Global Information Assurance Certification Paper

XML Digital Signature by Example
Practical Assignment: GSEC Version 1.3

Table of Contents

Joseph Kee **May 2002**

1) Abstract

With the release of the recommendation by the World Wide Web Consortium on XML Digital Signature [1] in February 2002, another milestone has been reached in developing trusted transactions in XML. As more guidelines become available from the Consortium, the development community will have standards to follow in developing web services.

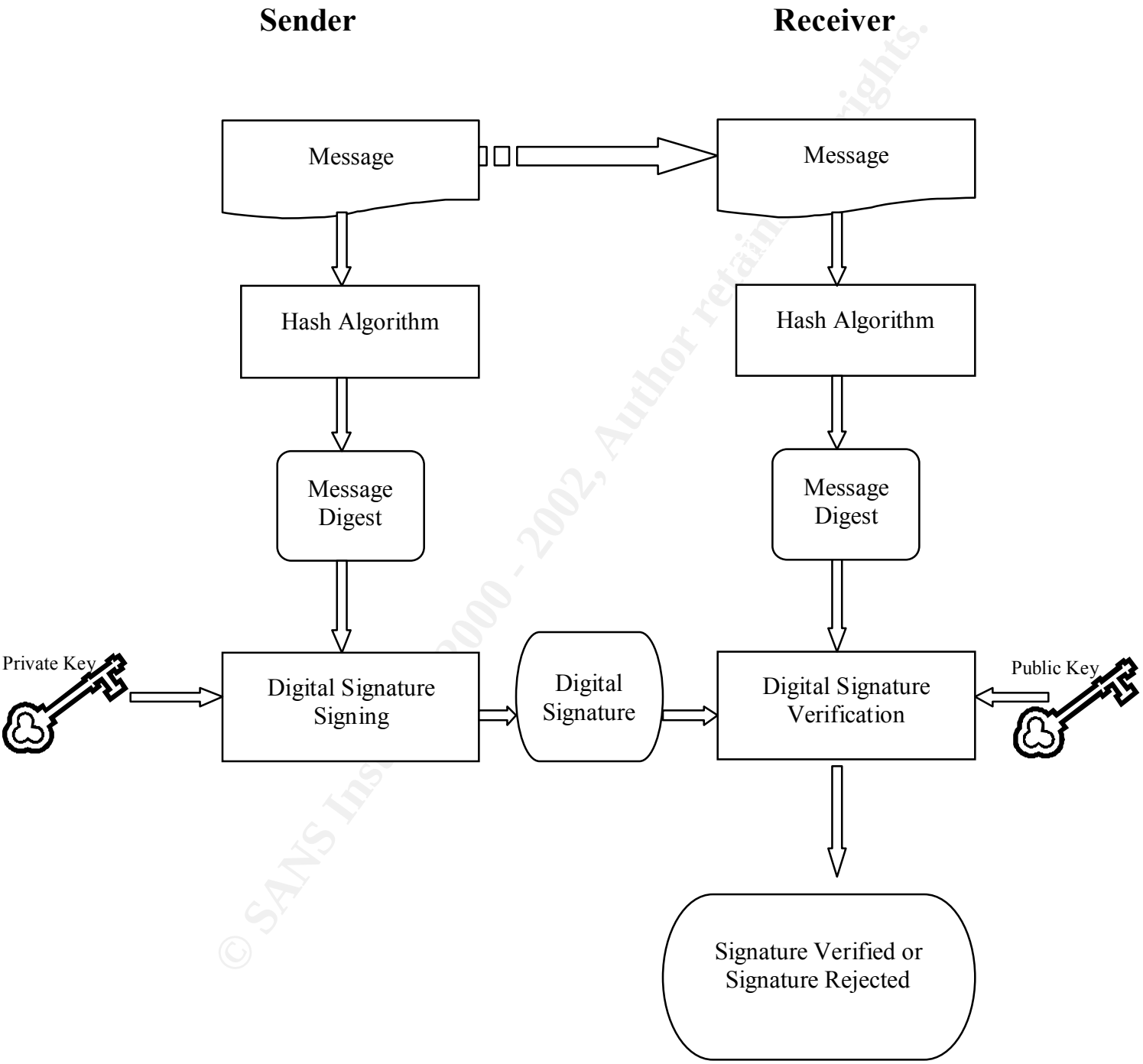
XML Digital Signature provides applications with authentication, data integrity and non-repudiation abilities. XML Digital Signature can be applied to one or more items of digital content. These items of digital content may be any type of data but are usually XML documents, and are referred to by URIs. Also, if the data items are within the same XML document, these local data objects are referred to via a fragment identifier. XML Digital Signature can be used in many application areas, such as electronic mail, electronic fund transfer, software distribution, EDI, etc. This paper starts by giving a general overview of the Digital Signature process, follows by discussion on the different types of Digital Signature. This is followed by an examination of XML Digital Signature implementations by some of the major software providers. Finally this paper will conclude with the security concerns governing the implementation of XML Digital Signature.

