XML Security

Claudio A. Ardagna, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati

Università degli Studi di Milano
Italia

Summary. The extensible markup language (XML) is a markup language promoted by the World Wide Web consortium (W3C). XML overcomes the limitations of hypertext markup language (HTML) and represents an important opportunity to solve the problem of protecting information distributed on the Web, with the definition of access restrictions directly on the structure and content of the document. This chapter summarizes the key XML security technologies and provides an overview of how they fit together and with XML. It should serve as a roadmap for future research and basis for further exploration of relevant scientific literature and standard specifications.

6.1 Introduction

Accessing information on the global Internet has become an essential requirement of the modern economy. Recently, focus has shifted from access to traditional information stored in WWW sites to access to large *e-services* such as e-government services, remote banking, or airline reservation systems. Security has always been of paramount importance to ensure data protection and transactions integrity and to maintain information privacy and confidentiality. In today's web-based business environment, however, the means for providing that security have changed dramatically. One of the most challenging problems in managing large, distributed, and heterogeneous networked systems is specifying and enforcing security policies regulating interactions between parties and access to services and resources. Superimposing a single pervasive security infrastructure over the Internet turned out to be difficult, due to system heterogeneity and conflicting security requirements.

An essential requirement of new Internet-wide security standards is that they apply to content created using *extensible markup language* (XML) [1, 2]. XML has been adopted widely for a great variety of applications and types of content. Examples of XML-based markup languages are security assertion markup language (SAML) (see [3] for more details) used to exchange security credentials among different parties, geography markup language (GML), wireless markup language (WML), physical markup language (PML), and mathematical markup language (MathML) (`http://www.w3.org/Math/`), just to name a few. XML is also at the basis of interoperability protocols used to integrate applications across the Internet, such as Web services protocols: the Web service technology relies on different XML-based languages such as Simple Object Access Protocol (SOAP), Web Service Definition Language (WSDL), and Universal Description Discovery and Integration (UDDI) [4]. In this scenario, it is necessary to provide integrity, confidentiality and other security benefits to XML documents or portions of them, in a way that does not prevent further processing by standard XML tools.

Traditionally, XML security has developed along two distinct though related lines of research, corresponding to two facets of the XML security notion. The first facet defines XML security as a set of security techniques (access control [5], differential encryption [6], digital signature [7]) tightly coupled with XML to maintain the main features of the XML semi-structured data model while adding to it all necessary security capabilities. This is especially important in XML-based protocols, such as SOAP [8], which are explicitly designed to allow intermediary processing and modification of messages [9]. XML security relies on some legacy security algorithms and tools, but the actual formats used to implement security requirements are specifically aimed at XML applications, supporting common XML technical approaches for managing content, such as specifying content with uniform resource identifier strings (URIs) or using other XML standard definitions like XPath and XQuery for locating portions of XML content. A second important facet of XML security deals with models and languages specifying and exchanging access control policies to generic resources (see Chap. 4 for more details), which may or may not comply with the XML data model [10]. XML appears in fact a natural choice as the basis for the common security policy language, due to the ease with which its syntax and semantics can be extended and the widespread support that it enjoys from all the main platform and tool vendors. To this purpose, several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources. One of the most important XML-based language is extensible access control markup language (XACML) [11, 12], a language for defining rules and policies for controlling access to information. Another security aspect that needs to be taken into consideration is the secure and selective dissemination of XML documents. Often, XML documents contain information with different level of sensitivity, which has to be shared by user communities and managed according to access control policies.

In this chapter, we illustrate recent proposals and ongoing work addressing XML security. The remainder of this chapter is organized as follows. Section 6.2 describes the main characteristics of XML signature and XML encryption. We also briefly review the XML key management specification.

Section 6.3 describes the XACML policy language and the WS-Policy language. Finally, Sect. 6.4 gives our conclusions.

