Towards Securing XML Web Services

Ernesto Damiani
DTI - Università di Milano
26013 Crema - Italy
damiani@dti.unimi.it

Sabrina De Capitani di Vimercati
DEA - Università di Brescia
25123 Brescia - Italy
decapita@ing.unibs.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

ABSTRACT

Security is currently one of the main concerns about XML Web services. Several initiatives are currently ongoing aimed at achieving a standardized way for supporting integrity, confidentiality, and access control for XML Web services. This paper looks into these approaches and gives some hints for future research.

1. INTRODUCTION

An XML Web service is a Web-based application that accepts requests from different systems across the Internet (or an Intranet) through the application of Web technology standards including XML [5], SOAP [4], WSDL [6], and HTTP [12]. While XML Web services are an increasingly successful paradigm for the development of complex Web-based applications, the original specifications of their underlying technologies did not even mention security. It is therefore easy to understand why security is currently one of the biggest concerns about future development XML Web services. Specifically, two main issues need to be addressed:

- restricting access to a XML Web service to authorized users;
- protecting the integrity and confidentiality of XML messages exchanged in a Web service environment.

At first sight, it may seem that both these issues can be addressed straightforwardly by relying on the security techniques already used for Web sites. In other words, one might assume that standard techniques for Web application and HTTP security can be applied alone or in combination to create secure XML Web services. Indeed, this approach is being taken by a number of current projects and commercial products based on XML Web services. Specifically, two currently available and well-known techniques are used: *Secure Sockets Layer* (SSL) [11] and *firewall-based* rules. The latter technique is mainly used on private networks: assuming that the computers that need to access an XML Web service are known in advance (as it often happens, for example, on a corporate Intranet), firewalls can accept only connections coming from computers whose IP addresses are known in advance. On public networks such as the Internet, where clients' addresses are unknown to servers before they connect, a variety of *Secure Sockets Layer* (SSL) implementations can be used to encrypt and decrypt messages exchanged between clients and servers. Besides protecting messages from unauthorized read while they are in transit, SSL software verifies that incoming messages actually come from the correct sender.[1] While these two techniques have proven to be robust and very effective, their use in Web service environments is not considered satisfying by the research community, specially in association with XML messages used in SOAP-based communications. Indeed, the*Simple Object Access Protocol* (SOAP) is a specification for performing method requests as XML documents and was conceived as a message format not tied to a single protocol: therefore, at least in principle, one should avoid relying on a specific underlying protocol (such as HTTP or even TCP/IP) to perform authentication for (and access control) to it.

Several proposals for securing Web services have been made in the last two years, some of them generically aimed at protecting XML integrity and privacy [3, 13], others specifically at passing authentication credentials with SOAP calls, such as [9] and [2]. In the following we shall compare and discuss these approaches.

2. W3C XML-SIGNATURE SYNTAX AND PROCESSING

The W3C and IETF have proposed since 1999 a standard for encrypting XML data and tags within a document. This allows for encrypting portions of an XML document at the granularity of XML subtrees and even single elements. Encrypting portions of a document with different keys permits to distribute the same XML document to multiple recipients, each being able to decrypt only the parts relevant to it. Also, the standard includes a mechanism for creating and verifying digital signatures for all or part of a generic XML message, and for holding a digital signature within XML documents. As an example, consider the XML message in Figure 1(a), broadcasted by the `ACME` company's procurement application to `ACME` suppliers. The message requests a quotation about a product from a number of suppliers.

[1]The server, the client, or both parties can also use *digital certificates* as part of the authentication process.

Obviously, it is crucial that ACME suppliers receiving this message are able to determine the identity of the sender. Using the W3C XML-signature syntax [3], the XML message broadcasted by the ACME company's procurement application can be signed simply by adding a header. Namely, the original QuotationRequest element will become a child of a new root element (e.g., signedQuotationRequest), and a sibling (a Signature element) will be added that holds ACME digital signature information, as well as the algorithm used to canonicalize the data, any transforms performed on it, the digest algorithm, and the signature algorithm itself. In our sample case, the signed document looks as illustrated in Figure 1(b).

Simple as it may seem, the W3C proposal had to tackle and solve a number of subtle problems. For instance, signing a document that includes the signature itself can be tricky, as the document's digest is also computed as part of the signing process and will change when the actual signature value is inserted into the SignatureValue node. The W3C specification solves this problem using a transformation to remove the entire Signature element from the data to be digested and signed.

