An Overview on XML Security Technologies

A. A. Abd El-Aziz*
Dept. IST, ISSR, Cairo University, Egypt.

* Corresponding author. Tel.: 01114907998; email: a.ahmed@cu.ed
Manuscript submitted December 23, 2015; accepted May 15, 2016.
doi: 10.17706/jcp.12.5.462-469

Abstract: XML has been widely adopted for information exchange across various networks due to flexibility providing common syntax for messaging systems. XML documents may contain private information that cannot be shared by all user communities. Therefore, XML Security has a great importance to the overall security of the distributed systems. In this paper, two XML security technologies, namely, XML signature and encryption have been presented. It presents an overview of how they integrate with XML in such a way, as to maintain the advantages and capabilities of XML, while adding the necessary security capabilities.

Key words: XML security, XML encryption, XML signature.

1. Introduction

There are two types of security ranges in Internet security, namely, Point-to-Point, and End-to-End security. The former ensures the security between two adjacent nodes of the network. The latter ensures the security from the initial sender until the final recipient, which is more secure for web services. The Secure Socket Layer (SSL) provides a Point-to-Point security. SSL is not suitable for the transmission modes of web services, such as the Transmission Control Protocol (TCP), the File Transfer Protocol (FTP), and messages formation. SSL can execute the encryption of complete information, but it cannot execute the encryption of partial information. SSL guarantees point-to-point security, but it does not guarantee end-to-end security. It provides confidentiality, authentication, and integrity. On the other hand, XML security is a representative of End-to-End security. XML security is flexible for the application, especially, of mobile web services that demand more flexible, customizable, and better-optimized security schemes. XML security provides the following services [1], [2], [3], and [4]:

1) Confidentiality: Ensuring that only the intended receiver will read the transmitted document, and others cannot access or copy the data.

2) Integrity: No change in the transmitted document from the source to the final destination.

3) Authenticity: Determining that, a user has a genuine identity.

4) Non-repudiation: The sender cannot disclaim his responsibility for sending the document.

Therefore, the concern of XML security has been raised to a significant level, focusing on methods and approaches, to secure the XML messages exchanged. This paper presents two XML security technologies, namely, XML signature and encryption. It presents an overview of how they integrate with XML in such a way, as to maintain the advantages and capabilities of XML, while adding the necessary security capabilities. The remainder of the paper is organized as the following: Section 2 represents XML signature. In Section 3, XML encryption is presented. Section 4 summarizes the conclusion.

2. XML Signature

XML signature (called XMLDsig, XML-DSig, or XMLSig) defines an XML syntax for digital signatures and is defined by W3C and IETF (The Internet Engineering Task Force) in 2002 [5] to create a highly extensible signature syntax that is tightly integrated with existing XML technologies. XML signature is a digital signature obtained by applying a digital signature operation to XML resources. XML signature is not limited to signing XML resources, however, as it can also be used to sign binary resources such as a JPEG-file. It provides similar functionality as PKCS#7 [6]. The existing technologies allow us to sign only a whole XML document. However, XML signature provides a means to sign the entire document, parts of a document, and multiple signatures written in the same document. This functionality is very important in a distributed multi party environment, where the necessity to sign only a portion of a document arises whenever changes and additions to the document are required. XML signature has been used to solve security problems, such as falsification, spoofing, and repudiation by ensuring confidentiality, integrity, authenticity, and non-repudiation. Submit your manuscript electronically for review.

2.1. Syntax

The structure of XML signature is depicted as in Fig. 1 [5]:

```
<Signature Id?>
  <SignedInfo Id?>
    <CanonicalizationMethod Algorithm/>
    <SignatureMethod Algorithm/>
    (<Reference Id? URI? Type?>
      (<Transforms>
        (<Transform Algorithm />)+
      </Transforms>)?
      <DigestMethod Algorithm />
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>
    <choice>
      <KeyName>*
      <KeyValue>*
    </choice>
  </KeyInfo>)?
  (<Object Id?>)*
</Signature>
```

Fig. 1. XML signature's structure.