6.2 XML Data Protection

Current security technologies are not sufficient for securing business transactions on the Net. XML represents an important opportunity to solve the problem of protecting information distributed on the Web, by ensuring authenticity, data integrity, and support for nonrepudiation. To this purpose, two important initiatives are *XML signature* [7, 13] and *XML encryption*. XML signature is a joint effort between the World Wide Web consortium (W3C) and the internet engineering task force (IETF), and XML encryption is a W3C effort. In the remainder of this section, we first describe the main characteristics of these two proposals and then briefly present the XML key management specification.

6.2.1 XML Signature

An XML signature is a digital signature obtained by applying a digital signature operation to arbitrary data. The concept of a digital signature is not new and several technologies have already been presented to the community (e.g., public key cryptography standards [14]). However, while the existing technologies allow us to sign only a whole XML document, XML signature provides a means to sign a portion of a document. This functionality is very important in a distributed multi party environment, where the necessity to sign only a portion of a document arises whenever changes and additions to the document are required. For instance, consider a patient record stored in a hospital repository. This record can contain several entries (diagnoses) coming from several doctors. Each doctor wants to take responsibility only over her diagnosis. In this case, every additional diagnosis added to the patient record must be singularly signed. This important feature is supported by XML signature. The extensible nature of XML also allows support for multiple signatures inside the same document. It is also important to highlight that the possibility of signing online a portion of a document and inserting the signature inside the document avoids the development of ad hoc methods to manage persistent signatures, and provides a flexible mechanism to sign and preserve part of the document.

The data to be signed are first digested (a digest is a fixed-length representation of a resource and is created using, for example, a hash function such as SHA-1) and the resulting value is placed in an element, called `DigestValue`, together with other information. This element is then digested and cryptographically signed. An XML signature is inserted in the `signature` element and it can be associated with the data objects in three different ways: (i) enveloping signature, where the `signature` element embedded the data

```
<patient>
  <patientId>123a45d</patientId>
  <diagnosis id="Diagnosis001">...</diagnosis>
  <Signature Id="Signature001" xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference URI="#Diagnosis001">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>dh5gf68fhgfjt7FHfdgS55FghG=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>...</KeyInfo>
  </Signature>
</patient>
```

Fig. 6.1. An example of internal XML detached signature

to be signed; (ii) enveloped signature, where the `signature` is a child element of the data to be signed; (iii) detached signature, where the `signature` element and the signed data objects are separated. Figure 6.1 illustrates an example of internal detached signature, where a doctor's diagnosis (element `diagnosis`) is signed. As is visible from this example, the `signature` element is inserted within the XML document as a sibling of the signed element. The `signature` element contains three subelements: `SignedInfo`, `SignatureValue`, and `KeyInfo`.

The required `SignedInfo` element contains the information signed and has three subelements: the required `CanonicalizationMethod` element defines the algorithm used to canonicalize the `SignedInfo` element before it is signed or validated; the required `SignatureMethod` element specifies the digital signature algorithm used to generate the signature (DSA-SHA1, in our example); one or more `Reference` elements identify the data that is digested via a URI. The `Reference` element contains: an option `Transforms` element that in turn contains a list of one or more `Transform` elements describing a transformation algorithm used to transform the data before they are digested; the `DigestMethod` element specifies the method used to generate the digest value reported in the `DigestValue` element.

The `SignatureValue` element contains the signature value computed over the `SignedInfo` element.

Finally, the `KeyInfo` element indicates the key that must be used for signature validation.

6.2.2 XML Encryption

XML encryption [6] can be used to encrypt arbitrary data. As for XML signature, the main advantage given by XML encryption is that it supports the

```
<patient>
  <patientId>123a45d</patientId>
  <diagnosis id="Diagnosis001">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          ...
        </ds:KeyInfo>
        <CipherData>
          <CipherValue>H343HJS90F</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </diagnosis>
</patient>
```