A similar approach can be adopted for protecting XML messages while in transit by encrypting their contents, so that only the authorized recipient can read them. Suppose, for example, that the procurement application, having chosen the offer of one of ACME suppliers, sends an order message including payment information (see Figure 2(a)). Using the W3C's XML Encryption Syntax and Processing standard [13], the XML order message can be encrypted as illustrated in Figure 2(b). In this example, ACME credit card number and payment amount have been replaced with an EncryptedData node, in turn including a CipherValue subelement for the encrypted data. The EncryptedKey node contains a copy of the session key encrypted with the supplier's Public Key. Upon reception of the order, the supplier can then use her private key to decrypt the session key; with it, she can safely decrypt the payment data.

Of course, the two techniques highlighted above could be combined, digitally signing and encrypting the SendOrder message.

The techniques do indeed a good job in providing the general functionalities for application-level XML security. However, they were conceived for securing generic well-formed XML documents, and they do not rely on, neither take advantage of the schema of the documents they encrypt. In other words, they are not specifically aimed to XML-based protocols and to the SOAP messages structure.

3. SOAP HIGHLIGHTS

SOAP requests carry remote method invocations over HTTP. They are fully declarative, inasmuch they do not dictate how the target component should handle the request. The overall structure of a SOAP invocation is depicted in Figure 3; the outside *Multimedia Internet Mail Extension (MIME)* layer refers to the type of the message as reported in the HTTP header, namely text/xml.

The SOAP XML payload contains an encoded method invocation; its lexicon is defined by a standard XML namespace SOAP-ENV. In SOAP, the XML payload is used mainly to encode parameters' datatypes in a platform independent way, much as CORBA's *Common Data Representation*. The XML payload includes a root Envelope element and a child

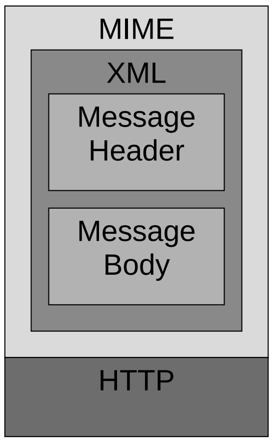

Figure 3: Overall structure of a SOAP invocation

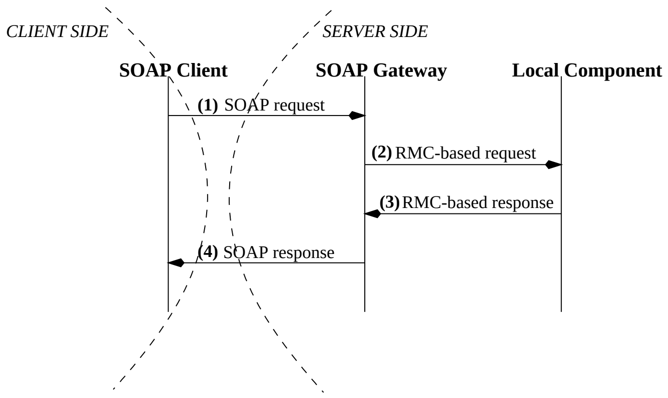

Figure 4: Execution sequence of a SOAP call

Body element, the latter having an optional Header sibling. The SOAP payload's root element Envelope provides the serialization context for the method calls that follow. The Envelope element can contain additional attributes (qualified by a suitable XML namespace). The SOAP Header element contains auxiliary information (called *header entries*) not functionally related to the method invocation, such as transaction management and payment. SOAP headers may contain the standard Actor and MustUnderstand attributes (as well as other optional, namespace-qualified ones), respectively stating the URI of the final destination of the message and whether header processing capability on the part of the recipient is mandatory (1) or not (0).

A SOAP response is similar to a request, apart from the fact that it adds a Response suffix to the element name used for the method. For instance, for the method QuotationRequest, the response element is QuotationRequestResponse. Figure 4 summarizes the phases of a SOAP invocation.

4. THE ROLE OF HEADERS IN SOAP SECURITY

Inside the *Envelope* SOAP message is broken up into two portions: the SOAP *header* and *body* [4]. The header is used to hold metadata associated with the request, while

```
<proc:QuotationRequest xmlns:proc="http://www.acme.com/Procurement">
  <proc:ProductType code='AC350'/>
  <proc:quantity>100,000</proc:quantity>
</proc:QuotationRequest>
```

(a)

```
<proc:signedQuotationRequest xmlns:proc="http://www.acme.com/Procurement">
  <Signature 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#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>......</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>......</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>......</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
  <proc:QuotationRequest xmlns:proc="http://www.acme.com/Procurement">
    <proc:ProductType code='AC350'/>
    <proc:quantity>100,000</proc:quantity>
  </proc:QuotationRequest>
</proc:signedQuotationRequest>
```

