XML Digital Signature System Independent of Existing Applications

Toshiro Takase and Naohiko Uramoto
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa, Japan
{E30809|uramoto}@jp.ibm.com

Kunimori Baba
Yamato Software Laboratory, IBM Japan
1623-14, Shimotsuruma, Yamato-shi, Kanagawa, Japan
kbaba@jp.ibm.com

Abstract

This paper describes a turnkey solution to add a XML digital signature capability without modifying existing XML-based B2B systems. The signature proxy between applications watches for XML messages exchanged on the network. Outbound messages are received by the proxy and automatically signed and by a signature server implemented as a Web service. Inbound messages are also verified by using the proxy and the signature server. The existing applications do not care about handling of digital signatures. The signature server can also provide (1) content-based key selection and (2) logging of signed documents with fine-grain access control. The system introduced in this paper is called the XML Security Services Suite (XS-Cube), a set of security-related Web services including digital signatures.

1. Introduction

Today, the Internet is widely used in business-to-business (B2B) transactions and security is one of the most important issues. SSL is widely used for transport-level security but is inadequate to assure non-repudiation, since it lacks digital signatures.

On April 1st, 2001, "The Digital Signature Law" [4] went into effect in Japan. In the United States, a similar law has been in effective on June 2000 in the United Status. This law states that a document which has a proper digital signature has the same legal status as a paper document with a signature. A digitally signed document should be preserved in an appropriate way with an audit trail equivalent to a paper document. Transport-level encryption or a transport-level signature is inadequate, but a digital signature within the document itself has become important to provide a "paperless" environment that dramatically reduces companies' costs.

XML has become widely used as a first class format for data description and exchange in B2B systems. Compared to HTML, XML can describe business data with complex data structures and data types. Therefore, digital signatures for XML documents are receiving more attention. Currently, standardization activities for XML digital signatures are proceeding at W3C and IETF. At W3C, the specification for XML signatures was published as Proposed Recommendation [3] and at IETF, as RFC 3075.

We assume that a B2B system exchanges business data as XML documents (Figure 1). When this system receives an XML document such as purchase orders, this system processes the information in a back-end application and returns a result in XML format as confirmation. Figure 1 shows a typical configuration of an XML-based B2B system.

If XML digital signatures are required, a simple solution is to modify the system itself. When confirmation XML documents are generated, a digital signature library is called and the library signs the XML document. Although this is a typical solution, the existing system must be modified, and there are risks that the modified part contains a security hole.

In this paper, we introduce a turnkey solution to sign XML documents without modifying the existing applications. This system consists of a signature proxy and a signature server. The signature proxy is placed between the applications that exchange XML documents. The signature proxy watches for XML documents on the network and the signature proxy sends the document to the signature server or verification server according to the content of the XML document. Because the signature and verification processes are done through the proxy, the actual applications do not have to be modified.

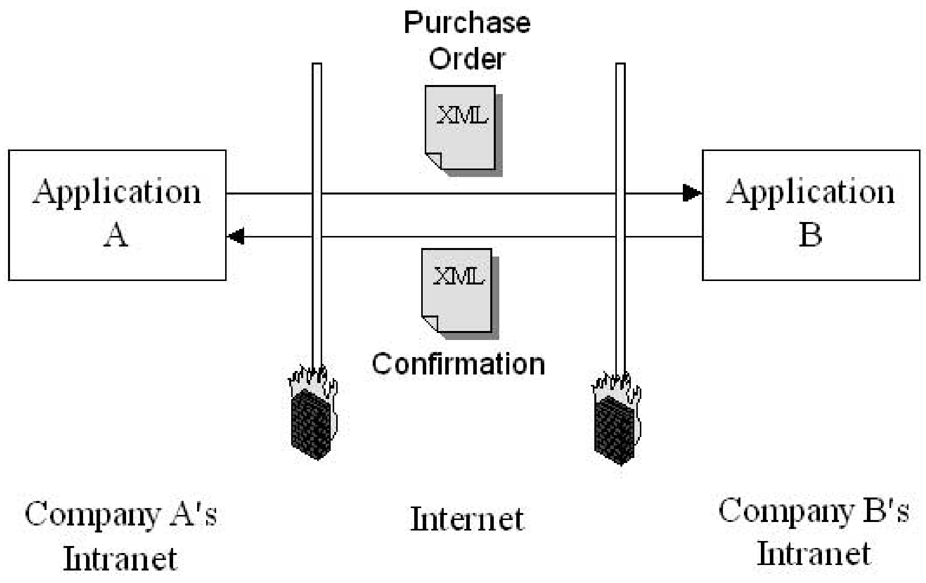

Figure 1. The existing B2B system

The signature server also provides a way to select appropriate keys for signing according to the content of the document by using key selection rules in XML format. Furthermore, the signed documents can be stored in a database where access to the documents is restricted by element-wise access control.