Fig. 6.2. An example of XML encryption

encryption of specific portions of an XML document rather than the complete document. This feature is particularly important in a business scenario, where different remote parties cooperate to provide a service. A consequence of partial encryption is also support for multiple encryptions. For instance, in a health-care scenario, when a patient goes to a hospital for a visit, her record contains both doctor's diagnosis and information for billing payment. In this case payment information must not be seen by a doctor and diagnosis must not be seen by the billing administrator. This requirement can be obtained by encrypting the two types of information using a different encryption key. XML encryption supports encryption at different granularity levels: document, element, and element-content level. As an example, suppose that we need to encrypt the `diagnosis` specified within a patient record. Figure 6.2 illustrates the XML encryption, where the content of the `diagnosis` element has been replaced by the `EncryptedData` element with attribute `Type`, which specifies the type of the encrypted data (`Element` in the example). The `EncryptedData` element contains: the `EncryptionMethod` element, which keeps track of the algorithm used to encrypt the data object; the `KeyInfo` element, which carries information about the key used to encrypt the data; and the `CipherData` element, which in turn has a subelement, namely `CipherValue`, containing the encrypted value.

6.2.3 XML Key Management Specification (XKMS)

XML signature and XML encryption specifications provide mechanisms to sign and encrypt XML documents in critical e-services scenario and they involve the use of cryptographic keys. The need to integrate public key infrastructure (PKI) [14, 15] and digital certificates with XML-based applications arises and a W3C working group has been developing an open specification named XML key management specification (XKMS) [13, 16].

XKMS specifies a protocol for distributing and registering public keys, used together with XML Signature and XML Encryption. The main goal of

XKMS is to allow the development of XML-based trust services managing PKI-based cryptographic keys. XKMS is also aimed at reducing the complexity of PKI technology by simplifying the addition of security mechanisms in applications and relying on a trusted third party for all the activities related to PKI tasks.

In particular, XKMS specifies protocols for registering, distributing, and processing public keys that are fully integrable with XML signature and XML encryption. At a high level, the protocol defines a set of predefined services, a set of message formats, communication protocols bindings, processing rules, error models, and responsibilities. XKMS is composed of two major components described below.

XML Key Information Service Specification (X-KISS)

X-KISS defines a protocol that manages public key information providing two services, *locate* and *validate*, used to process and validate public keys, respectively. More precisely, X-KISS is the protocol that provides support for processing the `ds:KeyInfo` element used by both XML signature and XML encryption. Relying on the X-KISS service, the application is not involved in all the activities requiring an interaction with the public key infrastructure, which could require some knowledge about specific standards such as X.509, Simple PKI, and so forth. X-KISS allows the definition of the information that gives to the verifier suggestions on the public key to use. X-KISS is defined as a three-layer service: with the *tier-0* service the processing of element `ds:KeyInfo` is by the applications; with the *tier-1* service the processing of element `ds:KeyInfo` is delegated to a service; with the *tier-2* the processing of element `ds:KeyInfo` is delegated to a service that can also provide additional information on the data specified in the `ds:KeyInfo` element.

XML Key Registration Service Specification (X-KRSS)

X-KRSS defines a protocol that accepts the registration of public key information and is responsible for the entire key lifecycle management. In particular, X-KRSS supports the following four operations, involved in the management of public keys and provided by a registration service. The *registration* operation allows every entity to register a particular public key, binding it to some information. The generation of a public key could be performed by both a client or the registration server. The registration service can require the client to provide additional information to authenticate the request and if the client has generated the ⟨public, private⟩ key pair itself, the service could require the client to provide a proof of possession of the corresponding private key. The *revocation* operation allows every entity to revoke a previously issued key registration. The *recovering* operation allows every entity to recover the private key associated with a registered public key. Note that the recovering operation could require time and the Registration Service often performs a revoking operation after a recovering request. The *Reissue* operation allows a previously registered key binding to be reissued.

A registration service needs to guarantee the validity of all requests, their authenticity and integrity, and needs to manage proofs of possession of private keys. To this purpose, a registration service sets an authentication policy defining an authentication mechanism that establishes offline a secret with a client.

