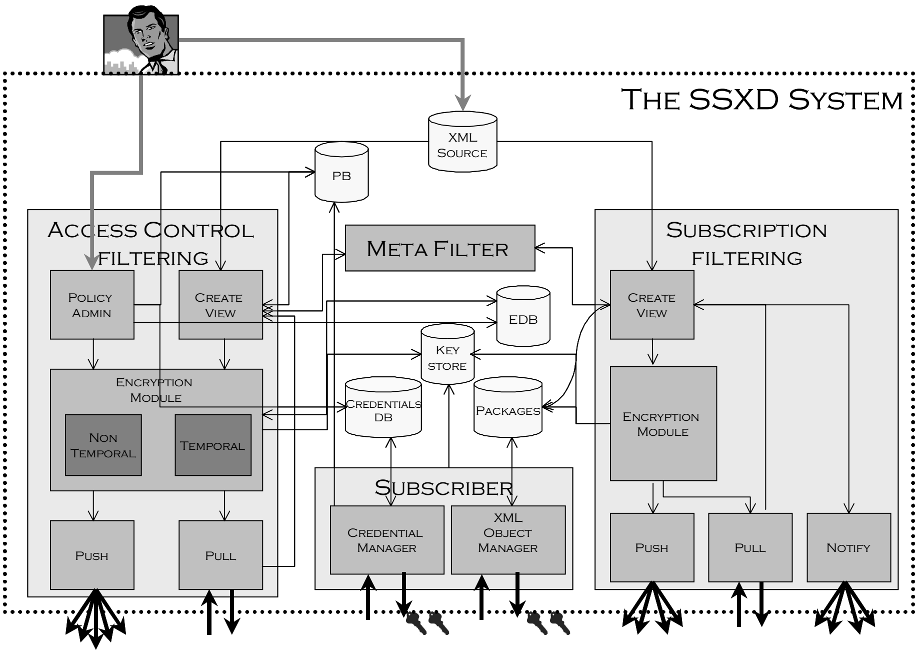
THE CORE SSXD ARCHITECTURE

According to the taxonomy previously introduced, we have designed an SSXD system supporting both the access control and the subscription paradigm. Our SSXD system, whose overall architecture is represented in *Figure 3*, is built on top of two systems previously developed by us, that is, the Author*X* (Bertino, 2001b) system, for a selective dissemination of XML documents according to a discretionary access control model, and the XPublisher (Bertino, 2001a) system, implementing a SSXD adopting a subscription-paradigm.

As depicted in *Figure 3*, the architecture includes two separate modules (i.e., the *Access Control Filtering* module, and the *Subscription Filtering module*, respectively) to implement these two data dissemination paradigms. Moreover, these modules have their own encryption scheme for push distribution and their own strategy for the view generation. However, it is possible to integrate the access control and subscription paradigms by means of the "Meta Filter" module. In particular, this module starts up each time a modification occurs on predefined data, that is, data on which both the access control and the subscription paradigms are defined.

Moreover, in addition to the modules implementing the access control and subscription dissemination mechanisms, the proposed SSXD architecture includes also a module managing the subscription phase, i.e., the *Subscriber* module. The aim of this module is to manage the registration of new users into

Figure 3. The SSXD core architecture



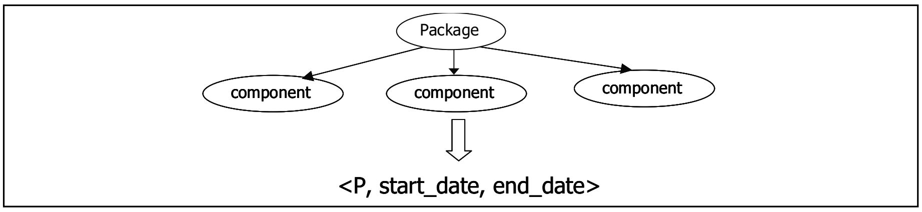
the SSXD system. Since our SSXD system combines both a subscription and an access control mechanism, the Subscriber must be able to manage the subscription phases of both of them. More precisely, as it will be explained in next sections, since the access control model adopted in our framework exploits the notion of credentials to identify the users, the Subscriber module needs to be able to manage credentials. By contrast, the user subscriptions are modelled as in the continual query paradigm, that is, they are defined as a set of queries (i.e., XPath expressions) (XPath, 1999) on the document source, which specify the information the user is interested in receiving. To cope with these requirements, the Subscriber module consists of two components: the *Credential Manager* and the *XML Object Manager*.

Since the Access Control Filtering and the Subscription Filtering modules are the core components of our SSXD system, in the following sections we will describe them in detail.

Subscription Filtering Module

The Subscription Filtering Module has been mainly conceived for a digital library-like scenario, where the most important requirement to be satisfied is the complete flexibility in the user subscription. According to this approach, our SSXDI system allows users to customize their subscriptions in terms of data they are interested in receiving, in terms of the subscription period during which the

Figure 4: Package structure



data must be sent, and in terms of the distribution mode under which the data are delivered. User subscriptions are defined according to the continual query paradigm, that is, they are defined as a set of queries (i.e., XPath expressions) (XPath, 1999) on the document source, which specify the information the user is interested in receiving. In such a context, the SSXD service has to ensure that information is accessible only during the subscription periods. In order to satisfy this requirement the user subscription contains also information on the subscription periods. More precisely, our approach is based on the concept of *package*. As depicted in *Figure 4*, a package is defined as a collection of *components*, where each component consists of one or more XPath expressions (denoted as P in *Figure 4*) specifying portions of information, together with a subscription period (i.e., the begin and the end date of the subscription period).