Recently, the Web Services model has attracted the attention of many companies. Web Services use XML as an interface to define business functions that are loosely coupled and dynamically bound. By using SOAP [1], a specification for lightweight Remote Procedure Calls (RPC) using XML documents, we can combine applications independently of specific platforms.

We define functions for signing and verifying Web Services using SOAP as the message format. Because SOAP is independent of the transport layer, we can easily use HTTP(S), Message Queuing, etc., for transport for the SOAP-based messaging . This capability is very important, since many companies have their own network and security policies, and requirements from them depend on their network configuration.

The signature proxy approach makes it easy. It is applicable for existing XML-based B2B systems without modifying them. Our system has been installed at a customer in Japan, and is running as a part of a real system.

To make Web Services more secure and safe, various kinds of security-related Web Services are required. We are currently working on the “XML Security Service Suite (XS-Cube)” that contains security-related Web Services such as digital signatures, encryption, key management, access control, and so on. The signature server is a one of the XS-Cube offerings.

In this paper, we describe basic idea of this system in Section 2. Sections 3 and 4 cover the signature proxy and the signature server, respectively. We introduce a core set for security Web Services in Section 5, and we offer conclusions in Section 6.

2. Basic Idea

The basic idea of the proxy approach is very simple. The structure of the system is shown in Figure 2. A proxy server is placed between each application and the Internet to examine the XML messages exchanged by the applications. The proxy calls the security service of the signature/verification server according to the content of the message. For example, the following sequence is the process of sending an order sheet from Application A to Application B. (The signature server and the verification server may be the same component.)

- Application A in Company A sends the order document to Company B’s application.

- As the order document goes through Proxy A, the document is sent to the signature server and signed before the document is sent to Company B.

- In Company B, Proxy B receives the signed document and sends it to the verification server. The verification server verifies the signed document.

- The result of verification is returned to Proxy B. If the signature is valid, Proxy B removes the signature part from the document and sends the document to Application B. It is possible to keep the signature information if the application wants to know about the signature (e.g., the distinguished name of signer)

- Application B receive the document. After processing, Application B generates a reply document and sends it to Company A.

- As the reply document goes through Proxy B, the document is sent to the signature server where it is signed using B's private key and is then sent to Company A as the signed document.

- In Company A, Proxy A directs the message to the verification server. If the signature is valid, the signature part is removed from the document and the document is sent to Application A.

Since the signature proxy checks the XML messages on the network and decides whether the message should be signed or verified, Applications A and B do not have to know about signatures and verification. Therefore, the existing applications do not have to be modified.

This feature is important from the following two points of view.

- In real-world systems, security functions are often added after the systems have already been developed.

- The addition of security functions to existing systems may create security holes. Therefore, it is desirable that security functions and the applications performing the business functions are designed and implemented independently.

3. The signature proxy

The signature proxy is the component that checks XML documents exchanged by applications and calls the signature/verification services. The proxy can be transparently implemented in hardware such as a Layer 4 switch, or the proxy can be implemented as software such as a servlet. As shown in Figure 2 the signature proxy is used as the gateway which can function as HTTP reverse and transparent proxies.

Figure 3 shows the structure of the signature proxy. Each signature proxy performs the following steps:

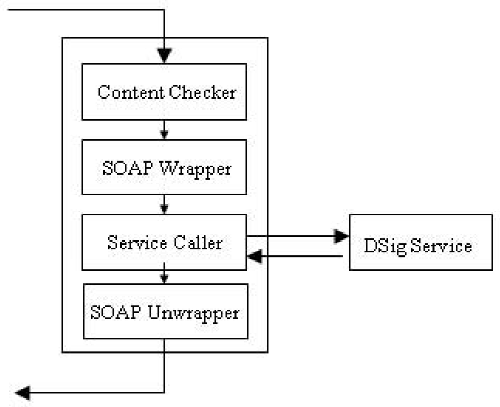

Figure 3. Composition of the signature proxy

1. The Content Checker parses the XML message and decides whether a signature or verification is required. The decision whether or not to sign the document is made by examining the DTD or list of root elements of the XML documents which require a signature. Because signed XML documents which are compliant with the W3C/IETF specification must have a <Signature> element, the proxy calls the verification service if this element exists in the document.