2) Signature Processing

Digital signature uses an asymmetric (public) key cryptography system, which employs a public key and a private key. A hash algorithm is used to calculate a fixed length message digest from a variable length message. This message digest is encrypted using the private key of the sender. The encrypted signature is sent to the receiver with the message, which may be encrypted as well.

The receiver decrypts the signature with the public key of the sender. The receiver also re-calculates the message digest using the same hash algorithm as the sender. The two message digests are then compared and if the two message digests are identical, the receiver can be sure that the message has not been changed and the sender cannot repudiate the message. This is a conceptual representation of signature processing. The exact verification process depends on the cryptography standard that a particular implementation employs. The following page shows a graphical representation of this signature generation and verification process.

Signature Processing



3) Types of XML Digital Signature

There are three types of XML Digital Signatures. The Classification of the type depends on the position of the signature in relation to the data being signed. The three types are Detached, Enveloping and Enveloped signature. The following sections describe of each of these three types of XML Digital Signature.

3.1) Detached Signature

In the Detached Signature type, the signature and the data being signed are located independently. In the example following, we are preparing a Digital Signature for the document from a web site called "somewhere.com". In the XML that contains the Digital Signature we reference this external network resource through URI like http://somewhere.com/data.xml. The data and the signature are therefore not in the same document, and are represented by two separate trees in different locations.

<Signature>

<SignedInfo>

.

.

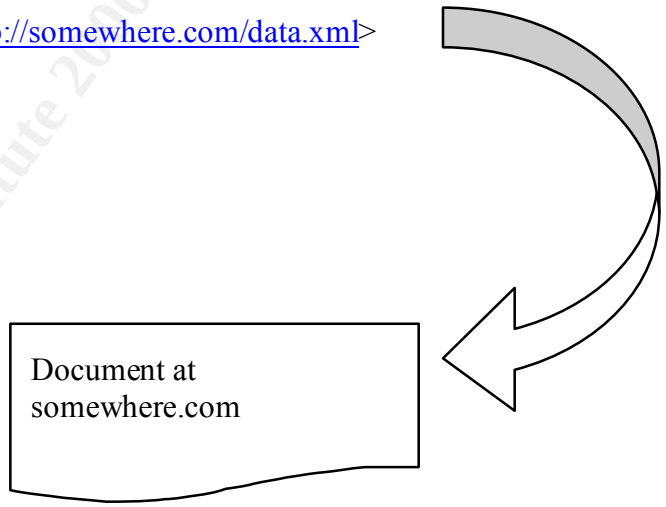
<Reference URI=http://somewhere.com/data.xml>

.

.

</SignedInfo

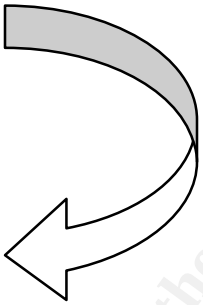
</Signature>



Detached Signature can also include references to objects within the same document as sibling items. In this case the signature and the object being signed co-exist in the same document. This is similar to the other two types of Signature (Enveloping and Enveloped) because the signature and the object being signed are in the same document. The difference is no parent-child relationship exists between the signature and the object being signed for Detached Signature. The following is an example of this kind of Detached Signature. If we view the XML

document in the example as a tree, the signature and the object being signed ("Customer") are two separate branches of the "Sales" tree.