A further feature is that it supports several dissemination modes, by allowing the user to select, for each package, the distribution mode to apply to that package according to his/her needs. More precisely, a user can select, during the subscription phase, if the content of the package (i.e., the content of the portions specified in the package), must be delivered according to a *pull* or a *push* distribution. The push mode is further specialized into $push_{on}$ and $push_{off}$. If the $push_{on}$ mode is selected, the SSXD system, upon any modification of the source content, sends the modified portions to the user subscribed to a package to which the modified portions belong. By contrast, if the $push_{off}$ mode is selected, the system periodically (for instance once a day) checks which portions of the source have been modified and sends the updated portions to all the users subscribed to a package to which the portions belong. We introduce these two kinds of push-based distribution modes, making the SSXD system more adaptable to different user characteristics. Moreover, in order to further improve the flexibility service, a further distribution mode has been introduced in addition to the pull and push distribution modes, that is, the *notify* mode. If the notify mode is selected, the user only receives a notification (for instance an e-mail or an SMS) informing him/her that a modification in one of his/her package occurs, and he/she has to explicitly request the portion(s) he/she is interested in receiving.

Thus, the Subscription Filtering module gives the user complete freedom in defining the information he/she is interested in, in that a user can subscribe to different packages and each package may contain portions with different subscription periods, according to the user needs.

Example 1

Consider four users, U_1, U_2, $U_{3,}$ and U_4, and the XML document in *Figure 2*. Suppose that user U_1 is interested in receiving the MyNewspaper Frontpage in the period from 1/01/03 to 12/31/03, and all the newspaper articles whose title contains the phrase “XML security” in the period from 01/01/03 to 06/30/03. Moreover, suppose that U_1 specifies, during the subscription phase, the $push_{on}$ mode for all these portions. Suppose that user U_2 is interested in receiving the Frontpage of all the newspapers in the period from 03/01/03 to 12/31/03. Moreover, he wants to receive in the period from 03/01/03 to 12/31/03 all the articles written by Tom Red. Furthermore, U_2 specifies the Pull mode for all those portions. Suppose that user U_3 is interested in receiving the MyNewspaper Frontpage in the period from 01/01/03 to 06/30/03, and that for this portion he specifies the $push_{on}$ mode. Finally, suppose that user U_4 is interested in receiving the MyNewspaper Frontpage in the period from 01/01/03 to 02/28/03, and that for this portion he specifies the $Push_{on}$ mode. Thus, the XPath expressions generated during the subscription phase are the following:

P_1: the Frontpage element of the instances of the newspaper DTD;
P_2: the Frontpage element of the XML document in Figure 2;
P_3: the Article elements of all the instances of the newspaper DTD, where the attribute Title contains the phrase “XML security”;
P_4: the Article element of all the instances of the newspaper DTD, where the attribute Author has value “Tom Red”.

As a consequence of the different subscription periods selected by the users and of the different subscription modes, the packages the publishing service creates are the following:

$PA_1 = <U_1, <C_2, C_3>, Push_{on}>$
$PA_2 = <U_2, <C_1, C_4>, Pull>$
$PA_3 = <U_3, <C_5>, Push_{on}>$
$PA_4 = <U_4, <C_6>, Push_{on}>$

where the components are:

C_1=<<“//Newspaper/Frontpage//node()”>, 03/01/01, 12/31/01>
C_2=<<“//Newspaper[@Title=“Times”]/Frontpage/node()”>, 01/01/01, 12/31/01>

C_3*=<<"//Newspaper//Article[@Title="XML security"]/node()">, 01/01/01, 06/30/01>*
C_4*=<<"//Newspaper//Article[@Author="Tom Red"]/node()">,03/01/01, 06/30/01>*
C_5*=<<"//Newspaper[@Title="Times"]/Frontpage/node()">, 01/01/01,06/30/01>*
C_6*=<< "//Newspaper[@Title="Times"]/Frontpage/node()">,01/01/01, 02/28/01>.*

Security properties are enforced by adopting suitable encryption schemes. In particular, in this scenario, it is necessary to ensure that a user can access the portions of information to which he/she has subscribed for the duration of his/her subscription and that such information is no longer accessible by the user when the subscription expires. To fulfill these requirements, in Bertino (2001a) we have proposed different encryption schemes for the different distribution modes we support. More precisely, we exploit two different symmetric encryption schemes, namely the *Pull Package* encryption scheme, and the *Push Package* encryption scheme, which are briefly described in the following.

Pull Package Encryption Scheme

If a user subscribes to a package in the notify mode, each time a portion of a non-expired component of such package is modified the user receives a notification that advises him/her of the modification. Then, the user can decide if he/she is interested in receiving the modified portion(s) or not. If the user is interested in receiving the modified portion(s), the user has to explicitly request them to the SSXD system as in the case of pull mode packages. Thus, we can uniformly treat the distribution of packages with notify and pull mode, by using the Pull package encryption scheme. More precisely, in case of pull and notify packages the subscriber receives, upon the completion of the subscription phase, a symmetric key (Stallings, 2000), which is then used by the SSXD system to encrypt the portions of the package returned to the user as answer to an access request. In such a way, only the entitled user is able to decrypt the access request answer because he/she is the only one sharing that encryption key with the SSXD system. Indeed, the basic idea of this scheme is to associate a different symmetric key, called *package key*, with each different package defined with the pull or notify distribution mode. Package keys are returned by the SSXD system to a user as a result of the subscription phase. More precisely, during the subscription phase a user defines her/his packages on the basis of her/his needs. Then, as last step of the subscription, he/she specifies the distribution mode. If the user selects the pull or notify distribution mode, the SSXD system checks if the chosen package is an existing one (i.e., the same portions with the same subscription period and under the pull or notify mode). If this is the case, the

SSXD system sends the package key associated with such an existing package to the new-subscribed user, otherwise it generates a new one. Thus, different users many have the same package key only if they are subscribed to the same portions with the same subscription period and under the same mode (i.e., pull or notify).

Then, each time a user requests a pull or notify package, the SSXD service first checks which package portions have been modified from the previous request by the same user. Then it selects only those portions belonging to a non-expired component. Finally, it encrypts all those portions with the package key associated with the requested package and sends them to the requesting user.

Push Package Encryption Scheme

As for the notify and pull mode, also the push$_{on}$ and push$_{off}$ modes are uniformly treated. Indeed, the difference between push$_{on}$ and push$_{off}$ mode is only on the event that triggers the delivery of a package or of some of its portions. In both cases, the system checks whether the portions that have been modified are contained in a non-expired component. The time of this check depends on the distribution mode. In case of push$_{on}$, this check is executed upon each source update, whereas for push$_{off}$ this check is periodically executed.

