XML Security

CMSC691X-Term Paper
Department of Computer Science
University of Maryland, Baltimore County
Baltimore, MD 21250
Kemal Akkaya
email:kemal1@cs.umbc.edu

1. Introduction

XML (Extended Markup Language) has gained an important amount of interest in last recent years and will become the standard for maintaining data on the World Wide Web. XML is a major enabler of what the Internet, and latterly Web services, require in order to continue growing and developing.

XML provides semantic data model in order to keep track of data through the web. This way it differs from HTML and brings great innovation for dealing with huge amount of data. By using the data models, schemas and query languages that come with XML, web data is easily maintained as in the case of conventional databases.

Despite the many advantages XML provide, there are critical security issues related to using XML. Any business transaction through the web using XML can be sniffed or spoofed. A lapse in the security of such data could produce dire results. Another example can be given for medical records of a patient, which are accessed through XML. Again any third party can obtain this private information and can use it or change it easily if there is no protection mechanism. Security breaches with such data can have very adverse consequences.

What is meant by XML security falls into two basic categories: the security of XML instances themselves, and the use of XML technology to enhance security for a wider range of applications. In this paper, we'll look at these issues and the applications for XML security. Next section talks about XML signatures and encryption by giving examples. Then in section 3 we'll talk about the usage of XML for enhanced security. Section 4 talks about some XML security languages. In section 5 we explain the commercial products and tools in this area. Finally, we conclude with a conclusion in section 6.

2. XML Encryption and Signatures

XML security is a wide area that can be interpreted in many ways. Yet a lot of work remains to be done on security-related issues before the full capabilities of XML languages can be realized. At present, encrypting a complete XML document, testing its integrity, and confirming the authenticity of its sender is a straightforward process. But it is increasingly necessary to use these functions on parts of documents, to encrypt and authenticate in arbitrary sequences, and to involve different users or originators. At present, the most important sets of developing specifications in the area of XML-related security are ***XML encryption, XML signature***. In this section, we'll try to investigate all the definitions and applications for XML security.

The power of XML comes from its flexibility and extensibility paired with its semantics and structure. But, as mentioned earlier, these characteristics also introduce

significant security issues. XML “instances,” or fragments, can present data from multiple sources. The components of an XML instance are like baking ingredients-you mix them together in varying amounts based on a recipe. These ingredients can be spread throughout the Internet “kitchen.” This potential dispersion of information introduces validation issues. Without a reliable method of validating the source of the data and the accuracy of the information itself, a hacker could introduce spoofed data into a transaction or transformation of data.

XML instances can use links to resources, making them ephemeral in nature. With all the “ingredients” identified, XML provides you with two options: You can make the cake, by collecting and presenting the XML information in an XML instance, or you can take a picture of the cake, by providing pointers and links to the applicable information. In either case, the end result for users looks the same. A complete XML instance may be presented without any real data in it, just Uniform Resource Identifiers (URIs) that point to particular elements. This ephemeral quality really extends the “security is only as strong as its weakest link” metaphor -it could be that you have limited control over some of the data but must still rely on the security controls surrounding it. But validity remains an issue, and any need for long-term archiving exacerbates validity concerns. An application must ensure that the data and its context are captured and stored so that they can be verified years into the future.

XML is transport independent; it does not specify a particular transport mechanism. Current implementations use HTTP for transport-a universal skeleton key for virtually any network. Firewalls do not stop HTTP, and they will not stop XML, regardless of your application. Thus, XML can generate security problems that other forms of data do not. XML instances can look exactly the same on the surface and yet still be different in content. Even well formed (for example, syntactically correct) XML instances may be structured differently due to tag placement, use of white space, and other style mechanisms. These differences, though they do not impact the quality and content of the information, introduce a level of ambiguity that adds to the need for validity and security.

Ensuring the security of XML documents can be very complex. When designing the flow of data within an application or among applications, an organization must consider four primary security objectives: confidentiality, integrity, authentication, and nonrepudiation. Due to XML' s flexible and ephemeral nature, you must establish and apply these objectives carefully with XML data.

The objective of confidentiality is protecting the contents of a document or string of data. This basically translates to ensuring that no inappropriate or unauthorized third parties can access or view messages that are not intended for them.