6.3 XML-Based Access Control Languages

Initially, XML-based access control languages were thought to be only for the protection of resources that were themselves XML files [17, 18, 19, 20]. Recent proposals instead use XML to define languages for expressing protection requirements for any kind of data/resources [3, 12, 21, 22]. Two relevant XML-based access control languages are the extensible access control markup language (XACML) [11, 12] and WS-Policy [22]. Based on WS-Security [23], WS-Policy provides a grammar for expressing Web service policies. XACML is the result of an Organization for the Advancement of Structured Information Standards (OASIS) standardization effort proposing an XML-based language to express and interchange access control policies. XACML is designed to express authorization policies in XML against objects that can themselves be identified in XML. While XACML and WS-Policy share some common characteristics, XACML has the advantage of enjoying an underlying policy model as a basis, resulting in a clean and unambiguous semantics of the language [21]. In the remainder of this section, we illustrate the main features of both XACML and WS-Policy.

6.3.1 XACML

The major functionalities offered by XACML can be summarized as follows.

- *Policy combination.* XACML provides a method for combining policies independently specified. Different entities can then define their policies on the same resource. When an access request on that resource is submitted, the system has to take into consideration all these policies.
- *Combining algorithms.* Since XACML supports the definition of policies independently specified, there is the need for a method for reconciling such a policies when their evaluation is contradictory. XACML supports different combining algorithms, each representing a way of combining multiple decisions into a single decision.
- *Attribute-based restrictions.* XACML supports the definition of policies based on properties (attributes) associated with subjects and resources other than their identities. This allows the definition of powerful policies based on generic properties associated with subjects (e.g., name, address,

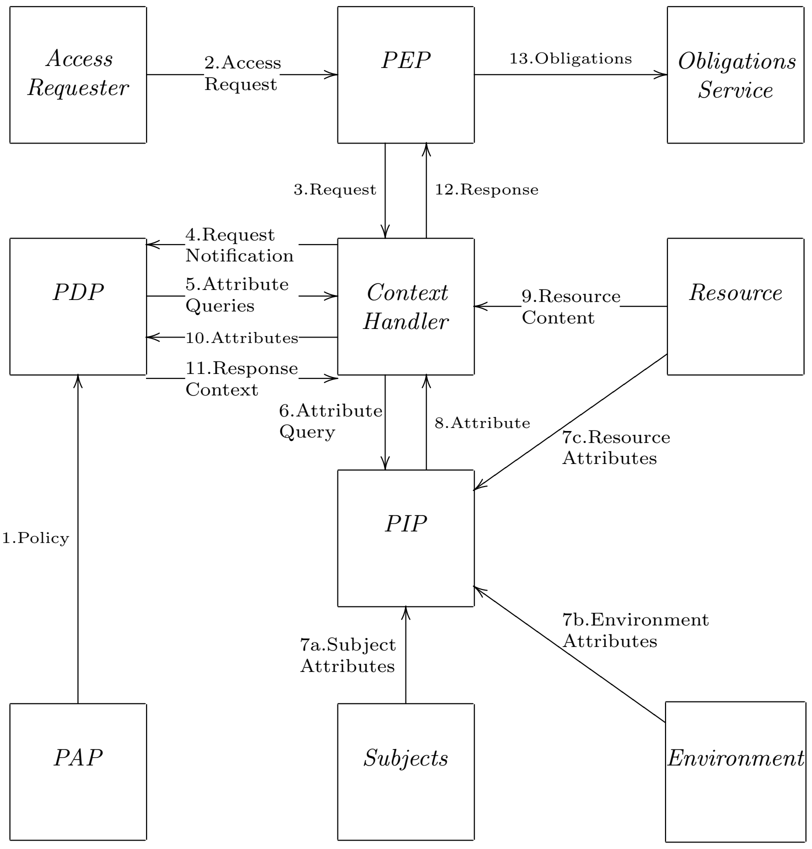

Fig. 6.3. Overview of XACML dataflow [11]

occupation) and resources. XACML includes some built-in operators for comparing attribute values and provides a method of adding nonstandard functions.