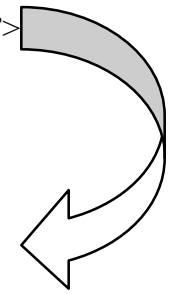
<Sales Id="Sales"
<Signature>
<SignedInfo>
.
<Reference URI="#Customer">
.
</SignedInfo
</Signature>
<Customer Id="Customer">
.
Data
.
</Customer>
</sales>



3.2) **Enveloping Signature**

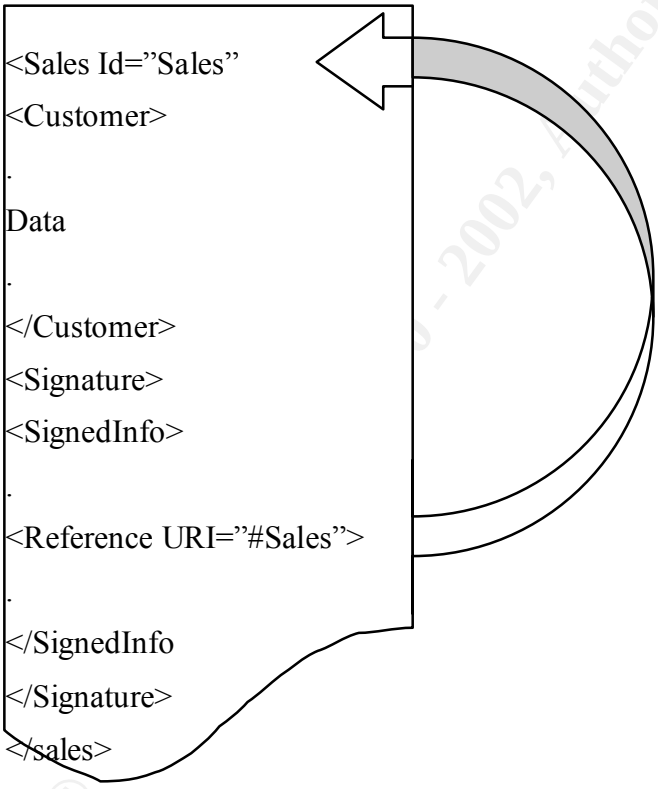
In the Enveloping Signature type, the signature is the parent of the object being signed. Therefore, the signed data object resides inside the XML signature structure. This object is identified via a URI fragment identifier. In this example the signature is the tree and the object being signed is a branch of the tree.

<Signature>
<SignedInfo>
.
<Reference URI="#Sales">
.
</SignedInfo
<object Id="Sales">
Data…
</object>
</Signature>



3.3) **Enveloped Signatures**

In the Enveloped Signature type, the signature is the child of the object being signed. The signature covers the entire XML document that contains the signature as an element. The enveloped signature must take care not to include its own value in the calculation of the Signature Value. In the following example the object being signed ("Sales") is the tree and the signature is a branch in the tree.



4) Components of XML Digital Signature

There are five parts to a Digital Signature. They are the XML Namespace, the signed information, the value of the signature, the key information, and the object to be signed.

4.1) Namespace

A XML namespace [3] distinguishes names used in an XML document by a URI reference. It is used to distinguish duplicate element type and attribute names. In line 1 of the example in Appendix A, the namespace points to http://www.w3.org/2000/09/xmldsig#, which is the W3C recommendation for XML signature processing and syntax. This is the name space that should be used in XML Digital Signature documents. This name space provides a universal unique identifier for element types and attributes names used in XML Digital Signature documents.

4.2) Signed Information (SignedInfo)

This section contains information about what is actually signed. It consists of five subsections. These are the Canonicalization method, the signature method, the reference, the transforms, and the digest method.

4.2.1) Canonicalization method

XML documents that are logically equivalent in the context of an application may differ in physical presentation. An example of physical difference may be the number of white spaces in the start and end tags or the order of the attributes. Every well-formed XML document has an equivalent structural unique canonical XML document. Line 3 of the example in Appendix A describes the Canonicalization method, which is based on the W3C recommendation [6] released on March 15, 2001. The source document is transformed using the W3C recommended algorithm into a canonical XML format, and the digest is then computed over this canonical form of the document. In a Microsoft .NET implementation, transformation can be done either with, or without comments through two separate classes.