When processing data in two separate applications, potentially on different systems and in different locations, ensuring its integrity is a crucial objective. Measures that ensure the data' s integrity must be consistent with the level of assurance required. Another critical objective is authentication, or verification of the originator of XML data. Devices and users must have stringent means of ensuring that data originators are properly authenticated before any transactions/processes are performed.

Finally, effective nonrepudiation mechanisms are another must with XML. This typically means providing proof of the signature source, which would make it difficult for

the source to deny having signed the data contents (assuming the core validation process is successful).

To achieve these security objectives, organizations must understand how XML processes and transactions are performed. The typical process begins with authentication, which is either inferred from a session or specific to a transaction. It is common to authenticate for a session that may contain multiple transactions. Sites with strict authentication requirements may require authentication before submission of any transaction.

Second, a transaction such as a purchase order is created or modified. Next, the transaction is submitted to begin transmission to the destination. This is where signing and encryption occur. The transaction is then received, and the data retrieved. This is where the reverse operation, signature verification (or decryption), is performed to provide confidentiality and integrity. Finally, archival capability must be provided for transactions and documents that require storage.

The security functions that accompany this process are encryption and signatures. Encryption provides the confidentiality, and signatures provide the integrity, authenticity, and, in some cases, nonrepudiation.

2.1 XML Signature

XML developers recognized the need early on to provide a mechanism to control or manage the data that is passed and presented in XML transactions. The primary requirements for data in transit are completeness and accuracy (integrity). XML signature is the simplest way to guarantee these objectives.

XML Signature defines the syntax required to sign all or part of an XML instance. XML, with its extensive capabilities and extreme flexibility, does not lend itself well to the needs of digital signatures, where a misplaced space results in a completely different fingerprint that is unverifiable. To address these issues, the use of canonicalization (XML-C14N) has been introduced. XML-C14N, defined in a W3C draft document called *Canonical XML*, follows a preidentified set of processing rules to structure an instance into its simplest form. The goal is to ensure that instances are structured the same way every time, thus ensuring that digital signatures will not be confused due to stylistic differences such as misplaced spaces. Note that even with C14N, you should identify the rule set or method used for extra assurance.

An XML operation must provide enough information to ensure that the signatures are verified, which provides sufficient integrity, authentication, and nonrepudiation. The information in a signed XML instance contains:

- The canonicalization method, which identifies the specific C14N rule set that simplifies and structures an XML instance prior to signature. This information ensures the proper form of the data contents being signed, so that the verification algorithm will succeed if the data contents have not been modified.
- The signature method, which identifies the algorithm used to sign the message digest. The message digest is a unique string of fixed size that is output when content is processed through a one-way hash function, specified in the digest method. (The digest method identifies the algorithm that creates the message digest signed by the signature method.) This signature method validates the

person or process that signs the data contents, thus ensuring authentication and nonrepudiation.

- The digest method, which identifies the algorithm used to create the message digest signed by the signature method. The digest method ensures that data contents can be processed with the same algorithm when comparing the resulting values.
- The digest value, which contains the message digest itself-the fixed-string output when data is processed through a message digest algorithm. This string is unique and one-way; there is almost no chance of duplicating the string with other contents or re-creating the original contents from the string. It is the fingerprint to the data contents being signed; upon a valid comparison of digest values, the integrity of the contents is assured.
- The reference information, which provides information about the data contents that were signed. The reference may include type information on signed non-XML data.
- The optional signature properties, which add context to a signature. This context may include a timestamp or serial number. These items can optionally be signed and included in the reference information for the signature itself.
- The optional manifest, which groups together multiple references and creates a single digest value for signatures by one or more users. This provides a mechanism to allow, for example, a supervisor to approve a number of individually signed transactions in one group, rather than individually approving and signing each transaction.

The “œore generation” process of signing an XML instance begins with canonicalization to simplify the data contents. The rest of the signing process is similar to the typical digital signature process-the digest value, or fingerprint, of the data contents is created using a digest method. This digest value is then signed with an entity' s private key.

For signature verification, XML Signature provides for a “œore validation” process with two steps. First, the signature is validated to ensure the authentication and nonrepudiation of the signee. Second, the digest value(s) is verified to ensure that the data has not changed, thus confirming content integrity. The XML Signature specification is an ongoing, joint effort by the IETF and W3C. The following example is taken from XML-Signature Syntax and Processing document of W3C [8].