(b)

Figure 1: An example of XML message (a) and the corresponding signed message (b)

the body is used to hold the service invocation and its parameters. In the approach proposed in [9], authentication information is seen as metadata sent together with a message. Furthermore, digital signatures can be hosted in this field making SOAP headers play an important role in SOAP security.[2]

4.1 Using Headers for Credential Transfer and Access Control

A first proposal exploiting SOAP's headers for carrying credentials and enforcing access control was presented by Damiani et al. in [9]. The authors exploit the features of the SOAP protocol, and represent all the information characterizing the subject of a request by means of a custom header included in each SOAP call. The motivation for this approach is that custom headers provide a uniform description framework for service requests and subjects submitting them, and represent an elegant and model-neutral way to specify security arrangements. The proposed custom header has a root `subject` with three types of children: `user`, `location`, and `role`; the first of which is mandatory (systems non requesting user authentication will have a dummy `Anonymous` identifier and a dummy password hash for this element) and the remaining two are optional. Also, the latter can have multiple occurrences. Element `user` contains the identity (element `userid` and the hashed password `passwdhash`) of the user on behalf of whom the request is submitted. Element `location` defines the network address of the machine from which the request originates and can be symbolic (`symname`) or numeric (`netaddr`). Each element `role` carries a certificate enabling the user to exercise the stated role. While the authors acknowledge that different systems may use different representations for certificates, in their presentation they assume certificates enabling a role to be characterized by four elements: `roleid`, `issuer`, `holder`, and `validity`. Element `roleid` states the identifier of the role enabled by the certificate. Element `issuer` specifies the authority that released the certificate. Element `holder` defines to whom the certificate is referred (and may refer to its identity or key). Finally, element `validity` imposes constraints on the time in which the certificate is to be considered valid. Figure 5 illustrates an example of SOAP header in [9].

Certificates and subject's information carried in headers is exploited in [9] where every request directed to the *SOAP gateway*[3] is intercepted and evaluated against authorizations specifying restrictions to service accessibility. Based on the authorizations, the request may: be rejected; be allowed as is; or be filtered and executed in a modified form, where filtering of a request may involve elimination of some of its parameters that the current invoker is not allowed to specify. Once filtered, requests are passed to the SOAP gateway, which will produce a response to be returned to the client. The response also is sent through the access control system

[2]There is of course an exception: if encryption is used, encrypted data will typically live in the SOAP body. However, session keys can still be included in the SOAP headers.

[3]The SOAP gateway is the component that translates the SOAP request to a call to a local or remote server; the answer of the server is then translated back to the format of the SOAP response. The communication between the client and the gateway uses the HTTP protocol.

```
<proc:SendOrder xmlns:proc="http://www.acme.com/Procurement">
  <proc:OrderID>64B4A0D1-814E-4FF6-918A-DD7E7E1AECEA</proc:OrderID>
  <proc:ProductType code='AC350'/>
  <proc:quantity>100,000</proc:quantity>
  <cc:paymentInfo xmlns:cc= "http://www.creditcardcompany/Charge">
    <cc:Amount currency='Euro'>2000</cc:Amount>
    <cc:CardNumber>1234123412341234</cc:CardNumber>
    <cc:Expiry>08-15-2002</cc:Expiry>
  </cc:paymentInfo>
</proc:SendOrder>
```

(a)

```
<proc:SendOrder xmlns:proc="http://http://www.acme.com/Procurement">
  <proc:OrderID>64B4A0D1-814E-4FF6-918A-DD7E7E1AECEA</proc:OrderID>
  <proc:ProductType code='AC350'/>
  <proc:quantity>100,000</proc:quantity>
  <cc:paymentInfo xmlns:cc= "http://www.creditcardcompany/Charge">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" ID="ED"
      Type="http://www.w3.org/2001/04/xmlenc#Content">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:RetrievalMethod URI="#SessKey" Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
      </ds:KeyInfo>
      <CipherData>
        <CipherValue>......</CipherValue>
      </CipherData>
    </EncryptedData>
    <EncryptedKey Id="SessKey" xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>Supplier's Public/Private Key</ds:KeyName>
      </ds:KeyInfo>
      <CipherData>
        <CipherValue>......</CipherValue>
      </CipherData>
      <ReferenceList>
        <DataReference URI="#ED"/>
      </ReferenceList>
    </EncryptedKey>
    <cc:Expiry>08-15-2002</cc:Expiry>
  </cc:paymentInfo>
</proc:SendOrder>
```

(b)

Figure 2: An XML order message including payment information (a) and the corresponding encrypted message (b)

and may be subject to some filtering.