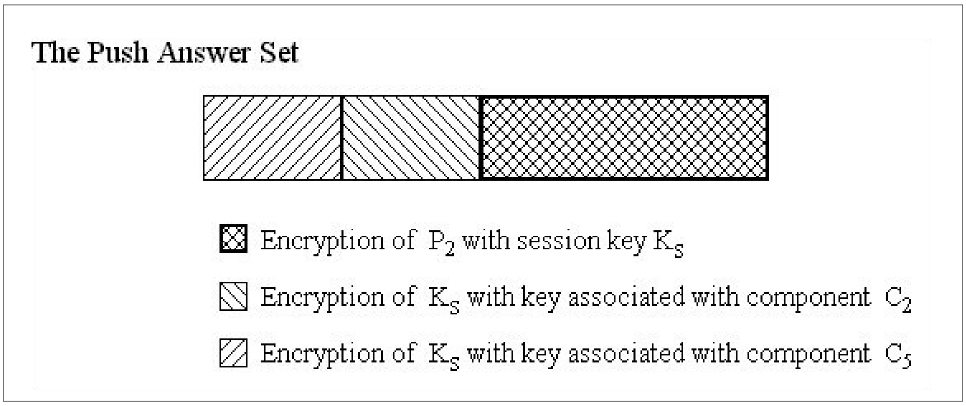
The push package encryption scheme aims at minimizing the number of keys that need to be generated by guaranteeing, at the same time, the security of information delivery. Indeed, the pull package encryption scheme is not efficient in this context, because the SSXD system has to broadcast the same portion of information, which may belong to several components or packages, to all the users entitled to access it. Thus, with the pull package encryption scheme it should encrypt the same content with a different key (i.e., the appropriate package key), and then send it to each different user. By contrast, to limit the number of keys that need to be generated, we use a technique based on session keys (Wool, 1998). In particular, the subscription Filtering module generates a different key for each component defined over the source. Once a user finishes the subscription phase, the SSXD system sends the user the keys associated with the components belonging to his/her packages. Then, when the SSXD system needs to send a modified portion, it encrypts it with a session key and broadcasts it to all the users subscribed to a package that contains a component defined over that portion and whose subscription period has not yet expired. Moreover, the SSXD system encrypts also the session key with the key associated with the component, and sends it together with the encrypted portions, generating thus a unique package, called *Push Answer Set*. Such an approach ensures that the portion can be accessed only by subscribers whose subscription period has not yet expired. Indeed, once a component is expired, the subscription period is over, the encryption key is no longer used. Moreover, such an approach ensures that only a limited number of keys need to be generated, since the set of components that can be defined over a source of information is limited.

Example 2

Consider the scenario of Example 1. Suppose that on 03/30/03 a new issue of MyNewspaper is made available at the source, and that this issue contains an article written by Tom Red. Moreover, suppose that this issue does not contain any article on XML Security. Therefore, the portions defined in Example 1 that are influenced by this update are: P_1, P_2, and P_4. Let us consider P_1. This portion denotes all the Frontpage elements of the instances of the DTD corresponding to the XML document in *Figure 2*. Thus, portion P_1 contains the Frontpage of the new issue of MyNewspaper. When a portion has been modified, the subscription filtering module checks whether the modified portion belongs to a non-expired component, contained into a package having a $push_{on}$ mode, and then creates the *Push Answer Set*. Considering the scenario of Example 1, portion P_1 belongs only to a non-expired component (i.e., C_1), but this component belongs to package PA_2, which has a pull distribution mode. Therefore, the module does not generate the Push Answer Set for P_1. By contrast, when the SSXD system checks portion P_2, it verifies that P_2 belongs to components C_2, C_5, and C_6. All the packages containing these components, that is, packages PA_1, PA_3, and PA_4, have a $push_{on}$ distribution mode. Thus, the SSXD system checks the subscription period of each of these components. It thus verifies that component C_6 is expired (on 02/28/03), whereas components C_2, and C_5 are not. Thus, the Push Answer Set consists of the encryption of P_2 with a session key K_s, the encryption of K_s with K_{C2}, that is, the key associated with component C_2, and the encryption of K_s with K_{C5}, that is, the key associated with component C_5. The Push Answer set for portion P_2 is illustrated in *Figure 5*.

Then, the system sends the Push Answer Set to the appropriate users, that for portion P_2 are users U_1 and U_3. Note that if the Push Answer is eavesdropped by other user, say U_4, he/she does not have the keys necessary to decrypt the

Figure 5: The Push Answer Set for portion P_2


The Push Answer Set
Encryption of P_2 with session key K_S
Encryption of K_S with key associated with component C_2
Encryption of K_S with key associated with component C_5

session key (indeed, U_4 has only key K_{C6} associated with component C_6). Thus, U_4 cannot decrypt portion P_2, since his/her subscription period to P_2 is expired.

Finally, when the SSXD system checks portion P_4, it verifies that it belongs to a non-expired component (i.e., C_4), and that C_4 is contained into package PA_2, whose distribution mode is Pull. Therefore, the Subscription Filtering module does not create the Push Answer Set for P_4. Let us suppose that after a week, user U_2 requests package PA_2. In this case, the SSXD system first identifies the portion of PA_2 belonging to non-expired components, that is, C_1 and C_4. Then, it verifies for each portions of those components (i.e., P_1 and P_4) if it has been modified from the last request by U_2. Suppose that the last request has been done on 03/29/03, therefore all portion contents are new for user U_2. Thus, the SSXD system creates the Pull Answer Set for U_2, by encrypting portions P_1 and P_4 with the key associated with package PA_2. Then, the SSXD system sends this Pull Answer Set to U_2.