Example:

```
[s01] <Signature Id="MyFirstSignature" mlns="http://www.w3.org/2000/09/xmldsig#">
[s02] <SignedInfo>
[s03] <CanonicalizationMethodAlgorithm="http://www.w3.org/TR/2001/ RECxml-
c14n-20010315"/>
[s04] <SignatureMethod Algorithm="http://www.w3.org/2000/09/ xmldsig#dsasha1"/>
[s05] <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06] <Transforms>
[s07] <Transform Algorithm="http://www.w3.org/TR/2001/REC-xmlc14n-
20010315"/>
[s08] </Transforms>
[s09] <DigestMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#sha1"/>
```

[s10] <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[s11] </Reference>
[s12]</SignedInfo>
[s13]<SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
[s14]<KeyInfo>
[s15a] <KeyValue>
[s15b] <DSAKeyValue>
[s15c] ... <Q>...</Q><G>...</G><Y>...</Y>
[s15d] </DSAKeyValue>
[s15e] </KeyValue>
[s16]</KeyInfo>
[s17]</Signature>

SignedInfo element between lines s0 and s12 contains the information about the used algorithms. The calculated signature value is included in the SignatureValue element in line s13 which is actually outside the signed section. In line s04, the SignatureMethod includes the algorithm used to convert the canonicalized SignedInfo into the SignatureValue. And finally, between lines s14-and s16 there is an optional element which is called the KeyInfo element and used to validate the signature.

2.2 XML Encryption

Proper encryption is crucial for XML data security, particularly sensitive data that' s passed across unprotected networks such as the Internet. Enter XML Encryption. It' s easy to think of encryption as a "blanket" operation -data is encrypted on one end, then decrypted on the other. But more information is required to perform this operation successfully. In an XML instance, there are four basic types of information:

- Encrypted content, which contains the actual encrypted data or a reference to the location of this data. There is virtually unlimited flexibility in both the types of data that can be included and methods for logical data collection for encryption.
- Unencrypted content, which contains other information that is pertinent to the context of the interaction but is not encrypted for some reason, perhaps due to performance concerns or because it was not deemed private or sensitive enough to warrant encryption.
- Key information, which contains information or pointers to information about the keys that perform the encryption, and, therefore the keys that perform the decryption. The key information can be maintained elsewhere and replaced by a URL in the XML instance.
- Recipient information, which contains information about one or more intended recipients of the encrypted data. This information is optional, thus allowing situations where the applicable recipient information is known or provided out of band, such as with business partners that have a preexisting contractual relationship.

Encrypting XML data follows the traditional encryption steps for public key cryptography. First, the data is encrypted, typically using a randomly created secret key. Then the secret key is encrypted using the intended recipient' s public key. This

information is packaged to ensure that only the intended recipient can retrieve the key and decrypt the data. Decryption involves applying the private key to decrypt the secret key, then decrypting the data with the secret key.

There are a number of options being evaluated for encrypting XML portions, as well as multiple ways of embedding these encryption elements within an XML instance. The W3C convened an XML Encryption working group to create a standard for this function. The following example is from W3C working draft [9]:

Example:

Smith' s credit card number is sensitive information. If the application wishes to keep that information confidential, it can encrypt the CreditCard element:

```
<?xml version=' 1.0' ?>
<PaymentInfo xmlns=' http://example.org/paymentv2' >
<Name>John Smith</Name>
<EncryptedData Type=' http://www.w3.org/2001/04/xmlenc#Element'
xmlns=' http://www.w3.org/2001/04/xmlenc#' >
<CipherData>
<CipherValue>A23B45C56</CipherValue>
</CipherData>
</EncryptedData>
</PaymentInfo>
```

By encrypting the entire CreditCard element from its start to end tags, the identity of the element itself is hidden. (An eavesdropper does not know whether he used a credit card or money transfer.) The CipherData element contains the encrypted serialization of the CreditCard element.

2.3 Transformations

Different type of operations such as signing, encrypting, modifying etc. can be performed on an XML document at any order. For instance the user even can modify a document which is already partially signed or encrypted. In those cases future validations or decryptions would not be affected [7].