In Fig. 1, the <Signature> is the root element of an XML signature. It is composed of one <SignedInfo>, one <SignatureValue>, zero or one <KeyInfo>, and zero or more <Object> elements. Moreover, it has an optional *Id* attribute, which allows the <Signature> to be referenced by other signatures or objects [5].

1) *Signed Information:* the required <SignedInfo> includes the data objects, which have been signed. The <SignedInfo> contains one <CanonicalizationMethod>, one <SignatureMethod>, and one or more <Reference> elements. The <SignedInfo> may include an optional *Id* attribute, which permits it to be referenced by other signatures or objects. The <SignedInfo> does not include explicit signature or digest properties, such as date/time of the signing process.

The required <CanonicalizationMethod> determines the canonicalization algorithm, which is used to canonicalize the <SignedInfo> before the proceeding of the signature calculations. It has a required attribute, called an *Algorithm*, which specifies the URI (Uniform Resource Identifier) of the used algorithm in the W3C specification.

The required <SignatureMethod> designates the algorithm used for creating the signature and performing the validation of the canonicalized <SignedInfo>. It includes an Algorithm attribute, which specifies the URI of the cryptographic algorithm.

The required <Reference> includes the digest method, and the digest value calculated over the identified data objects. The structure of the <Reference> contains optionally, an *Id* attribute, a URI attribute, and a *Type* attribute. Moreover, it contains zero or one <Transforms>, one <DigestMethod>, and one <DigestValue>. The optional *Id* attribute allows the <Reference> to be referenced by other signatures or objects. The optional *URI* attribute identifies a data object that will be signed, using a URI-Reference. For example, URI = "http://Company.com/emp.xml" means that, the emp.xml document will be signed. The *URI* attribute and <Transforms> describe the retrieving and preparation of the data objects, which will be digested (i.e., the input to the digest method). The optional *Type* attribute provides information about the data objects, obtained by the URI attribute, to ease the processing of the referenced data.

The optional <Transforms> contains one or more ordered <Transform> elements, which are applied to the original data objects before they are digested. These ordered elements describe the preparation of the original data objects before digesting. The transform operations include canonicalization, encoding/decoding, XSLT, XPath, XML schema validation, or XInclude, which are applied to the original data. The <Transform> includes a required attribute, called an *Algorithm*, which indicates the operation that will be applied to the original data objects. If the <Transforms> is omitted, the data objects are digested directly without any transformation. The output of each <Transform> is the input to the next one. The input to the first <Transform> is the original data objects obtained by the *URI* attribute of the <Reference>. The output from the last <Transform> is the input for the <DigestMethod>. When the transform operations are carried out on the data objects, the signing is not done on the original data objects but on the resulting transformed data objects.

The required <DigestMethod> specifies the digest algorithm that will be applied to the transformed data objects, to get the digest value that will be signed. The required <DigestValue> includes the encoded digest value, using base64 that will be signed.

2) *Signature Value*: the required <SignatureValue> includes the encoded value, using base 64 of the digital

signature.

3) *Key Information*: The optional <KeyInfo> specifies the needed key to the recipients to validate the signature. It includes *keys*, *names*, or *certificates*. The <KeyInfo> contains multiple <KeyName> or multiple <KeyValue> elements. Only one of them appears in the <KeyInfo>. If the recipient knows the key, the <KeyInfo> can be omitted. The <KeyName> includes a string value, and a whitespace is allowed. It is used to identify the key for the recipient. The <KeyValue> includes a public key value of the signer that is exploited in validating the signature. The <KeyValue> may include externally public key values as Parsed Character Data (PCDATA), or element types from an external namespace.

4) *Objec*t: Since the <SignedInfo> does not include explicit signature or digest properties, if an application needs to associate properties with the signature or digest, it may include such information in a <SignatureProperties> within an <Object>. The optional <Object> includes any data (signature properties) within the signature. The <Object> includes an optional *Id* attribute used to refer to it by the *URI* attribute of the <Reference> to be signed in the enveloping signature. Moreover, the *URI* attribute of a <Reference>

can refer to the *Id* attribute of the <SignatureProperties> in the <Object> to be signed in the enveloping signature.

2.2. Processing

In this work, XML documents are signed and verified by applying the following steps [7]:

1) Specify the location of each XML fragment that will be signed, and assign it to a *URI* attribute of a <Reference>.