4.2.2) **Reference**

A reference contains the location of the entity to be signed, the transforms, the digest method and the digest value. There are one or more references inside the Signed Information section.

4.2.2.1) **URI attribute**

The URI attribute identifies the location of the entity to be signed. The location of the signed entity can either be a remote document, a fragment of a remote document, or a fragment located within the same document as the signature. Line 5 of the example in Appendix A refers to a fragment "SalesId" in the same document. This is an example of an enveloping signature where the signed data is inside the signature structure.

4.2.2.2) **Transforms**

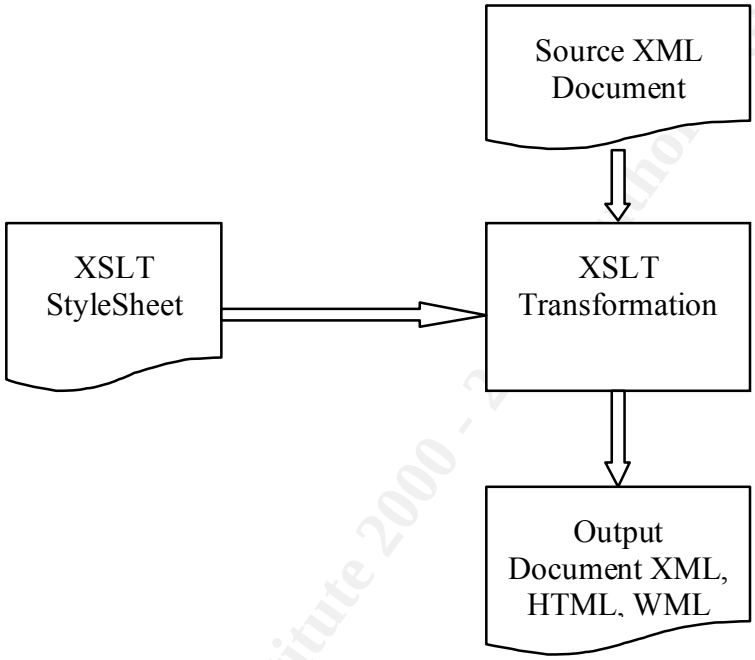
Transforms are optional. We may have one or more transforms where the output of one transform is the input to the next. The output of the last transform is then used as the input to the message digest calculation. Example of standard transforms includes Base-64 encoding, Xpath filtering, enveloped signature transformation and XSLT transformation.

Xpath Filtering

Xpath (XML Path Language) is used when addressing parts of an XML document. It uses a paths expression to identify nodes in an XML document. Xpath supports numerical, equality, relational and Boolean expressions. The Xpath expression is evaluated against each node set of the input document. If the result is true the node set is included in the output, otherwise the node set is excluded. Xpath filtering allows part of an XML document to be omitted when required. An example is the enveloped signature, which must be removed from the digest calculation.

XSLT (Extensible Stylesheet Language Transformation)

The XSLT [16] process takes the source XML document and applies a XSLT stylesheet and creates a new document. The transformation process is basically pattern matching where the XSLT stylesheet is the template of what we want the result to look like. It is recommended that the output from XSLT transform be canonicalized. The following diagram is a representation of this transformation process.



4.2.2.3) **Message Digest**

Signing of a large document is time consuming, therefore the message digest, which is a hash value of the canonical XML document, is signed instead. In the W3C recommendation only one digest algorithm is defined. This is the secure hash algorithm (SHA-1) as designed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA). For a message of less than 2 to the power 64 bits in length, the hashing algorithm produces a 160-bit condensed representation called a message digest. In future, additional strong digest algorithms may be developed through the US Advanced Encryption effort. Line 6 of the example in Appendix A refers to the SHA1 algorithm. Lines 7 and 8 of the example show the digest value.