Access Control Filtering Module

The access control module has been designed in the framework of the Author*X* project (Bertino, 2001b) and it provides discretionary access control for XML documents. The access control model (Bertino, 2002b), on which the Access Control Filtering module is based, takes into account for policy specification XML document characteristics, the presence of DTDs/XML schemas describing the structure of documents at a schema level, and the types of actions that can be executed on XML documents (i.e., navigation, browsing, and update). The access control model provides a fine-grained access control, in that it is possible to specify policy that apply to collection of XML documents (for instance, all the documents instance of a DTD/XML Schema), or selected portions within a document (e.g., an element or attribute). Additionally, access control policies are characterized by a temporal dimension, in that it is possible to express policies that hold only for specific periods of time (such as for instance a particular day of the week). This is a relevant feature because very often users must have the right to access a document or a document portion only in specific periods of time.

Furthermore, access control model supports the specification of *user credentials* as a way to enforce access control based on user qualifications and profiles.

In the following sections we briefly explain the pull and push distribution, by mainly focusing on the information push mechanisms since it represents the most novel approach to data dissemination. We refer the interested reader to Bertino (2001b) for a detailed description of the information pull mechanism implemented in Author*X*, as well as to Bertino (2002b) for an exhaustive explanation of the access control model on which the Access Control Filtering module relies.

Pull Distribution

Access control usually takes place in conventional DBMSs according to this traditional mode. If the pull mode is adopted, the users explicitly require the XML documents (or portions of documents) when needed. Upon a document request, Access Control Filtering module first verifies whether the requesting user is entitled to access the requested document, according to the specified access control policies. Based on these authorizations, the user is returned a *view* of the requested document(s), consisting of all and only those portions for which he/she has a corresponding authorization. When no authorizations are found, the access is denied.

Push Distribution

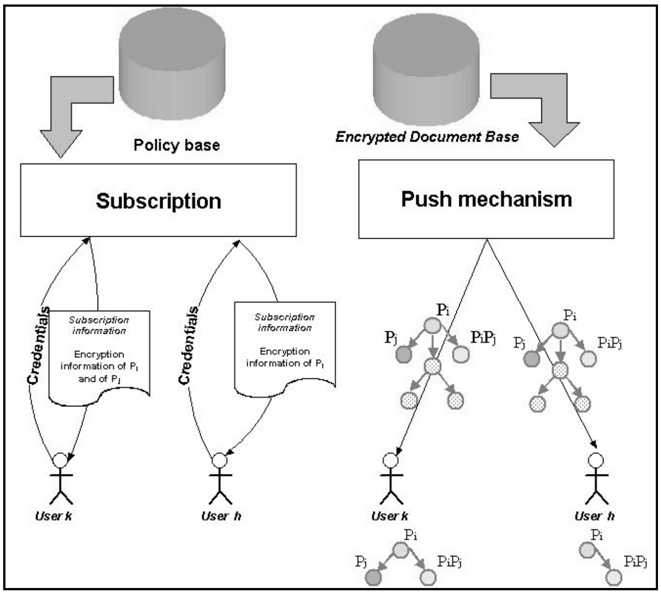
The push distribution mode is particularly suitable for XML documents that must be released to a large community of users and which show a regular behavior with respect to their release. Also in this case, different users may have privileges to see different, selected portions of the same document based on the specified access control policies. To respect these different privileges, the mechanism enforcing information push must ensure that the correct view is delivered to each different user. Obviously the naïve solution to generate different physical views of the same document for each group of users to which the same access control policies apply is impracticable, since it could imply, in the worst case, the generation of a different view for each different user. Moreover, after the generation of the views, the Access Control Filtering module should properly deliver all these views to the interested users. In this scenario, we need to take in consideration that due to the possibly high number of users accessing an XML source, and the wide range of access granularities provided by the underlying access control model, the number of these views might become considerably large and, thus, such an approach cannot be practically applied.

For these reasons, to efficiently support information push we adopt an approach based on the use of broadcast encryption techniques. According to this technique, the first step is the development of a strategy to generate a correct document encryption, that is, a document encryption ensuring that all the users are able to decrypt, and thus to access, all and only the authorized portions. In order to obtain a correct encryption, Access Control Filtering module encrypts all the portions of an XML document to which the same *policy configuration* applies with the same secret key, where with the term policy configuration we denote a set of access control policies. We refer to these encryptions as well-formed encryptions. The encrypted copy of the document is then placed into the Encrypted Document Base (EDB), which stores the encrypted copies of all the documents to be released under the push mode. We refer the interested reader to Bertino (2002b) for a detailed description of the algorithms computing the well-formed encryption. More precisely, the generation of the document encryption is performed by two main algorithms. First, the XML document undergoes a *marking phase* in which each document node is marked with the access control policies that apply to it. Then, the second algorithm groups all the nodes with the same policy configuration and encrypts them with the same key. The same encrypted copy of the document is then distributed to all users, whereas each user only receives the key(s) for the portion(s) he/she is authorized to access.

The main issue in the generation of a well-formed encryption is that it could imply the management of a high number of secret keys. For instance, by using a traditional symmetric scheme, in the worst case the well-formed encryption implies the generation of 2^{Np} different secret keys, where Np is the number of access control policies defined for the XML source, that is, one for each different policy configuration that could be generated from Np policies. Moreover, the problem of key generation and management is further complicated by the temporal dimension associated with access control policies. To cope with such a high number of secret keys, in Bertino (2002a) it has been proposed a flexible key assignment scheme [adapted from (Tzeng, 2001)], which greatly reduces the number of keys that need to be managed.

More precisely, the adopted key assignment scheme relies on the framework depicted in *Figure 6*. Each user is required to register to the SSXD system, during a mandatory *subscription phase*. During the subscription phase, a user can be assigned one or more credentials, which are stored at the service site. As a result of the subscription phase, SSXD returns the user some information, called *subscription information*. In particular, the subscription information

Figure 6: Push mechanism enforcement



allows the user to decrypt all the nodes that he/she is entitled to access according to the specified access control policies and only for the set of time instants representing the validity period of the policies. More precisely, the subscription information consists of a set of parameters and additional information, one for each policy the user satisfies, called *encryption information*.