2) Apply the ordered transform operations to each XML fragment that will be signed (optional).

3) Assign each used transform operation to an *Algorithm* attribute of a <Transform>.

4) Add the ordered list of the <Transform> in a <Transforms>.

5) Apply a hashing algorithm to each transformed XML fragment and get a digest value for each fragment.

6) Assign the used hashing algorithm to an *Algorithm* attribute of a <DigestMethod>.

7) Add each calculated digest value in a <DigestValue>.

8) Add the <Transforms>, <DigestMehod>, and <DigestValue> of each XML fragment to its <Reference>.

9) Add the <Reference> of each XML fragment to a <SignedInfo>. Hence, the <SignedInfo> contains all the digested XML fragments.

10) Assign a signature algorithm to an Algorithm attribute of a <SignatureMethod>.

11) Add the <SignatureMethod> and <CanonicalizationMethod> to the <SignedInfo>.

12) Apply the selected canonicalization process to the <SignedInfo>.

13) Apply the same hashing algorithm to the <SignedInfo> and get a digest value for it. Since the <SignedInfo> contains all the digested XML fragments, they are implicitly signed as well, when signing the <SignedInfo> using a digital signature algorithm, such as the RSA, DSA, or Elliptic Curve.

14) Apply the selected signature algorithm, using the signer's private key to the <SignedInfo> and get the signature value for the <SignedInfo>.

15) Add the signature value in a <SignatureValue>.

16) Add a <KeyInfo> to specify the signer's public key, which will be used to validate the signature (optional).

17) Add the <SignedInfo>, <SignatureValue>, and <KeyInfo> to a <Signature>, which is the final resulting signature. **To verify a signature, perform the following steps:**

1) Apply the ordered transform operations specified in the <Transforms>, which is within the <Reference> to each original XML fragment.

2) Apply the hashing algorithm specified in the <DigestMethod>, which is within the <Reference>, to each transformed XML fragment to be sure it is not changed, and get a digest value for each fragment.

3) For each transformed XML fragment, compare the calculated digest value with the one in the <DigestValue>, which is within the <Reference>. If they do not match, the signature verification fails. Otherwise, go to the next step.

4) Apply the canonicalization process specified in the <CanonicalizationMethod> to the <SignedInfo>.

5) Apply the hashing algorithm specified in the <DigestMethod> to the <SignedInfo>,to besure it is not changed.

6) Retrieve the digest value from the signature value, which is included in the <SignatureValue> using the signer's public key.

7) Compare the retrieved digest value with the calculated digest value. If they do not match, the signature verification fails. Otherwise, the signature verification succeeds.

3. XML Encryption

XML encryption is a specification developed by W3C specification in 2002 [8]. XML encryption defines a

process for encrypting data, decrypting encrypted data, and representing the result using syntax of XML. XML encryption is used to achieve the confidentiality to the parts of an XML document by hiding selected sensitive information in a message using cryptography algorithms. It can be used to encrypt an XML document, an XML element including attributes, XML element contents, or non-XML data. The main advantages of XML encryption is that, it provides the encryption of specific parts of an XML document, and multiple encryptions, which encrypt multiple parts of an XML document. XML encryption supports the confidentiality of the XML documents or parts of XML, documents by protecting the critical information, using cryptography algorithms. XML encryption can be used to encrypt web pages, when there are neither authentications, access control mechanisms, nor an end-to-end encryption scheme. Moreover, the use of XML Encryption assists in solving the security problems, such as XML data eavesdropping and the safety of the encrypted documents during transmission and storing of them [4]. **XML encryption and signature are similar in:**

1) The input, which is an XML document, parts of an XML document, or non-XML data (binary or digital data).

2)Both XML signature and encryption use the <KeyInfo>, which appears as a child of a <SignedInfo>,an<EncryptedData>,or an <EncryptedKey>. The <keyInfo> provides information to a recipient about the key, which will be used to validate signatures or decrypt encrypted data. **However, they differ in:**

1)The output in the case of multiple encryptions or multiple signatures; for multiple encryptions, XML encryption returns a separate XML element, called a<EncryptedData> for each encryption in the same XML document, where the <EncryptedData> contains only one encrypted part. For multiple signatures, the XML signature returns one XML element, called <Signature> that contains all the signatures.