4.2 Using Headers for Credential Transfer

Microsoft Web Services Security Language [2] develops on this idea identifying three areas: *Credential Transfer*, *Message Integrity*, and *Message Privacy*. While in [9] the header contained an extension of an XML-SPKI certificate; in Microsoft's proposal, the credentials element passed in the SOAP headers is used to carry standard X.509 certificates and Kerberos tickets.

In both cases, while it is possible to refer to traditional approaches assuming an encrypted username and password, the credentials can be used to support complex RBAC schemata, holding any number of certificates[4]. Figure 6 illustrates an example of Microsoft header, where the credentials node holds a public certificate.

[4]Note that certificates can be used for any number of reasons beyond just authenticating the sender of the message. For instance, they can specify the Certificate Authority guaranteeing a particular certificate.

Both [2] and [9] guarantee the integrity of the SOAP message in the sense the supplier will know that the SOAP message comes from an authorized source. However, using headers to carry credentials and certificates is not in itself sufficient to create a security infrastructure for XML Web services. Indeed, such infrastructure needs to include support for *Policy Administration*, that is, the possibility of writing policies that exploit information carried in the headers for subject and environment specification. Also, the infrastructure needs to address *Policy Evaluation*, that is, processing a policy when a certain SOAP message is received, and *Policy Decision*, that is, the kind of notification that the sender must receive about the access decision. A preliminary assessment of these problems was given in in [9]; in the next section we will briefly discuss recent research and standardization approaches to this issue.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-SBJ = "http://www.xmlsec.org/subject">
  <SOAP:Header>
    <SOAP-SBJ:subject>
      <SOAP-SBJ:user>
        <SOAP-SBJ:userid>Alice</SOAP-SBJ:userid>
        <SOAP-SBJ:passwdhash SOAP-SBJ:hash-alg="sha1">xxj31ZMTZzkVA</SOAP-SBJ:passwdhash>
      </SOAP-SBJ:user>
      <SOAP-SBJ:role>
        <SOAP-SBJ:roleid>ACME_Client</SOAP-SBJ:roleid>
        <SOAP-SBJ:issuer>
          <SOAP-SBJ:public-key>...</SOAP-SBJ:public-key>
        </SOAP-SBJ:issuer>
        <SOAP-SBJ:holder>
          <SOAP-SBJ:name>Alice</SOAP-SBJ:name>
        </SOAP-SBJ:holder>
        <SOAP-SBJ:validity>
          <SOAP-SBJ:notbefore>2002-09-22_12:00:00</SOAP-SBJ:notbefore>
          <SOAP-SBJ:notafter>2002-11-22_24:00:00</SOAP-SBJ:notafter>
        </SOAP-SBJ:validity>
      </SOAP-SBJ:role>
    </SOAP-SBJ:subject>
  </SOAP:Header>
  <SOAP:Body>
    <proc:SendOrder xmlns:proc="http://www.acme.com/Procurement">
      <proc:ProductType code='AC350'/>
      <proc:quantity>100,000</proc:quantity>
      <cc:paymentInfo xmlns:cc= "http://www.creditcardcompany/Charge">
        <cc:Amount currency='Euro'>2000</cc:Amount>
        <cc:CardNumber>12341234123412341234</cc:CardNumber>
        <cc:Expiry>12-15-2002</cc:Expiry>
      </cc:paymentInfo>
    </proc:SendOrder>
  </SOAP:Body>
</SOAP:Envelope>
```

Figure 5: An example of header in [9]

5. USING HEADERS IN CONNECTION WITH XML-BASED ACCESS CONTROL LANGUAGES

Header-based authentication is of course only the first step: in the Policy Evaluation phase it must be possible to reference information contained in the headers, as well as other environment metadata in a suitable policy language. Starting from early research proposal such as [7, 8] and [14], standards are underway allowing to refer to various types of XML-based metadata from inside policies. Their application envisions a complete infrastructure for XML Web services access control following the line of [9] and [14].

- **Security assertion markup language (SAML) [1].** SAML handles the actual exchange of authentication and authorization requests and responses. An SAML request is sent, via SOAP over HTTP, to a system with the appropriate means for processing the request. A SAML request plays much the same role of a SOAP security header: it contains information such as authentication username and password, or other details about the individual making the request. SAML uses an "assertion schema" to determine who the requesting agent (or user) is, what it is requesting, and whether or not their request has been granted.
- **eXtensible access control markup language (XACML) [10].** XACML is a specification from Oasis that is used in conjunction with SAML (explained below), and it provides a means for standardizing access control decisions for XML documents. XACML is used to define whether to permit requested access to a resource. XACML uses a *context*, that can easily mapped on SAML requests, in order to determine if access should be granted to a resource based on rule-sets, or policies. Once the policy is evaluated and returns a true or false value to indicate whether or not access is granted, an SAML authorization decision assertion is returned, which is then processed accordingly.

6. CONCLUSION

Thanks to work done in the XML signature and encryption standards, security for SOAP messaging (as opposed to protocol-level security) is being tackled successfully. However, some problems related to establishing a global security infrastructure for XML Web services are still to be solved, especially concerning the coexistence and relations among a number of different, though related, standard proposals, none of which has been fully realized and adopted yet. This paper attempts to start a discussion on the different proposals and how they can coexist and be exploited for providing a secure infrastructure to XML Web services.

7. REFERENCES

[1] Advancing SAML, an XML-based security standard for exchanging authentication and authorization

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
    <m:path xmlns:m="http://schemas.xmlsoap.org/rp">
      <m:action>http://supplier.com/orders</m:action>
      <m:to>soap://supplier.com/orders</m:to>
      <m:id>uuid:......</m:id>
    </m:path>
    <wsse:credentials xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="SigningCertificate">
        <ds:X509Data><ds:X509Certificate>......</ds:X509Certificate></ds:X509Data>
      </ds:KeyInfo>
    </wsse:credentials>
    <wsse:integrity xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/xml-exc-c14n#"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="">
            <ds:Transforms>
              <ds:Transform Algorithm="http://schemas.xmlsoap.org/2001/10/security#RoutingSignatureTransform"/>
              <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>aiYECAxNqK2PivQaRweWajXup5zJa...</ds:SignatureValue>
        <ds:KeyInfo><wsse:licenseLocation>#SigningCertificate</wsse:licenseLocation></ds:KeyInfo>
      </ds:Signature>
    </wsse:integrity>
  </SOAP:Header>
  <SOAP:Body>
    <proc:SendOrder xmlns:proc="http://www.acme.com/Procurement">
      <proc:ProductType code='AC350'/>
      <proc:quantity>100,000</proc:quantity>
      <cc:paymentInfo xmlns:cc= "http://www.creditcardcompany/Charge">
        <cc:Amount currency='Euro'>2000</cc:Amount>
        <cc:CardNumber>1234123412341234</cc:CardNumber>
        <cc:Expiry>12-15-2002</cc:Expiry>
      </cc:paymentInfo>
    </proc:SendOrder>
  </SOAP:Body>
</SOAP:Envelope>
```