- *Multiple subjects.* XACML allows the definition of more than one subject relevant to a decision request.
- *Policy distribution.* Policies can be defined by different parties and enforced at different enforcement points. Also, XACML allows one policy to contain or refer to another.
- *Implementation independence.* XACML provides an abstraction layer that isolates the policy-writer from the implementation details. This means that different implementations should operate in a consistent way, regardless of the implementation itself.
- *Obligations.* XACML provides a method for specifying some actions, called *obligations*, that must be fulfilled in conjunction with the policy enforcement.

A typical scenario involving XACML is when someone wants to perform an action on a resource. For instance, suppose that a physician wants to access a patient's record for inquiry only. The physician would log on to the hospital information system, enter the patient identifier, and retrieve the corresponding record. Data flow through a XACML model can be summarized as follow (see the entities involved and the data flow in Fig. 6.3).

- The requester sends an access request to the *policy evaluation point* (PEP) module, which has to enforce the access decision that will be taken by the policy decision point.
- The PEP module sends the access request to the *context handler* that translates the original request into a canonical format, called *XACML request context*, by querying the *policy information point* (PIP) module. The PIP provides attribute values about the *subject*, *resource*, and *action.* To this purpose, PIP interacts with the *subjects*, *resource*, and *environment* modules. The *environment* module provides a set of attributes that are relevant to take an authorization decision and are independent of a particular *subject*, *resource*, and *action.*
- The context handler sends the XACML request to the *policy decision point* (PDP). The PDP identifies the applicable policies by means of the *policy administration point* (PAP) module and retrieves the required attributes and, possibly, the resource from the context handler.
- The PDP then evaluates the policies and returns the *XACML response context* to the Context Handler. The context handler translates the XACML response context to the native format of the PEP and returns it to the PEP together with an optional set of obligations.
- The PEP fulfils the obligations and, if the access is permitted, it performs the access. Otherwise, the PEP denies access.

As described above, XACML defines a canonical form of the request/response managed by the PDP, allowing policy definition and analysis without taking into account application environment details. Any implementation has to translate the attribute representations in the application environment (e.g., SAML, .NET, Corba [24]) into the XACML context. For instance, an application can provide a SAML [3] message that includes a set of attributes characterizing the subject making the access request. This message has to be converted to the XACML canonical form and, analogously, the XACML decision has then to be converted to the SAML format.

Policy Set, Policy and Rule

XACML relies on a model that provides a *formal* representation of the access control security policy and its working. This modeling phase is essential to ensure a clear and unambiguous language, which could otherwise be subject to different interpretations and uses. The main concepts of interest in the XACML policy language model are *rule*, *policy*, and *policy set*.

An XACML policy has, as root element, either `Policy` or `PolicySet`. A `PolicySet` is a collection of `Policy` or `PolicySet` elements. An XACML policy consists of a *target*, a set of *rules*, an optional set of *obligations*, and a *rule combining algorithm*. A `Target` basically consists of a simplified set of conditions for the *subject*, *resource*, and *action* that must be satisfied for a policy to be applicable to a given request. If all the conditions of a `Target` are satisfied, then its associated `Policy` (or `PolicySet`) applies to the request. If a policy applies to all entities of a given type, that is, all subjects, actions, or resources, an empty element, named `AnySubject`, `AnyAction`, `AnyResource`, respectively, is used. The components of a rule are a *target*, an *effect*, and a *condition*. The target defines the set of resources, subjects, and actions to which the rule is intended to apply. The effect of the rule can be `permit` or `deny`. The condition represents a boolean expression that may further refine the applicability of the rule. Note that the `target` element is an optional element: a rule with no target applies to all possible requests. An `Obligation` specifies an action that has to be performed in conjunction with the enforcement of an authorization decision. For instance, an obligation can state that all accesses to medical data have to be logged. Note that only policies that are evaluated and have returned a response of `permit` or `deny` can return obligations. This means that if a policy evaluates to `indeterminate` or `not applicable`, the associated obligations are not considered. Each `Policy` also defines a *rule combining algorithm* used for reconciling the decisions each rule makes. The final decision value, called *the authorization decision*, inserted in the XACML context by the PDP is the value of the policy as defined by the rule combining algorithm. XACML defines different combining algorithms. The *deny overrides* algorithm states that, if there exists a rule that evaluates to `deny` or if all rules evaluate to `not applicable`, the result is `deny`. If all rules evaluate to `permit`, the result is `permit`. If some rules evaluate to `permit` and some evaluate to `not applicable`, the result is `permit`. The *permit overrides* algorithm states that, if there exists a rule that evaluates to `permit`, the result is `permit`. If all rules evaluate to `not applicable`, the result is `deny`. If some rules evaluate to `deny` and some evaluate to `not applicable`, the result is `deny`. The *first applicable* algorithm states that each rule has to be evaluated in the order in which it appears in the `Policy`. For each rule, if the target matches and the conditions evaluate to true, the result is the effect (`permit` or `deny`) of such a rule. The *only-one-applicable* algorithm states that, if more than one rule applies, the result is `indeterminate`. If no rule applies, the result is `not applicable`. If only one policy applies, the result coincides with the result of evaluating that rule. According to the selected combining algorithm, the authorization decision returned to the PEP can be `permit`, `deny`, `not applicable` (when no applicable policies or rules could be found), or `indeterminate` (when some errors occurred during the access control process).