Secured Hash Algorithm (SHA-1)

SHA-1 [8] is a one-way hash and as such is secured in two aspects. Firstly, it not possible to calculate the message from the message digest. Secondly, two messages will not yield the same message digest. The size of the message used in this hash algorithm is the number of bits in the message. The message is padded at the end so that the total size of the message is multiple of 512 bits. To calculate the message digest, each 512-bit block is separated into 16 words of 32 bits each. A message digest of 160 bits is calculated from running the algorithm repeatedly against each of these 16-word blocks until the whole of the message is processed.

Base64 encoding

The content of the Digest Value in line 8 of the example in Appendix A is the base64 encoding of the 160-bit string produced from the Secured Hash Algorithm. Base64 encoding uses the 65-character subset of the US-ASCII character set. This character set consists of the 26 alphabetical characters in upper case (A to Z), the 26 alphabetical characters in lower case (a to z), the 10 digits (0 to 9), and the "+", "/", and "=" characters. The 65th character ("=") is reserved the special processing function of padding.

The encoding process divides the input into 24 bits block. Each 24-bit block is processed and produces four encoded characters of 6 bits each. The first 144 bits of the 160 bits from the message digest is processed initially as 6 blocks of 24 bits. This results in twenty-four Base64 encoded characters. The remaining 16 bits is padded to form an additional four Base64 encoded characters. This gives a total of 28 Based64 encoded characters as show in line 8 of the example in Appendix A. Padding is performed using two different methods. The input is divided into 6-bit groups, and the last group with less than 6 bits is padded with zeros. If there are less than four 6-bit group, the "=" character is added to the end to make up the four characters. The advantage of using Base64 encoding is that the 65 characters used are universal in most character sets including ASCII US and EBCDIC.

Digest Value for Document Object Model (DOMHASH)

The Digest Value for Document Object Model (DOMHASH) specification [7] has been available since April, 2000. Any examination of XML Digital Signature would not be complete without a look at this model as well.

XML data is structured in a tree form with terminal nodes. The DOM allows programmatic dynamic access and update to an XML document by defining the logical structure of the document. The same internal DOM structure may be

represented by different surface characteristics, and differences such as the order of attributes, white spaces and character encoding may arise. One usage of DOMHASH is to provide a canonicalized digest value of a document operating on a DOM tree. The hash value is calculated for node types such as text and attribute, with comment and document type definitions excluded from the hash value calculation.

4.3) Signature Value

The value of the XML Digital Signature is calculated from the signed information. The Digital Signature Standard from the National Institute of Standards and Technology (NIST) allows the use of the RSA Digital Signature Algorithm and the Elliptic Curve Digital Signature Algorithm (ECDSA) in addition to Digital Signature Algorithm (DSA). All three algorithms must be used in conjunction with the Secured Hash Algorithm (SHA-1), which computes a fixed length message digest from the variable length message. Then RSA, DSA or ECDSA is used to calculate a digital signature from this message digest. RSA refers to the Public Key Cryptography Standard (PKCS1) as described in RFC 2437. RSA Laboratory released version 2 of the specification in 1998. Draft 3 of Version 2.1 was released in April 2002 by RSA. The example in Appendix A uses RSA to calculate the digital signature.

4.3.1) RSA Cryptography

RSA [10] uses both a public and private key. The public key (K) consists of two components, which are the RSA modulus (n) and the RSA public exponent (e). Both the modulus and the exponent are positive integers. The private key also consists of two components, which are the RSA modulus (n) and the RSA private exponent (d). The modulus (n) is the same in both the private and the public key.

RSA signature and verification primitives are defined as follows:

Let m be the message digest, s the signature and K the private key, which consists of the modulus and private exponent (n, d).

The Signature Primitive uses the private key (K) and the message digest (m) as the input and calculates the signature (s). The signature is calculated as: $s = m^d \bmod n$.

The Verification Primitive uses the signature (s) and the public key (n, e) to recover the message digest (m). The message digest is calculated as $m = s^e \bmod n$.

4.3.2) DSA Cryptography

The Digital Signature Algorithm (DSA), which is part of the Digital Signature Standard (DSS) [9] in computing and verifying digital signatures, is published by the National Institute of Standard and Technology. The private key is a randomly generated integer and the public key is calculated using the following formula.

$y = g^x \bmod p$ where

y is the public key

x is the private key

p is a prime modulus

g is calculated from a combination of p and a prime divisor of p – 1.