Both XMLSignature and XML-Encryption operations may be performed on an XML document at any time and in any order, especially in scenarios such as workflow. For example, Alice wishes to order and pay for a book from Bob using the mutually trusted payment system ZipPay. Bob creates an order form including the book title, price and his account info. He wants to sign all of this information, but will subsequently encrypt his account info for ZipPay only. He sends this to Alice who affirms the book title and price, signs the form and presents the twice-signed order with her own payment information to ZipPay. To validate both signatures ZipPay will have to know that the cipher data version of the encrypted information is necessary for validating Alice' s signature, but the plain data form is necessary for validating Bob' s signature.

The example below, taken from [6], shows how the recipient of a document can be advised as to the proper order of decryption and signature verification. The first fragment shows the part of the document to be signed, the order element; within this, the personal and financial details of the cardinfo element, lines 7 to 11, are in clear text but some encrypted data is already present (line 12).

Order element within an XML document

```
[01] <order Id="order">
[02] <item>
[03] <title>XML and Java</title>
[04] <price>100.0</price>
[05] <quantity>1</quantity>
[06] </item>
[07] <cardinfo>
[08] <name>Your Name</name>
[09] <expiration>04/2002</expiration>
[10] <number>5283 8304 6232 0010</number>
[11] </cardinfo>
[12] <EncryptedData
id="enc1"xmlns="http://www.w3.org/2001/04/xmlenc#">...</EncryptedData>
[13] </order>
```

Order document after signing and further encrypting and now showing transform information

```
[01] <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
[02] <SignedInfo>
[03] ...
[04] <Reference URI="#order">
[05] <Transforms>
[06] <Transform Algorithm="http://www.w3.org/2001/04/ xmlenc#decryption">
[07] <DataReference
URI="#enc1"xmlns="http://www.w3.org/2001/04/xmlenc#"/>
[08] </Transform>
[09] <Transform Algorithm="http://www.w3.org/TR/2000/ CR-xml-c14n-
20001026"/>
[10] </Transforms>
[11] ...
[12] </Reference>
[13] </SignedInfo>
[14] <SignatureValue>...</SignatureValue>
[15] <Object>
[16] <order Id="order">
[17] <item>
[18] <title>XML and Java</title>
[19] <price>100.0</price>
[20] <quantity>1</quantity>
[21] </item>
[22] <EncryptedData Id="enc2"
xmlns="http://www.w3.org/2001/04/xmlenc#">...</EncryptedData>
[23] <EncryptedData Id="enc1"
xmlns="http://www.w3.org/2001/04/xmlenc#">...</EncryptedData>
[24] </order>
[25] </Object>
[26] </Signature>
```

The Signature element, lines 1 to 26, now includes the previous order element, lines 16 to 24, with its earlier plain text cardinfo element encrypted and shown in the single line 22. There is two transform references which are decryption (lines 6 to 8) and canonicalization (line 9). The decryption transform instructs the signature verifier to decrypt all the encrypted data except for the one specified on line 7 in the DataRef element. After the EncryptedData element in line 22 is decrypted, the order element is canonicalized and the signature fuly verified.

2.4 Problems with Encryption and Signatures in XML

There are three major problems with digital signatures and encryption in XML documents [7]. First of all, the different sequence of encryption or signatures applied by different people to an XML document is not as straightforward as just doing it as a whole for the document. In that case, every user acting on the XML document will need to know the required sequence in order to take further action. Moreover, dealing with the private keys for different parts of the document will introduce further complications. A typical example might be a workflow environment where a series of data records are used by different applications and users.