An important feature of XACML is that a rule is based on the definition of attributes corresponding to specific characteristics of a subject, resource, action, or environment. For instance, a physician at a hospital may have the attribute of being a researcher, a specialist in some field, or many other job roles. According to these attributes, the physician can be able to perform different functions within the hospital. Attributes are identified by the `SubjectAttributeDesignator`, `ResourceAttributeDesignator`, `ActionAttributeDesignator`, and `EnvironmentAttributeDesignator` elements. These elements use the `AttributeValue` element to define the requested value of a particular attribute. Alternatively, the `AttributeSelector` element can be used to specify where to retrieve a particular attribute. Note that both the attribute designator and `AttributeSelector` elements can return multiple values. For this reason, XACML provides an attribute type called *bag*, an unordered collection that can contain duplicates values for a particular attribute. In addition, XACML defines other standard value types such as string, boolean, integer, time, and so on. Together with these attribute types, XACML also defines operations to be performed on the different types such as equality operation, comparison operation, string manipulation, and so on.

As an example of XACML policy, suppose that a hospital defines a high-level policy stating that "any user with role `head physician` can read the `patient record` for which she is designated as head physician". Figure 6.4 illustrates the XACML policy corresponding to this high-level policy. The policy applies to requests on the `http://www.example.com/hospital/patient.xsd` resource. The policy has one rule with a target that requires a `read` action, a subject with role `head physician` and a condition that applies only if the subject is the head physician of the requested patient. For more details about roles and role-based access control (RBAC) see Chap. 5.

XACML Request and Response

XACML also defines a standard format for expressing requests and responses. The original request submitted by the PEP is then translated through the context handler in a canonical form, then forwarded to the PDP to be evaluated. Such a request contains attributes for the subject, resource, action, and, optionally, for the environment. Each request includes exactly one set of attributes for the resource and action and at most one set of environment attributes. There may be multiple sets of subject attributes each of which is identified by a category URI.

A response element contains one or more results corresponding to an evaluation. Each result contains three elements, namely `Decision`, `Status`, and `Obligations`. The `Decision` element specifies the authorization decision (i.e., `permit`, `deny`, `indeterminate`, `not applicable`), the `Status` element indicates if some error occurred during the evaluation process, and the optional `Obligations` element states the obligations that the PEP must fulfil. For instance, suppose that a user, belonging to role `head physician` and with ID `354850273` wants to read resource `www.example.com/hospital/patient.xsd`

```
<Policy PolicyId="Pol1"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
  rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects> <AnySubject/> </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:stringmatch">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            http://www.example.com/hospital/patient.xsd
          </AttributeValue>
          <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions> <AnyAction/> </Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              head physician
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId= "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources> <AnyResource/> </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              read
            </AttributeValue>
            <ActionAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:head-physicianID"/>
      <AttributeSelector RequestContextPath="/ctx:Request/ctx:Resource/ctx:
        ResourceContent/hospital:record/hospital:patient/hospital:
        patient-head-physicianID"
        DataType="http://www.w3.org/2001/XMLSchema#string"/ >
    </Condition>
  </Rule>
</Policy>
```

Fig. 6.4. An example XACML policy

with patient ID equal to `123a45d`. This request is compared with the XACML policy in Fig. 6.4. The result of this evaluation is that the user is allowed (`permit`) to access the requested patient record.