In DSA signature generation a pair of numbers r and s are generated from the message M and the private key x after the Secure Hash Algorithm is applied to the message. In the verification process, the value of the signature v is recalculated from the received message (M), the public key (y) and signature (r, s). The signature is verified if v is the same as the received r.

4.4) Key Information (KeyInfo)

The Key Information section is optional. It may contain keys, key name, public key management information and a digital certificate. The receiver may obtain this key information from different source. Lines 13 to 22 in the example in Appendix A show the key information. Line 15 in the example identifies to the recipient that RSA cryptography is used in this signature. Lines 16 to 20 show the value of the public key, which consist of a modulus and a public exponent. The receiver uses this key value to validate the signature.

4.5) Object signed

The documents to be signed are identified by URI. In the example in Appendix A, the document to be signed is included in the same document as the digital signature. Line 23 to 26 is the object being signed in this case. The signature is the parent node of the object element (SalesId) to be signed. This is an example of an enveloping signature. XML Digital Signature is useful in many applications. In an electronic mail system, it is used to verify the identity of the sender. In electronic funds transfer applications, it can be used to ensure that the transfer request was not altered in transit. In software distribution applications, it ensures that the software was not tampered with before installation. In legal systems, it can also be used in conjunction with a time stamp to protect and verify the integrity of the document. In Electronic Data Interchange (EDI), it helps to ensure non-repudiation

of transactions. These are a small sample of some of the types of applications that can benefit from digital signature.

5) Implementation

There are a number of Vendors providing Toolkits to implement XML Digital Signature. Most of these vendors support Java, with Microsoft also providing support in their Visual Studio .NET development environment. The following is a brief description of three of the leading providers.

IBM XML Security Suite

IBM developed the XML [11] Security Suite in 1999, with the latest version being released in April 2002. It provides utilities in three areas, including XML Digital Signature, XML Encryption, and XML Access Control Language (XACL). It provides full support in every aspect of XML Digital Signature. For an application that uses the Document Object Model (DOM) or Simple API of XML (SAX), it provides utilities to handle Canonicalization and hashing. For the JAVA programmer it is an excellent toolkit for XML security implementation. This product can be downloaded from the IBM Alpha Works website.

VeriSign XML Trust Services

VeriSign [12] is a leading provider of digital trust services, and provides services called XML trust Services. It consists of four components including XML Key Management Specification, Security Assertion Markup Language, XMLPlay and Extensible Provisioning Protocol. XML Key Management Specification provides integrated authentication, digital signature and encryption services. With their flavor of XKMS, a developer can delegate the entire XML Digital Signature processing to VeriSign. This service is invoked through an XML transaction with a trust processor.

Microsoft Visual Studio .NET

Microsoft .NET technology provides a rich platform for application development. With Visual Studio .NET, millions of VB programmers will be able to learn and deploy Web Services without learning a new programming language like JAVA. The .NET framework also provides extensive support in security. The .NET

framework provides the developer with an easy to use toolset they can use to implement authentication and cryptographic routines. All of the security functions in .NET are packaged together as a single component. By adding a reference to the security component, the developer will be able to use all the security functions including XML Digital Signature.

6) Summary

XML Digital Signature can be used for authentication, data integrity and non-repudiation but cannot be used for data encryption. A fully secured message requires the use of XML encryption in addition to XML Digital Signature. The standard for XML encryption is close to completion, with the Candidate Recommendation being released by W3C in March 2002. Furthermore, a standard protocol for registration and distribution of public keys is required for the success of XML Digital Signature. XML Key Management Specification (XKMS) will provide this standard. As with SSL keys, most organizations will likely delegate this function to a third party such as VeriSign. Other related areas to XML security include XACL and SAML. Extensible Access Control Language XACL and Security Assertion Markup Language SAML are published by the Organization for the Advancement of Structured Information. (OASIS)

As more companies increase their use of XML for business-to-business applications, the security of these XML-based transactions becomes paramount. It is vital that the private key used in XML Digital Signature is protected against modification and disclosure, otherwise transaction security can be compromised. When implementing XML Digital Signature it is also useful to have a full understanding of the cryptographic algorithm being used because, as always, poor implementation will increase the vulnerability of any system.