Figure 6: An example of Microsoft header

information. http://www.oasis-open.org/committees/security/.

[2] B. Atkinson and et al. Web services security (ws-security), April 2002. http://msdn.microsoft.com/ws/2002/04/Security.

[3] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. *XML-Signature Syntax and Processing*. World Wide Web Consortium (W3C), February 2002. http://www.w3.org/TR/xmldsig-core.

[4] D. Box. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web Consortium (W3C), May 2000. http://www.w3.org/TR/SOAP.

[5] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. World Wide Web Consortium (W3C), October 2000. http://www.w3.org/TR/REC-xml.

[6] R. Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana. *Web Services Description Language (WSDL) Version 1.2*. World Wide Web Consortium (W3C), July 2002. http://www.w3.org/TR/wsdl12.

[7] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Controlling access to XML documents. *IEEE Internet Computing*, 5(6):18–28, November/December 2001.

[8] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5(2):169–202, May 2002.

[9] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing SOAP e-services. *International Journal of Information Security (IJIS)*, 1(2):100–115, February 2002.

[10] Defining XACML, an XML specification for expressing policies for information access over the internet. http://www.oasis-open.org/committees/xacml.

[11] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL Protocol - Version 3.0, March 1996. http://ftp.nectec.or.th/CIE/Topics/ssl-draft/INDEX.HTM.

[12] J. Gettys, J. Mogul, H. Frystyk, L. Masinter,

P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[13] T. Imamura, B. Dillaway, and E. Simon. *XML Encryption Syntax and Processing*. World Wide Web Consortium (W3C), August 2002. http://www.w3.org/TR/xmlenc-core.

[14] M. Kudo and S. Hada. XML Document Security and e-Business applications. In *Proc. of the 7th ACM Conference on Computer and Communication Security*, Athens, Greece, November 2000.