6.3.2 WS-Policy

Web service policy framework (WS-Policy) provides a generic model and a flexible and extensible grammar for describing and communicating the policies of a Web service [22]. The WS-Policy includes a set of general messaging related assertions defined in WS-PolicyAssertions [25] and a set of security policy assertions related to supporting the WS-Security specification defined in WS-SecurityPolicy [26]. In addition, WS-PolicyAttachment [27] defines how to attach these policies to Web services or other subjects such as service locators. A WS-Policy is a collection of one or more *policy assertions* that represent an individual preference, requirement, capability, or other properties that have to be satisfied to access the *policy subject* associated with the assertion. The XML representation of a policy assertion is called *a policy expression.*[1] Element `wsp:Policy` is the container for a policy expression. Policy assertions are typed and can be *simple* or *complex.* A simple policy can be compared to other assertions of the same type without any special consideration about the semantics' assertion. A complex policy requires comparison by means of type-specific assertions. The assertion type can be defined in such a way that the assertion is parameterized. For instance, an assertion describing the maximum acceptable password size (number of characters) would likely accept an integer parameter indicating the maximum character count. In contrast, an assertion that simply indicates that a password is required does not need parameters; its presence is enough to convey the assertion. Every policy assertion could be defined `optional`. WS-Policy provides an element, called `wsp:PolicyReference`, that can be used for sharing policy expressions between different policies. Conceptually, when there is a reference, it is replaced by the content of the referenced policy expression. WS-Policy also provides two operators, namely `wsp:All` and `wsp:ExactlyOne`, that can be used for combining policy assertions. The first operator requires that all of its child elements be satisfied; the second operator requires that exactly one of its child elements be satisfied. In case no operator is specified, the `wsp:All` operator is taken as default.

Figure 6.5(a) illustrates a simple example of policy stating that the access is granted if exactly one security token among the following is provided: a Kerberos certificate and a UsernameToken with Username `Bob`; an X509 certificate and a UsernameToken with Username `Bob`; an X509 certificate and a UsernameToken with Username `Alice`. The third option corresponds to the referred policy, called `opts`, illustrated in Fig. 6.5(b).

[1] Note that using XML to represent policies facilitates interoperability between heterogeneous platforms and Web service infrastructures.

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
 <wsp:ExactlyOne>
  <wsp:All>
   <wsse:SecurityToken>
    <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
   </wsse:SecurityToken>
   <wsse:SecurityToken>
    <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
    <wsse:Username>Bob</wsse:Username>
   </wsse:SecurityToken>
  </wsp:All>
  <wsp:All>
   <wsse:SecurityToken>
    <wsse:TokenType>wsse:X509v3</wsse:TokenType>
   </wsse:SecurityToken>
   <wsse:SecurityToken>
    <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
    <wsse:Username>Bob</wsse:Username>
   </wsse:SecurityToken>
  </wsp:All>
  <wsp:PolicyReference URI="#opts" />
 </wsp:ExactlyOne>
</wsp:Policy>
```
(a)