2. The signature server receives and returns a SOAP formatted XML document because the signature server is a Web Service. If the existing B2B system does not support SOAP but instead exchanges (raw) XML messages directly, the proxy transforms the XML messages to a SOAP format (in the SOAP Wrapper module) and also transforms the returned SOAP formatted message back to a (raw) XML message (in the SOAP Unwrapper module). If the existing system is exchanging SOAP-formatted message, these transformations are not necessary.

3. The signature proxy and the signature server exchange SOAP messages (refer to the next section about the syntax). Although SOAP was originally developed as an XML-based RPC, SOAP can be regarded as an envelope wrapping any XML documents. Using SOAP, messages can be exchanged independent of the transport layer. In our system, Apache SOAP [2] is used as the middleware for SOAP message exchange. Also, we extended the system for the IBM Message Queue (MQ).

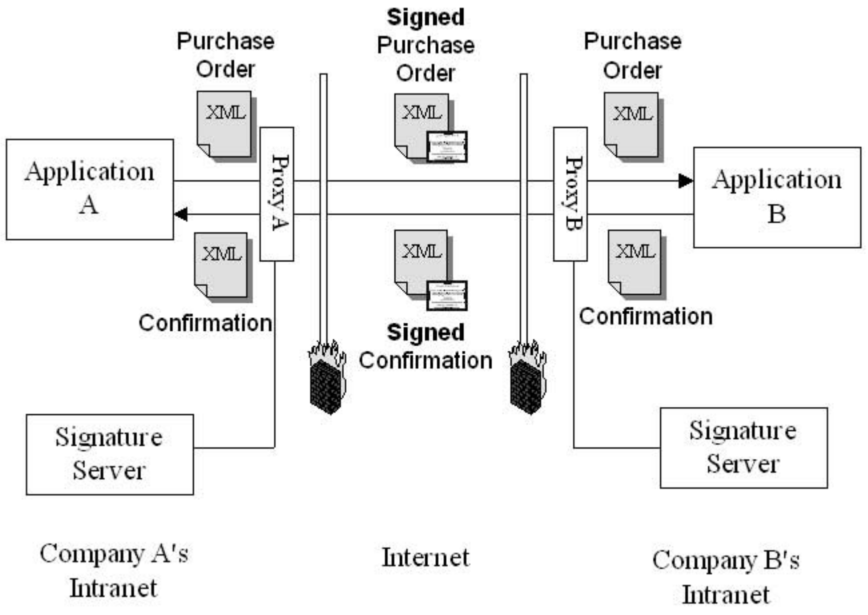

Figure 2. The signature proxy and the signature server

4. The signature/verification server

4.1. Composition

The signature/verification server is a Web Service accepting objects to be signed or verified. Because the interface is a SOAP XML message, the services can be provided by various transport protocols. Figure 4 shows the structure of the signature/verification server. The signature server selects the signature key according to the content of the SOAP requests. The server uses the XML Security Suite for Java (xss4j) [5] as the XML digital signature library. The signed XML documents are stored in a database and then returned to the signature proxy.

Figure 5 shows an example of a request document sent to the signature server. The root element in this XML message is an <Envelope> element. The SOAP-ENV prefix represents the namespace for SOAP as defined in the W3C specification. The <Envelope> element consists of a <Header> element (there is no <Header> element in the example shown in Figure 5) and a <Body> element. The <Header> element has the address of the message or the information which should be interpreted by the intermediary. The <Body> element has the data for the final destination. The <Sign> element, which is the child of the <Body> element, represents the content of the request which has the following information.

- The scope of the signature. The XML message to be

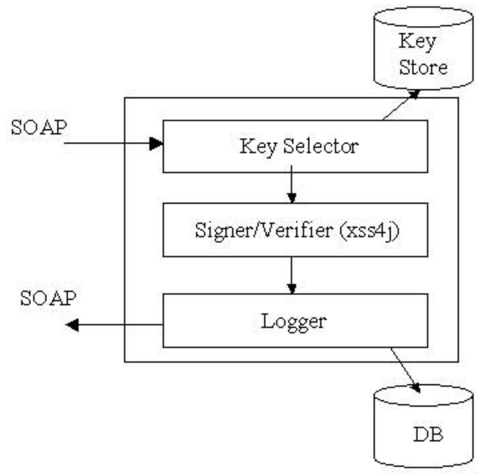

Figure 4. Composition of the signature/verification server

signed can be included in the request message as the child element of the <Target> element. Otherwise, the URI of the object XML can be specified in the URI attribute.