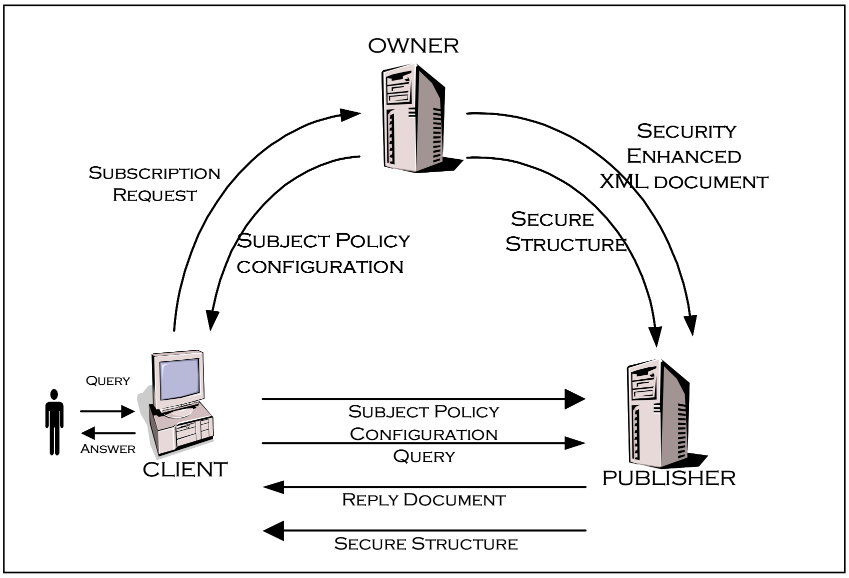
The problem of key generation and management is further complicated by the temporal dimension associated with our access control policies. To manage the temporal dimension of access control policies, the encryption key is obtained by combining the encryption information associated with a policy configuration with generic temporal information, denoted as w_t. Thus, at each time granule *t*, the system verifies whether a modification occurs in the XML source. In this case, the system groups the modified source portions according to their policy configurations. Then, for each group it generates a secret key by which it encrypts the modified portions. The secret key is the result of hashing the temporal information w_t together with the encryption information associated with the policy configuration marking the interested portions. Thus, in order to decrypt these portions, a user needs to have both the temporal information w_t and the encryption information associated with the policy configuration.

The most relevant aspects of the proposed key assignment scheme is that it is defined in such a way that, from the encryption information associated with a policy $acp_{j,}$ it is possible to derive all and only the encryption information associated with policy configurations containing acp_j. Thus, using this management scheme the system has to manage only Np different encryption information, whereas the others associated with the remaining policy configurations can be derived only when needed. Moreover, by using only the encryption information received during the subscription phase, a user is able to generate all the keys associated with the policy configurations he/she satisfies, and to generate all the w_ts corresponding to time granules belonging to the interval of validity of the policies he/she satisfies.

SCALABILITY ISSUES IN SECURE SELECTIVE DATA DISSEMINATION

Given the possibly high number of subjects accessing a SSDI system, one of the most important issues is the scalability of the service. In the last few years, to overcome scalability problems, a new paradigm for data dissemination over the Web is receiving growing attention: the third-party architecture. A third-party architecture 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 queries submitted by subjects. A relevant security issue in this architecture is how the Owner can ensure a secure

Figure 7: A third-party architecture



dissemination of its data, even if the data are managed by a third-party that can be untrusted with the considered properties. In Bertino (2002c), we have proposed a third-party architecture for SSDI of XML data, by which a user is able to verify the authenticity, integrity and completeness of the answer received by an untrusted Publisher. In the following, we briefly summarize the overall architecture by explaining the role of each party in the architecture. Then, in next sections we give more details on authentication and completeness verification.

Owner

In our approach, the Owner specifies access control policies over the XML source, and it sends them to the Publishers together with the documents they are entitled to manage. For each document, the Owner sends the Publisher also some security information needed by the Publisher to correctly answer subject's queries. More precisely, the Owner supplies the Publisher with information on which subjects can access which portions of the document, according to the access control policies it has specified. Access control policies are specified according to the access control model presented in Bertino (2002b). In this scenario, in order to ensure authenticity it is not enough that the 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 which requires that the Owner sends the Publisher, in addition to the documents it is entitled to manage, a summary signature (called *Merkle Signature*) for each managed document, generated using a technique based on Merkle hash trees (Merkle, 1989). The idea is that, when a subject submits a query to a Publisher, the Publisher 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.

All this additional information is encoded in XML and attached to the original document. The original document complemented with this additional information is called the *security enhanced XML document* (called SE-XML, hereafter). An example of SE-XML document is presented in *Figure 8*. The information contained into the SE-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 Publisher receives from the Owner some additional information about the structure of the original XML document, which must be sent in turn to interested subjects. This information is encoded into an XML document, called *secure structure* (called ST-XML, hereafter), which contains an obfuscated version of the structure of the XML document. Moreover, the secure structure is complemented with its Merkle Signature, to prevent Publisher alterations. An example of ST-XML document is presented in *Figure 11*. Further details on the ST-XML generation will be provided in section devoted to the completeness property.

Subjects

As depicted in *Figure 7*, subjects are required to register with the Owner, through a mandatory subscription phase. As a result of the subscription process, the Owner returns the subject a data object, called the *subject policy configuration*, which stores information on the access authorizations that apply to the subject, according to the policies stated by the Owner. Since in a third-party architecture the subject policy configuration plays the role of a certificate, it must be signed with the private key of the Owner, to prevent the subject from altering its content.

Publisher

Once the subscription phase has been completed, a subject can submit queries to a Publisher. As reported in *Figure 7*, when a subject submits a query, it also sends the Publisher its subject policy configuration to enable the Publisher to determine which access control policies apply to the subject. On the basis of the subject 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 into 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-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 document portions not contained in the view). Both the view and the additional information are locally computed by the Publisher by considering only the SE-XML version(s) of the requested document(s). The result is an XML document, called *reply document*. In addition to the reply document, the Publisher sends the subject also the *secure structure* associated with the document on which the query applies. 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* received by the Publisher.

In the next sections we describe in more in detail how a subject can verify the authenticity and the completeness of a query answer.