Appendix A

Example generated using Visual Studio .NET.

Line	
1	<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
2	<SignedInfo>
3	<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
4	<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
5	<Reference URI="#SalesId">
6	<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
7	<DigestValue>
8	5hhaOqoD+/8KPBK7kVB5oruAO6M=</DigestValue>
9	</Reference>
10	</SignedInfo>
11	<SignatureValue>
12	TdInKSBKGykx8zKjMlbcYmFOArMLTECcowoyR4RNOy0KBaWktS0PF5DsPdnrAL1VUAtfBC7k9+ldo8Xb/v2ks74bvJNE7D5Queu/vTeqVPQCscf4RUiVOr1PsumWxAlGtiHRHWgubDUsSYMy9rFrMTKeYjJGrhLvirEMssk2kYU=</SignatureValue>
13	<KeyInfo>
14	<KeyValue xmlns="http://www.w3.org/2000/09/xmldsig#">
15	<RSAKeyValue>
16	<Modulus>
17	txJ9CpOap1IRkC3HalgkKQniqiHzIiiV4l3ghdtQ9XvVfOoAC04XcP3WYikA9JlgSxcr+Qm54FXtcOHJYBHtYi3tyKgHBDQysj7glewp+m7mZUHP4mLzDYOcGBOBT1sjJHwrJiADSc50ZvrSzGnA0WbmgACHptgoHZUwz21XN2U=</Modulus>
18	<Exponent>
19	AQAB</Exponent>
20	</RSAKeyValue>
21	</KeyValue>
22	</KeyInfo>
23	<Object Id="SalesId">

24	<Product xmlns="http://www.aisnet.com">
25	Intrusion Detection</Product>
26	</Object>
27	</Signature>

Appendix B

References

[1] XML – Signature Syntax and Processing (W3C Consortium Feb 12, 2002)
http://www.w3.org/TR/xmldsig-core/

[2] XML – Signature Interoperability (W3C Consortium May 2002)
http://www.w3.org/Signature/2001/04/05-xmldsig-interop.html

[3] Namespace in XML (W3C Consortium January 14, 1999)
http://www.w3.org/TR/REC-xml-names/

[4] XML Electronic Signatures (Institute for Applied Information Processing and Communication Sept 24, 2001)
http://www.nanobiz.com/xmlsig/docs/iaik/iaik.htm

[5] XML Namespaces by Example (XML.COM January, 1999)
http://www.xml.com/lpt/a/1999/01/namespaces.html

[6] Canonical XML (W3C Consortium March 15, 2001)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315

[7] Digest Value for DOM (IBM April 2000)
http://www.ietf.org/rfc/rfc2803.txt

[8] Secured hash standard (Federal Information Processing Standards April 17, 1995)
http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt

[9] Digital Signature Standard (Federal Information Processing Standards January 27, 2000)
http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf

[10] RSA Cryptography Standard (RSA Laboratory Oct 1998)
http://www.ietf.org/rfc/rfc2437.txt

[11] IBM XML Security Suite (IBM April 26, 1999)
http://www.alphaworks.ibm.com/tech/xmlsecuritysuite

[12] VeriSign XML Key Management (VeriSign October 2001)
http://www.verisign.com/resources/wp/xml/keyManagement.pdf

[13] Fact Sheet on Digital Signature Standard (National Institute of Standards and Technology May 1994)

http://www.nist.gov/public_affairs/releases/digsigst.htm

[14] XML & Security (XML Journal Dec 2000)

http://www.sys-con.com/xml/articleprint.cfm?id=91

[15] XML and Security (XML Journal Dec 2001)

http://www.sys-con.com/xml/articleprint.cfm?id=294

[16] XSL Transformations (W3C Consortium Nov 16 1999)

http://www.w3.org/TR/xslt

[17] KeyTools XML (Baltimore May 2002)

http://www.baltimore.com/keytools/xml/w3cietf.asp

[18] XML Digital Signature (The XML Cover Pages April 26 2002)

http://www.oasis-open.org/cover/xmlSig.html

Last Updated: May 25th, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event