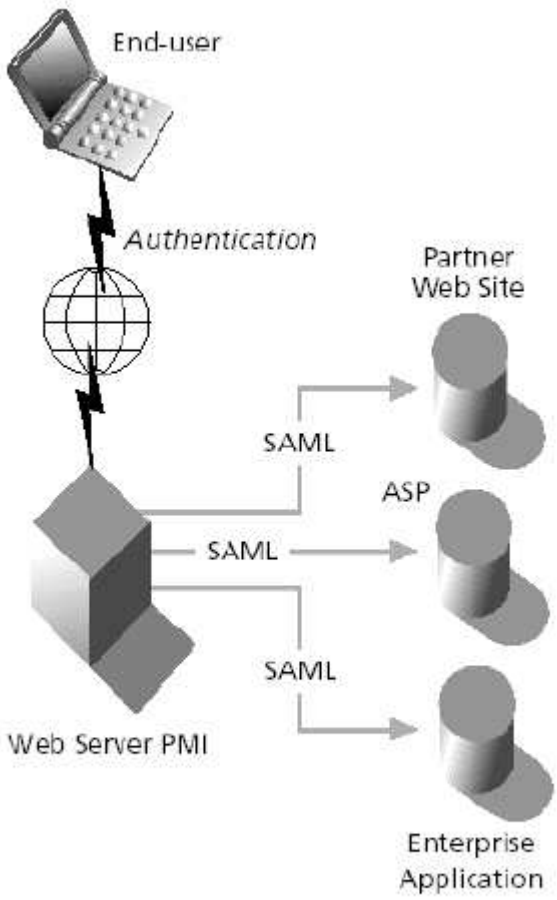
The second problem is on the powerful search capability of XML. Since encrypting the document means, encrypting the tags or information provided by the schemas, then there will be no more ability to search on encrypted text which will mean nothing to the XML language search tool. What' s more, as both the plaintext and ciphertext versions of the tags will now be known the opportunities for hostile cryptanalysis will be made easier.

And finally, there is another problem related to the digital signatures. It is known that the digital signatures use message digests which are obtained by using a cryptographic hash algorithm. In that case, even a character change in the message will result in a different hash value, leading to a different message digest. This might be the case for the XML documents where two of them are logically equivalent but have different texts. This is due to line delimiters, empty tags or presence of comments. In order to deal with this problem, the canonical form is proposed. The canonical form suggests that two XML documents will be considered same in terms of physical context as long as logical structures are same. The canonical form is very useful where different parsers may generate different texts and hence different message digests. In order to avoid such complications, canonical form suggests that the message digest should be done in canonical form during the process of both signing and validating.

3. XML for Security Functions

The other side of the XML security coin is leveraging the features of XML to provide enhanced security for a broader range of applications and processes. Under the auspices of the OASIS XML-Based Security Services Technical Committee, SAML is one of many markup languages being created via the syntax and semantics of XML. Security Assertions Markup Language (SAML) is a new standard that uses XML to encode authentication and authorization information. That information can then be moved between systems within an organization, or between organizations in a transaction. In this case, it' s using XML to extend security functions for authentication and authorization. The primary goals of SAML are to create the core assertions required for credential

exchange; to define protocols for requests, responses, and ground rules; and to define SAML bindings to other protocols, such as HTTP, Simple Object Access Protocol (SOAP), and ebXML. SAML shares credential information across disparate Web sites or applications, essentially operating like "CredEx" and delivering the credentials in real time-anytime, anyplace. Because its basis is XML, SAML is platform-independent and can move around as simply as text. SAML can be the solution for a variety of security problems facing many organizations today.



SAML Architecture

Usage Cases

There are a number of usage cases being evaluated that illustrate the functionality of XML, but two general categories emerge. The first is single sign-on, with both "push" and "pull" models, as well as support for third -party security services. The intent is to enable a user to sign on once, then permit Web sites to negotiate follow-up authentication.

The second usage case, Authorization Services, shares authorization information through an application chain, for back office transactions, and with third-party security services. Authorization services can provide a basis for managing business relationships

or personalizing the end-user experience by passing along profile information to make an access control decision.

In a typical Web services-based system, vendor A makes a service available on the Web that vendor B decides to incorporate into its own Web offering. Users who log in to vendor B's site might be using the Web service from vendor A without being aware of it. Preferably, the entire process would be transparent to the user. However, if vendor A's service requires separate authentication, that transparency would be weakened, which could bother users. A system where the user's login to vendor B were passed along to vendor A would be far better. Vendor A would agree to trust users automatically whom vendor B already acceptably authenticates. Such an approach is an uphill struggle.

Historically, each separate system in an organization has had its own login protocol. Administering separate login information was always a problem: people were hired, changed jobs, or left the organization-all requiring changes to authorization and authentication. In addition, users disliked having to log in separately to each application. The goal now is to enable-as far as possible-single sign-on across organizations and across application vendors.