Authenticity Property

In this section we focus on authenticity verification. More precisely, we consider each single party of the architecture by pointing out its role in the authenticity verification process.

The Owner: Generation of the SE-XML Document

To make the Publisher able to answer subject queries and the subjects able to verify the authenticity of the query answers, the Owner inserts into the XML documents sent to the Publishers two distinct pieces of information: the *Merkle signature*, and a set of *Policy Information*. In the following we briefly explain both of them.

Merkle Signature

Traditional digital signature techniques are not suitable to ensure authenticity in a third-party architecture. Indeed, since the Publisher may return a subject only selected portions of an XML document, depending on the query the subject submits and on the access control policies in place, the subject could not be able to validate the Owner signature, which is computed over the whole XML document. We need, thus, a mechanism allowing the Publisher to prune some nodes and, at the same time, allow the subject to verify the Owner signature having only the returned view and some additional information.

In our approach, we propose a different way to sign an XML document, which is based on an alternative mode to compute the *digest*. The function we use to compute the digest value is the Merkle function (denoted in the following as *MhX()*), which exploits the Merkle tree authentication mechanism proposed in Merkle (1989). By this function, it is possible to univocally associate a hash value with a whole XML document through a recursive computation. As depicted in *Figure 9*, the basic idea is to associate a hash value with each node in the graph representation of an XML document. 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. Once the digest of the XML document is computed (i.e., the Merkle hash value of the root of the document), it is then encrypted with the private key of the Owner, generating what we call the *Merkle Signature*.

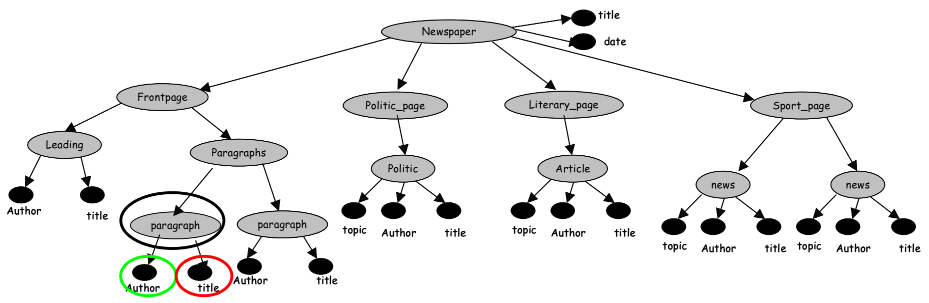
By using Merkle signatures the Owner is able to apply a unique digital signature on an XML document by ensuring at the same time the authenticity and integrity of both the whole document, as well as of any portion of it. The Merkle

Figure 8: An example of SE-XML document

```
<?xml version="1.0" encoding="UTF-8" ?>
 <Annual-report Year="2002" name="University of Milano" Sign="CG4g3D8/mPVV/t+T2O1kZRFhdio=">
  <Policy>1,2,3,4,5,6,7</Policy>
 <Assets>
   <Asset Dept="DICO">
        <Expenses Tot="...." PC_ATTR="08"/>
         <Funds>
    <Fund Funding-Date="15/09/2002" Type="CNR" Amount="…" PC_ATTR="080808"/>
        </Funds>
  </Asset>
   <Asset Dept="EED">
        <Expenses Tot="...." PC_ATTR="02"/>
         <Funds>
         <Fund Funding-Date="01/20/2002" Type="CNR" Amount="..." PC_ATTR="060602"/>
         <Fund Funding-Date="06/01/2002" Type="MURST" Amount="..." PC_ATTR="060602"/>
        </Funds>
  </Asset>
</Assets>
<Patents>
 <Patent date="02/26/2002" Id-Pat="...." Dept="DICO" PC_ATTR="808080">
   <Short-descr PC="90">.....</Short-descr>
  <Tech-details PC="80">.....</Tech-details>
  <Authors PC="90">....</Authors>
   </Patent>
 <Patent date="05/12/2002" Id-Pat="...." Dept="EED" PC_ATTR="202020">
   <Short-descr PC="60">.....</Short-descr>
   <Tech-details PC="20">.....</Tech-details>
<Authors PC="60">....</Authors>
   </Patent> </Patents>

</Annual-report>
```

Figure 9: The Merkle function



MhX(Author)=h(h(Author)||h(Author.value)) MhX(title)=h(h(title)||h(title.value))

MhX(paragraph)=h(h(paragraph)||h(paragraph.content)||MhX(Author)||MhX(title))

signature of an XML document is inserted by the Owner into the SE-XML document, by means of the Sign attribute (see *Figure 8*).

Policy Information

In addition to the Merkle signature, the Owner inserts in the SE-XML document also information about the access control policies applied on the corresponding XML document. More precisely, the Owner inserts into each element, to which at least one policy applies, information about the policy configuration applied to such an element. Policy information is specified at the element level, since different access control policies can apply to different portions (i.e., 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 SE-XML document (i.e., PC attribute). Whereas, to store information about policies that apply to attributes, we propose a slightly different approach. The idea is to store into a unique attribute associate with an element, called PC_ATTR, the concatenation of the policy configurations of the element attributes.

More precisely, the policy configuration is the hexadecimal encoding of a binary string where each bit represents the state of a policy (1, if the policy is applied to the node, 0 otherwise). This means that the *i-th* bit of a policy configuration is set to 1 if and only if the policy whose identifier is *i* is applied to that element. We have to note that, since the number of policies defined by the Owner can be very large, this approach implies very large identifiers, and, as a consequence, a large string as policy configuration. For this reason, we have proposed to represent a *local policy configuration*, that is, a binary string built by considering only the policies applied to the document. Thus, we need to insert in addition to the policy configuration also information about the set of access control policies applied to a document. This information is contained into the Policy element (see *Figure 8*).