- The format of the result (signed) document. The format can be included in the request message as the child element of the <Template> element. Otherwise, the URI of the template can be specified in the URI attribute.

Each service requester can change the format of the signed document. Although the signature block is represented as a <Signature> element, the format depends on the applications. One application may require a signature block embedded in an XML document as a child element. Another application may like a document in which the <Signature> element is the root element. In order to support to such requests flexibly, the signature server provides a customizable template that specifies the format of a response message.

Figure 6 shows an example of a template file. This example is a template for the "enveloping signature" that means the <Signature>contains a document to be signed. The signature service parses the template and finds the algorithms for the canonicalization and signing processes. Then the service signs the target (element) in the request message (Figure 5) and stores the signature result in the <SignatureValue> element (the <SignatureValue> element is an empty element in the template). From this point, the service uses the <valueOfTarget> element in the template as a macro and replaces the <valueOfTarget> element with the target document. We can use the following macros in the template.

- <valueOfTarget> , replaced with the target element to be signed.

- <valueOfTimestamp> , replaced with the signature timestamp.

- <valueOfNonce> , replaced with a unique number generated by the signature service.

4.2. Key selection by the contents

The signature/verification server manages the signature keys and selects the appropriate key according to the content of the object to be signed. Here is an example:

- The key for the order documents for Company A and B must be different.

- If the amount of money is greater than one million yen, the manager's signature is required. Otherwise the assignee's signature is required.

- Every day from 9 AM to 5 PM the primary key is used, and another key is used at other times.

Our system can change the signature keys (private keys) in the Java2 key store according to the content of the XML document. An example of a key selection configuration file is shown in Figure 7. The configuration file consists of a <keys> element and a <rules> element. The <keys> elements have the key information and the <rules> elements are the key selection rules. A <key> element has information about a single key (key aliases, keystore password, and so on). A <rule> element has one rule for the key selection. The <rule> element includes the condition for selection (<condition> element) and the action if the condition is satisfied (<action> element). The condition is a Boolean formula consisting of several built-in predicates. Table 1 shows the list of built-in predicates.

During key selection, the condition in each of the <rule> elements is evaluated one after the other. The action corresponding to the first condition that is satisfied is executed. In our implementation, an <action> element includes a link to the key information, which is defined in the <keys> element. Using this information, the appropriate key is selected. If no conditions are satisfied, the default rule is applied.

4.3. Logging signed documents and access control

Documents that are signed and verified by the signature/verification servers are stored in a database. Since according to the digital signature law, a document having a digital signature is as valid as a paper document, the transaction records can be preserved as digital documents. Of course, the digital documents should be managed properly. Also, a digital document is subject to auditing, just like a paper document.

In our implemented system, signed and verified documents are automatically stored in a database. The relational database used is IBM DB2 UDB. The complete documents are stored as BLOB data and the text nodes and their associated XPaths are stored in a table so that effective searches are possible.

Element-wise access control to the stored XML documents uses the XML Access Control Language (XACL) [6]. For example, a XACL access policy can describe the following permissions.

- Only managers can see the <price> element.

- Users belonging to the business division can see the element for <businessPartner> .

The role assigned to each user can be managed by the Tivoli Policy Director or by our own original module. We

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                   xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <SOAP-ENV:Body>
   <Sign xmlns="urn:X-WEBSRV:DSIG">
      <Request>
        <Target>
          <m:BookOrder xmlns:m="Some-URI">
            <m:buyer>Naohiko Uramoto</m:buyer>
            <m:item isbn="0-201-48543-5">XML and Java</m:item>
          </m:BookOrder>
        </Target>
        <Template URI="http://www.schema.org/enveloping.tmpl">
      </Request>
      <Respond>
        <element>Signature</element>
      </Respond>
   </Sign>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Figure 5. a example of the request document passed to the signature services

developed a GUI to reference the stored documents. At login time, the roles corresponding to that user are activated. As the user browses XML documents, only permitted elements are displayed.