2) In XML encryption, the <KeyInfo> may contain <KeyValue>, <KeyName>, or <RetrievalMethod>. However, in the XML signature, the <KeyInfo> contains either a <KeyValue>, <KeyName>, or <RetrievalMethod>. Only one of them appears in the <KeyInfo>, which is called the Required Choice.

3.1. Syntax

The structure of XML encryption is depicted in Fig. 2 [7]-[10]:

```
<EncryptedData Id? Type? MimeType? Encoding?>
  (<EncryptionMethod Algorithm />)?
  (<ds:KeyInfo>
      (<EncryptedKey>)?
      (<AgreementMethod >)?
      (<ds:KeyName>)?
       (<ds:RetrievalMethod URI Type? />)?
  <ds:/KeyInfo>)?
  <CipherData>
    <CipherValue> | <CipherReference URI >
  </CipherData>
  (<EncryptionProperties Target? Id?>)?
</EncryptedData>
```

Fig. 2. XML encryption's structure.

In Fig. 2, the <EncryptedData> is the root or result of XML encryption. When the encrypted data is an XML document, the <EncryptedData> becomes the root of the encrypted XML document. When the encrypted data is an XML element or element content, the <EncryptedData> is the result of XML encryption, because it replaces the element or content respectively, in the encrypted XML document. It presents the encryption content and the encryption keys of one resource. An XML document may contain zero or more <EncryptedData> elements, where each <EncryptedData> contains the encrypted data of only one resource.

The <EncryptedData> cannot be a parent or a child of another <EncryptedData>. The<EncryptedData> contains zero or one <EncryptionMethod>, zero or one <KeyInfo>, zero or one <EncryptionProperties>, and one <CipherData>. It has four optional attributes, namely, an *Id*, a *Type*, a *MimeType*, and an *Encoding*. The *Id* attribute allows it to be referenced by other encryptions or objects. The *Type* attribute identifies the type of the plaintext, which has been encrypted, such as *Type* = "element", *Type* = "element content", or *Type* = "attribute". The *MimeType* describes the media type of the encrypted data. The value of this attribute is a string, such as:

1) *Encryption Method*: The optional <EncryptionMethod> defines the encryption algorithm carried out on the resource to get the cipher data. It has a required attribute, called an Algorithm, which specifies the URI of the used algorithm in the W3C specification, such as the AES-128/256. If the element is omitted, the receiver must know the encryption algorithm; otherwise the decryption attempt will fail.

2) *Key Information*: The optional <ds:KeyInfo> includes the information to a recipient about the key (a private key of the recipient or shared key) used for decrypting the encrypted data in <EncryptedData> elements or encrypted symmetric keys in <EncryptedKey> elements. The <ds:KeyInfo> contains zero or one <ds:KeyValue>, zero or one <ds:KeyName>, zero or one <ds:RetrievalMethod>,or zero or one <EncryptedKey>.

The optional <ds:KeyValue> includes a plaintext key value (i.e., a secret key value in the case of symmetric encryption) needed for decrypting the encrypted data or encrypted keys. In asymmetric key encryption, the <ds:KeyValue> is not needed. In general, using the <ds:KeyValue> is not recommended. The optional<ds:KeyName> includes a name (a string value, and a whitespace is allowed) of the key used for decrypting the cipher data. It is necessary in asymmetric encryption to specify the name of the private key used for decrypting the cipher data. In symmetric encryption, since the receiver knows the shared key, it can be omitted.

The optional <ds:RetrievalMethod> is used to refer to an encrypted key information stored at another location outside an <EncryptedData>. It includes a *URI* attribute and a *Type* attribute. The required *URI* attribute is used to indicate the location of the encrypted key information. The optional *Type* attribute has the value "http://www.w3.org/2001/04/xmlenc#EncryptedKey". When an <EncryptedKey> is located outside an <EncryptedData>, the <ds:RetrievalMethod> with the attribute *Type* ="http://www.w3.org/2001/04/xmlenc#EncryptedKey" specifies the link of the <EncryptedKey> containing the encrypted symmetric key, required to decrypt the encrypted data in the <EncryptedData> or the encrypted key in another <Encryptedkey>. When the symmetric key is encrypted inside the <ds:KeyInfo> of an <EncryptedData>, the <RetrievalMethod> is not necessary. Either or both the <ds:KeyName> and <ds:RetrievalMethod> are used to identify the same key.