AuthXML and Security Services Markup Language (S2ML) were two approaches to the problem of adding authentication features to XML. Both used XML as the metalanguage to encode information on authorization and authentication. AuthXML is a specification for encoding authentication and authorization information in transportindependent XML. (The "Auth" stands for both authentication and authorization.) AuthXML allows security authorities in separate organizations to communicate about security including authentication, authorization, user profiles, and authenticated user sessions. AuthXML was designed to simplify transactions between partners that may be using different security systems and between applications that need integrated security. Outlook Technologies, Securant Technologies, and other contributors developed the AuthXML specification.

Similarly, S2ML is a standard for enabling secure e-commerce transactions through XML. The idea was to provide a common language for security between companies in B2B and B2C business transactions. S2ML allows companies to exchange authentication, authorization, and profile information securely regardless of platform. Authors of the S2ML specification included Bowstreet, Commerce One, Jamcracker, Netegrity, Sun Microsystems, VeriSign, and webMethods. AuthXML and S2ML have now been subsumed into SAML.

Work on the SAML standard started in January 2001. SAML got its formal name in February. OASIS (Organization for the Advancement of Structured Information Standards) is still working on specification set of SAML 1.0. It's not standardized yet. As for the spec itself, the cochairs of OASIS' SAML committee have their fingers crossed that it will become a completed standard by summer (2002), with full products based on SAML out by fall. Despite the fact that SAML is still not a finished standard, that does not mean you cannot do developing with it. In fact, Netegrity has already released its JSAML toolkit; a free downloadable Java technology-based toolkit that will help corporate and independent software vendors deploy SAML-based applications in their enterprises. Netegrity hopes the free toolkit will get people started developing ahead of the specification's completion.

4. Security Languages

There are also a handful of other languages in the early stages of development and support. One is the XML Key Management System (XKMS), a specification spearheaded by Verisign that' s designed to register, locate, and validate keys used for security purposes. Another is the XML Trust Assertion Service Specification (XTASS), a standardized way to make trust assertions for business partner relationships, also initiated by Verisign. A third language, the eXtensible Access Control Markup Language (XACML), permits access control policies to be created and applied against XML instances. The SAML technical committee is discussing this specification, but it' s more closely aligned with XML Signature and XML Encryption.

4.1 XKMS

XKMS specifies protocols for distributing and registering public keys, suitable for use in conjunction with the proposed standard for XML Signature [XML-SIG] developed by W3C and IETF and an anticipated companion standard for XML encryption. The XML Key Management Specification (XKMS) comprises two parts -- the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS).

The X-KISS specification defines a protocol for a *Trust service* that resolves public key information contained in XML-SIGelements. The X-KISS protocol allows a client of such a service to delegate part or all of the tasks required to process <ds:KeyInfo> elements. A key objective of the protocol design is to minimize the complexity of application implementations by allowing them to become clients and thereby to be shielded from the complexity and syntax of the underlying PKI used to establish trust relationships. The underlying PKI may be based upon a different specification such as X.509/PKIX, SPKI or PGP. The X-KRSS specification defines a protocol for a web service that accepts registration of public key information. Once registered, the public key may be used in conjunction with other web services including X-KISS.

Both protocols are defined in terms of structures expressed in the XML Schema Language, protocols employing the Simple Object Access Protocol (SOAP) v1.1 and relationships among messages defined by the Web Services Definition Language v1.0 (WSDL). Expression of XKMS in other compatible object encoding schemes is also possible. W3 consortium is still working on XKMS and there is a draft on it.

4.2 XTASS

The XML Trust Assertion Service Specification defines an architecture and retrieval protocol for "Trust Assertions". A Trust Assertion consists of a statement bound to a unique identifier that is cryptographically authenticated. Trust Assertions may be used to establish and manage long term trust relations between principals. The trust assertion architecture is designed to be extensible to support management of any form trust assertion. In particular assertions need not be bound to a public key infrastructure. A Trust Assertion may be bound directly to a document that represents or facilitates a financial transaction, for example bonds, equities and bills of lading.

XTASS provides a generic framework for specifying information relevant to any form of trust assertion:

- The identifier of the Issuer
- The time instant of issue
- Reissue location and scheduling

Assertions may be addressed to a specific audience. Relying parties may be required to verify the status of an assertion before each use. The research for defining XTASS specification set is still in progress as a draft.