5. XML Security Services Suite

In Section 4, we described a Web Service for signing and verifying XML documents. Now we are working on developing a complete “XML Security Services Suite” that provides security for Web Services such as:

- Digital signature
- Encryption/Decryption
- Key management (XKMS [7])
- Access control (XACL/XACML)
- Logging/Notary functions

It is possible to build a complex Web Service by combining the services. For example, the signature server introduced in Section 4 can be built by combining the digital signature, key management, and logging services. These services are very generic and it can provide services to various components. For example, services can be called from middleware such as the Apache SOAP engine or by an application server, Java Servlet, EJB, or others.

6. Conclusion

In this paper, we have described a XML digital signature proxy server usable without modification of existing applications. Because this system is implemented as a Web Service, this system can be flexibly applied to various network environments. Also, we have described key selection according to the XML document’s content, function for logging documents, and controlling the access to these documents. Our future work involves developing a core set of security Web services that broadly support systems that use Web Services in a very flexible and secure manner.

References

[1] “Simple Object Access Protocol (SOAP) 1.1,” W3C Note, http://www.w3.org/TR/SOAP

[2] “Apache SOAP,” http://xml.apache.org

[3] “XML-Signature Syntax and Processing,” W3C Proposed Recommendation, http://www.w3.org/TR/xmldsig-core/

[4] “The Signature Law”, http://www.mpt.go.jp/top/ninshou-law/law-index.html (in Japanese), 2001

[5] XML Security Suite for Java, http://www.alphaworks.ibm.com/tech/xmlsecuritysuite

```
<dsig:Signature xmlns:dsig=http://www.w3.org/2000/09/xmldsig#
                 xmlns="urn:X-WEBSRV:DSIG">
     <dsig:SignedInfo Id="sig">
      <dsig:CanonicalizationMethod
                       Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
      <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    </dsig:SignedInfo>
     <dsig:SignatureValue></dsig:SignatureValue>
     <dsig:Object>
       <valueOfTarget/>
    </dsig:Object>
 </dsig:Signature>
```

Figure 6. A example of template file

```
<?xml version="1.0"?> <config xmlns="http://www.ibm.com/foak/dsig/keyconfig">
   <keys>
     <keyInfo id="key1" keyName="key1" keyPass="key1" storePass="key1"/>
     <keyInfo id="key2" keyName="key2" keyPass="key2" storePass="key2"/>
   </keys>
   <rules>
     <rule>
       <condition>
         <and>
           <predicate name="equal">
              <parameter type="xpath">/Order/ID</parameter>
              <parameter type="string">01</parameter>
           </predicate>
           <predicate name="lessThan">
              <parameter type="xpath">//amount</parameter>
              <parameter type="int">10000</parameter>
           </predicate>
         </and>
       </condition>
       <action>
         <select href="key1"/>
       </action>
     </rule>
     <default>
       <action>
         <select href="key2"/>
       </action>
     </default>
   </rules>
 </config>
```

Figure 7. A example of key selection configuration file

Table 1. built-in predicate

predicate name	explanation	example
exist	true if a certain element exist in the document	<predicate name="exist"> <parameter type="xpath">/Order</parameter> </predicate>

equal	true if content of a certain element correspond with a certain string	<predicate name="equal">
		  <parameter type="xpath">/Order/ID</parameter>
		  <parameter type="string">041</parameter>
		</predicate>

intime	true if the signing time is in a certain time period	<predicate>name="intime"¿ <parameter type="time">08:00:00</parameter> <parameter type="time">20:00:00</parameter> </predicate>

indate	true if the signing time is within a certain range of dates	<predicate name="indate">
		  <parameter type="date">2001:01:01:08:00:00</parameter>
		  <parameter type="date">2001:01:16:14:00:00</parameter>
		</predicate>

greaterThan	true if the content of the element is greater than a given constant	<predicate name="greaterThan"> <parameter type="xpath">//amount</parameter> <parameter type="int">10000</parameter> </predicate>
lessThan	true if the content of the element is less than a given constant	<predicate name="lessThan"> <parameter type="xpath">//amount</parameter> <parameter type="int">10000</parameter> </predicate>
isValidSchema	true if the document is valid according to a certain DTD or schema	<predicate name="isValidSchema"> <parameter type="string">file:///dtd/po.dtd</parameter> </predicate>

[6] Michiharu Kudo and Satoshi Hada, "XML Document Security based on Provisional Authorization", ACM CCS, 2000

[7] XML Key Management Specification (XKMS), W3C Note 30 March 2001, http://www.w3.org/TR/xkms/

[8] eXtensible Access Control Markup Language, OASIS, http://www.oasis-open.org/committees/xacml/