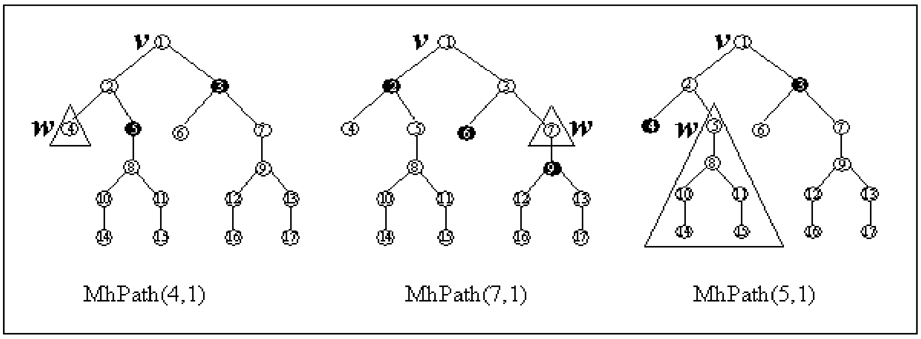
The Publisher: Generation of the Reply Document

When a subject submits a query to a Publisher, together with the query, he/she also sends information on the policies he/she satisfies (i.e., the subject policy configuration). On the basis of the subject policy configuration and the submitted query, the Publisher computes a view of the requested document, which contains all and only those portions of the requested document for which the subject has an authorization. Then, besides the query result, the Publisher returns the requesting subject also the Merkle 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 the source 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 Merkle 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. This additional information is called *Merkle Hash Paths*. Both the view and the Merkle Hash Paths are locally computed by the Publisher 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 its content (i.e., the Merkle signature, the Merkle hash paths, and the query answer), the authenticity of the answer, without interacting with the Owner. The idea is that by performing a computation on the Merkle hash Paths and the nodes of the query answer and by comparing the result with the Merkle signature of the XML document, the subject is able to verify the authenticity of the answer. The next section deals with Merkle hash paths computation.

Merkle Hash Paths

Merkle Hash paths can be intuitively defined as the set of the Merkle Hash values of those nodes pruned during query evaluation. In general, given two nodes *v*, *w* such that *v* belongs to Path(*w*) (where Path(*w*) denotes the set of nodes connecting a node *w* to the root of the corresponding document), the Merkle Hash path between *w* and *v*, denoted as *MhPath(w,v)*, is the set of Merkle Hash values needed to compute the Merkle Hash value of *v* having the Merkle Hash value of *w*. More precisely, the Merkle Hash path between *w* and *v* consists of all the Merkle Hash values of *w*'s siblings, together with the Merkle Hash values of all the siblings of the nodes belonging to the path connecting *w* to *v*.

Figure 10: Examples of Merkle Hash paths



To better clarify how the proposed approach works, let us consider *Figure 10*, which depicts three different examples of Merkle hash paths. In the graph representation adopted in *Figure 10* we do not distinguish elements from attributes, by treating them as generic nodes. In the figure, triangles denote the view returned to the subject, whereas black circles represent the nodes whose Merkle hash values are returned together with the view, that is, the Merkle hash paths. In the first example, the view consists of a leaf node *w*. In such a case, the Merkle hash path between nodes 4 and 1 consists of the Merkle hash values of nodes 5 and 3. Indeed, by using node *w* (i.e., 4) and the Merkle hash value of node *5* it is possible to compute the Merkle hash value of node 2. Then, by using the Merkle hash values of 2 and 3, it is possible to compute the Merkle hash value of 1. In the second tree depicted in *Figure 10*, the view consists of a non-leaf node. In such a case MhPath(7,1) also contains the Merkle hash value of the child of node 7, that is, node 9. Thus, by using the Merkle hash value of node 9 and node 7, it is possible to compute the Merkle hash value of 7. Then, by using this value and the Merkle hash value of node 6, it is possible to generate the Merkle hash value of node 3. Finally, by using the Merkle hash values of nodes 3 and 2 it is possible to generate the Merkle hash value of node 1. By contrast, in the third example, the view consists of the whole sub-tree rooted at node 5. In such a case, MhPath(5,1) does not contain the hash values of the children of node 5. Indeed, having the whole subtree rooted at 5, it is possible to compute the Merkle hash value of node 5 without the need of any further information.

Completeness Property

The Merkle signature allows a subject to detect an alteration on the content of an XML document (or portions of it). More precisely, the subject is able to verify that the content/value and the tagname/name of an element/attribute have not been modified by an intruder. However, by using only the Merkle signature, the subject is not able to verify if the Publisher sends him/her the correct view, that is, a view containing all the portions of the XML document answering the submitted query and satisfying access control policies.

To make the subject able to verify the completeness of the view, we have introduced the *secure structure* of an XML document, which basically gives information about the structure of the documents on which the query is submitted. Indeed, by secure structure of an XML document we mean the XML document without its element contents, thus, containing only the names of the tags and all the attributes of the XML document (that is, the values and names of the attributes). Moreover, to avoid allowing the subject to see tagname and attribute values of portions he/she may not be authorized to access, we impose that each tag and attribute name and value has to be hashed with a standard hash function, before being inserted into the secure structure (see *Figure 11*).

This information makes the subject able to locally perform on the secure structure all queries whose conditions are against the document structure of the original document or on the attribute values. Indeed, these queries specify their conditions by using only the tag/attribute names, and the attribute value, which are information all contained as hash values in the secure structure. The proposed solution for completeness verification implies the translation of the submitted query *q*, by substituting the tag/attribute name and attribute values with the corresponding hash values. Afterwards, 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 document *d*, answering query *q*. We have to note that since the Publisher generates the view according to the access control policies stated by the Owner, during the completeness verification the subject must consider also the access control policies specified on the document. For this reason, we have inserted in the

Figure 11: An example of secure structure

```
<?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="fNhtL"  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-82592553>
```

secure structure also the Policy element, PC, and PC_ATTR attributes, that is, the attributes containing the policy information of the elements and attributes.

Additionally, in order to prevent alterations by the Publisher, the Owner computes the Merkle Signature of the secure structure, and, similarly to the SE-XML document, it sends this signature to Publishers together with the corresponding secure structure (i.e., the Sign attribute).