4.3 XACML

OASIS, the XML interoperability consortium, defined a standard for security access control using XML. The eXtensible Access Control Markup Language (XACML) defines an XML specification for expressing policies for information access over the Internet. It will define the representation for rules that specify the who, what, when and how of information access. This is also an ongoing effort, which has not yet been standardized.

5. Commercial Products and Tools

There are a lot of tools and products for ensuring XML security. Some of these are below:

- ***Baltimore KeyTools XML for C++ and Java*** represents the fusion of XML technology and digital certificate-based security and allows developers to build full Public Key Infrastructure based security into XML applications.
- ***IBM alphaWorks XML Security Suite*** is a tool that provides security features such as digital signature, encryption, and access control for XML documents.
- ***nanobiz.com NanoX Platform*** provides customers the ability to digitally sign documents using leading-edge emerging XML standards for authentication of digital signatures Available with the platform is a suite of Extragration modules.
- ***VisualSoft XMLSecure*** is a powerful COM component that provides cryptographic features such as digital signatures, verification, encryption and decryption in any XML framework.
- ***Done Information Inc.'s XML based Done Signature*** software component has been developed for secure personal and business transactions.
- ***Intel 7210 XML Accelerator*** provides up to 3DES level of encryption (trusted XML transactions) and handles up to 600 secure connections per second (security without sacrificing speed).
- ***IAIK XML Signature Library*** is conceived as a toolkit which enables Java developers to easily integrate the creation and verification of XML based digital signatures into their applications.
- ***XMLDsig - Digital Signatures for XML Documents*** is an implementation of the W3C Candidate Recommendation for XML-Signature Syntax and Processing
- ***SecureXML*** is the first C-Language implementation of the W3C XML Digital Signature standard. Its small code size is highly optimized making it most appropriate for high volume XML transaction applications.
- ***Phaos XML Toolkit*** provides a comprehensive solution for creating secure, XML applications. And since Phaos XML is 100% Java, your XML applications can run anywhere Java runs - from enterprise servers to Web browsers.

- ***Verisign's simple XML-Signature package*** signs and verifies XML documents according to the latest W3C XML-Signature
- ***XML Security Library*** is a C library based on LibXML2 and OpenSSL. The library was created with a goal to support major XML security standards. It is released under the MIT Licence.

6. Conclusion

With the increasing use of XML in both industry and academia to deal with web data, security issues about such data will become more important than today. Hence, securing XML documents will be the key problem with the usage. In order to solve these kinds of problems, several research groups have been formed and worked on XML signatures and encryption particularly in terms of secure communications. The research efforts still go on with the development of increasing amount of tools and products utilizing XML signatures and encryption.

On the other hand, XML is also being utilized in order to provide better authentication, authorization and access services to the users. Due to fundamental limitations of HTML in terms of semantics and information structure, the proposals for security services are faced with the problems of HTML. XML represents an important opportunity to solve these problems and commercial products are becoming available which provide security features around XML.

Acknowledgements

The author acknowledges the help given him in preparing this article by the feature ' Enabling XML security' written by Murdoch Mactaggart.

7. References

[1] XML Security Page, "http://www.nue.et-inf.uni-siegen.de/~geuer pollmann/xml_security.html"
[2] Ernesto Damiani, Sabrina De Capitani, Stefano Paraboschi, Pierangela Samarati, "Securing XML Documents", EDBT 2000, pp121-135
[3] IBM AlphaWorks. XML Security Suite, April 1999, "http://www.alphaWorks.com/~tech/xmlsecuritysuite.
[4] W3C, XML Signature WG, "http://www.w3.org/Signature/"
[5] W3C, XML Encryption WG, "http://www.w3.org/Encryption/2001"
[6] Decryption Transform for XML Signature, "http://www.w3.org/TR/2002/CR-xmlencdecrypt-20020304"
[7] Murdoch Mactaggart, "Enabling XML Security", at "http://www-106.ibm.com/developerworks/security/library/s-xmlsec.html"
[8] XML-Signature Syntax and Processing, "http://www.w3.org/TR/xmldsig-core/"
[9] XML Encryption Syntax and Processing, "http://www.w3.org/TR/xmlenc-core/"