```
<wsp:Policy xmlns:wsse="..."
  xmlns:ns="...">
  <wsp:All wsu:Id="opts">
    <wsse:SecurityToken>
      <wsse:TokenType>
        wsse:X509v3
      </wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken>
      <wsse:TokenType>
        wsse:UsernameToken
      </wsse:TokenType>
      <wsse:Username>
        Alice
      </wsse:Username>
    </wsse:SecurityToken>
  </wsp:All>
</wsp:Policy>
```
(b)

Fig. 6.5. A simple example of WS-Policy (a) and the corresponding referred WS-Policy (b)

6.4 Conclusions

In this chapter we introduced the most important XML security technologies. We described two important initiatives, namely XML signature and XML encryption, facing the problem of protecting information distributed on the Internet. We then briefly reviewed the XML key management specification, which provides facilities for the management of public keys used together with XML signature and XML encryption. We concluded the chapter with the description of two XML-based access control languages, namely XACML and WS-Policy, discussing their peculiarities and their principal features.

6.5 Acknowledgements

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

References

1. Apache XML Project. http://xml.apache.org/.
2. N. Bradley (2002). The XML Companion. Addison Wesley, 3rd edition.

3. OASIS Security Services TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
4. E. Newcomer (2002). Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison Wesley.
5. P. Samarati, S. De Capitani di Vimercati (2001). Access control: Policies, models, and mechanisms. In Focardi R, Gorrieri R, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag.
6. XML Encryption Syntax and Processing, W3C Recommendation (2002). http://www.w3.org/TR/xmlenc-core/.
7. XML-Signature Syntax and Processing, W3C Recommendation (2002). http://www.w3.org/TR/xmldsig-core/.
8. D. Box et al. (2000). Simple Object Access Protocol (SOAP) version 1.1. http://www.w3.org/TR/SOAP.
9. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati (2002). Securing SOAP E-services. International Journal of Information Security (IJIS), 1(2):100–115.
10. E. Damiani, S. De Capitani di Vimercati, P. Samarati (2002). Towards securing XML web services. In Proc. of the 2002 ACM Workshop on XML Security, Washington, DC, USA.
11. OASIS eXtensible Access Control Markup Language TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
12. T. Moses (2005). eXtensible Access Control Markup Language (XACML) version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.
13. B. Galbraith, W. Hankinson, A. Hiotis, M. Janakiraman, D.V. Prasad, R. Trivedi, D. Whitney (2002). Professional Web Services Security. Wrox Press.
14. A. Arsenault, S. Turner (2002). Internet X.509 Public Key Infrastructure: Roadmap. Internet Draft, Internet Engineering Task Force.
15. A. Essiari, S. Mudumbai, M.R. Thompson (2003). Certificate-Based Authorization Policy in a PKI Environment. ACM Transactions on Information and System Security, 6(4):566–588.
16. W. Ford et al (2001). XML Key Management Specification (XKMS), W3C Note. http://www.w3.org/TR/xkms/.
17. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati (2000). Securing XML documents. In Proc. of the 2000 International Conference on Extending Database Technology (EDBT2000), Konstanz, Germany.
18. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati (2002). A fine-grained access control system for XML documents. ACM Transactions on Information and System Security (TISSEC), 5(2):169–202.
19. A. Gabillon (2004). An authorization model for XML databases. In Proc. of the ACM Workshop Secure Web Services, George Mason University, Fairfax, VA, USA.
20. A. Gabillon, E. Bruno (2001). Regulating access to XML documents. In Proc. of the Fifteenth Annual IFIP WG 11.3 Working Conference on Database Security, Niagara on the Lake, Ontario, Canada.
21. C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, P. Samarati (2004). XML-based access control languages. Information Security Technical Report.
22. S. Bajaj et al (2004). Web Services Policy Framework (WS-Policy). http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp.

23. B. Atkinson, G. Della-Libera et all (2002). Web services security (WS-Security). http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-security.asp.
24. Object Management Group. The CORBA Security Service Specification. ftp://ftp.omg.org/pub/docs/ptc.
25. D. Box et al. (2003). Web Services Policy Assertions Language (WS-PolicyAssertions) version 1.1. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassertions.asp.
26. G. Della-Libera et al (2005). Web Services Security Policy Language (WS-SecurityPolicy). http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-securitypolicy.pdf.
27. S. Bajaj et al. (2006). Web Services Policy Attachment (WS-PolicyAttachment) version 1.2. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyattachment.asp.