CONCLUSION

The chapter has focused on SSDI systems. We have first provided an overview of the work carried out in the field then we have focused on the security properties that an SSDI system must satisfy and on some of the strategies and mechanisms that can be used to ensure them. More precisely, since XML is the emerging standard today for data exchange over the Web, we have cast our attention on SSDI systems for XML documents (SSXD). As a result, we have presented a SSXD system providing a comprehensive solution to XML documents. Finally, since the third-party architecture is receiving growing attention as a new paradigm for data dissemination over the Web, we have discussed a new architecture for SSXDI system, which has the benefit of improving the scalability.

REFERENCES

Alert. (2003*). ALERT project web page*. Available on the World Wide Web at: http://alert.uni-duisburg.de/.

Altinel, M., & Franklin, M.J. (2000, September). Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th VLDB Conference* (pp. 53-64), Cairo, Egypt.

Belkin, N.J., & Croft, B.W. (1987). Retrieval techniques. *Annual Review of Information Science and Technology* (109-145). Elsevier.

Bertino, E., & Ferrari E. (2002b). Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security* (TISSEC), *5* (3), 290-331.

Bertino, E., Carminati, B.,& Ferrari, E. (2001a). A secure publishing service for digital libraries of XML documents. In *Information Security Conference (ISC01)* (pp. 347-362), LNCS 2200, Malaga, Spain. Springer-Verlag.

Bertino, E., Castano, S., & Ferrari E. (2001b, May/June). AuthorX: A comprehensive system for securing XML documents. *IEEE Internet Computing*, *5* (3), 21-31.

Bertino, E., Carminati, B., & Ferrari, E. (2002a, November). A temporal key management scheme for broadcasting XML documents. In *Proceedings.*

of the 9th ACM Conference on Computer and Communications Security (CCS'02), Washington. ACM Press.

Bertino, E., Carminati, B., Ferrari, E., Thuraisingham, B., & Gupta, A. (2002c). *Selective and authentic third-party distribution of XML document*. Accepted for publication with IEEE Transactions on Knowledge and Data Engineering (TKDE), February 2002. Available on the World Wide Web: http://ebusiness.mit.edu/research/papers/187_Gupta_XML.pdf.

Bray, T., & Paoli, J., & Sperberg-McQueen, C.M. (1998, February). *Extensible Markup Language (XML) 1.0*. W3C Recommendation.

Carzaniga, A., Rosenblum, D.S., & Wolf, A.L. (2000). Achieving scalability and expressiveness in an Internet-scale event notification service. In *Symposium on Principles of Distributed Computing,* (pp. 219-227).

Chen, J., DeWitt, D., Tian, F., & Wang, Y. (2000, May). NiagaraCQ: A scalable continuous query system for Internet databases. In *Proceedings of the ACM SIGMOD Conference,* Dallas, Texas.

Diao, Y., & Franklin, M.J. (2003). High-performance XML filtering: An overview of YFilter. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*.

Fiat, A., & Noar, M. (1994). Broadcast encryption. In *Advances in Cryptology* (Crypto 93), (pp. 480-491). LNCS 773. New York: Springer-Verlag.

Housman, E.M. (1973). Selective dissemination of information. *Annual Review of Information Science and Technology*, *8*, 221-241.

Liu, L., Pu, C., & Tang, W. (1999, January). Continual queries for Internet scale event-driven information delivery. Special Issue on Web Technologies, *IEEE Transactions on Knowledge and Data Engineering* (TKDE) (Special Issue on Web Technologies).

Loeb, S., & Terry, D. (1992). Information filtering. *Communications of the ACM*, *12* (35), 26-28.

Luhn, H.P. (1958). A business intelligent system. *IBM Journal of Research and Development, 4* (2), 314-319.

Luhn, H.P. (1961). Selective dissemination of new scientific information with the aid of electronic processing equipment. *American Documentation*, *12*, 131-138.

Merkle, R.C. (1989). A certified digital signature. *Advances in Cryptology-Crypto '89*.

Pereira, J., Fabret, F., Llirbat, F., Jacobsen, H.A., & Shasha, D. (2001). *WebFilter: A High-throughput XML-Based Publish and Subscribe System*.

Robertson, S.E. (1977). The probability ranking principle in IR. *J. Doc, 33* (4), 294-304.

Salton, G. (1989). *Automatic text processing*. Addison Wesley, 1989.

Salton, G., & McGill, M.J. (1983). *Introduction to modern information retrieval*. McGraw-Hill.

Stallings, W. (2000). *Network security essentials: Applications and standards*. Prentice Hall.

Strom, R., Banavar, G., Chandra, T., Kaplan, M., Miller, K., Mukherjee, B., Sturman, D., & Ward, M. (1998). *Gryphon*: *An information flow based approach to message brokering*.

Terry, D. B., Goldberg, D., Nichols, D. A., & Oki, B. M. (1992, June). Continuous queries over append-only databases. In *Proceedings of ACM SIGMOD Conferences* (pp. 321-330).

Tzeng, Wen-Guey. (2002). A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Transactions on knowledge and Data Engineering* (TKDE), *14* (1), 182-188.

Wool, A. (1998, November). Key Management for Encrypted Broadcast. In *Proceedings of the 5th ACM Conference Computer and Communication Security* (pp. 7-16), San Francisco, CA.

Xpath. (1999). *Word Wide Web Consortium. XML Path Language (Xpath), 1.0.* W3C Recommendation. Available on the World Wide Web: http://www.w3.org/TR/xpath.

Yan, T.W., & Garcìa-Molina, H. (1994a). Index structures for selective dissemination of information under the Boolean model. *ACM Transactions on Database Systems* (TODS), *2* (19), 332-334.

Yan, T.W., & Garcìa-Molina, H. (1994b). Index structures for information filtering under the vector space model. In *Proceedings of International Conference on Data Engineering* (pp. 337-347). IEEE Computer Society Press.

Yan, T.W., & Garcìa-Molina, H. (1999). The SIFT information dissemination system. *ACM Transactions on Database Systems, 4* (24), 529-565. <Article topic=”Movies” Author=”...” Title=”...”>

ENDNOTES

[1] In this section, the term ‘user profile’ means the set of information associated to a user, relevant for the selective dissemination (e.g., the user preference, subscription information or the access control policies applied to the user).