The optional <EncryptedKey> includes an encrypted symmetric key used for decrypting the encrypted data or encrypted keys. It may be a stand-alone XML document, placed within an application document, or included as a child element of a <ds:KeyInfo> of an <EncryptedData>. When an <EncryptedKey> is a stand-alone XML document or placed within an application, a <ds:RetrievalMethod> with an attribute *Type* = "http://www.w3.org/2001/04/xmlenc#EncryptedKey" in a <ds:KeyInfo> in an <EncryptedData> is used to specify the link of the <EncryptedKey>. The<EncryptedKey> and <EncryptedData> are both derived from the same abstract type, namely, an <EncryptedType>.

In addition to all the child elements of the <EncryptedData>, the <EncryptedKey> contains zero or one <ReferenceList> and zero or one <CarriedKeyName>. A<CipherData> of an <EncryptedKey> includes the symmetric key in an encrypted form, while the <KeyInfo> within the <EncryptedKey> provides information about the key of the receiver used for decrypting the secret key (in general, the key is a receiver's private key).

The optional <ReferenceList> includes references to data or keys encrypted using this encrypted symmetric key. The <ReferenceList> may refer to multiple encrypted data and encrypted keys, which are encrypted by this encrypted symmetric key. The <ReferenceList> refers to the encrypted data, using a required element, called a <DataReference> and to the encrypted keys, using a required element, called a <KeyReference>. When the same encrypted symmetric key encrypts multiple <EncryptedData> or <EncryptedKey> elements, multiple <DataReference> or <KeyReference> elements occur inside the <ReferenceList>.

The optional <CarriedKeyName> of an <EncryptedKey> includes the name of the encrypted symmetric key that has encrypted data or keys referenced in the <ReferenceList>. The whitespace is allowed in a <CarriedKeyName>. The name of a <CarriedKeyName> is similar to the name of a <ds:KeyName> within a <ds:KeyInfo> in an <EncryptedData>. When the same encrypted symmetric key encrypts multiple <EncryptedData> or <EncryptedKey> elements, the same <CarriedKeyName> may occur multiple times. The plaintext key values of all <EncryptedKey> elements, which have the same <CarriedKeyName> within a single XML document, are the same.

3) *Cipher Data*: The required <CipherData> of the <EncryptedData> includes the encrypted data. The encrypted data may be included explicitly or referenced, if it is stored in an external location. The <CipherData> contains either one <CipherValue> or one <CipherReference> (Required Choice). The <CipherValue> includes the encrypted data directly. The <CipherReference> refers to the encrypted data, if it is stored in an external location. The <CipherReference> includes a required URI attribute used to indicate the location of the encrypted data.

4) Encryption Property: The optional <EncryptionProperty> associates properties with the <EncryptedData> or <EncryptedKey>. It may include information, such as the date/time of the encryption process, and the type of the encrypted object. It includes an optional Target attribute, which identifies whether the type of an encrypted object is data or a key.

3.2. Processing

In this work, the XML documents are encrypted by applying the following steps [7]:

1) Select an algorithm for the encryption and assign the used encrypting algorithm to an *Algorithm* attribute of an <EncryptionMethod>.

2) Generate a random key. In asymmetric encryption, the key pair is a public key for the encryption and a private key of a recipient for the decryption. In a symmetric encryption, the key is a secret key. If the decryption keys (the private key of the recipient or secret key) are identified by a *name* or a *URI*, construct for them a <ds:KeyInfo>, which may contain a <ds:KeyName>, <ds:KeyValue>,or <ds:RetrievalMethod>.If an asymmetric algorithm is to encrypt the symmetric key, construct an <EncryptedKey> by recursively applying this encryption process. The <EncryptedKey> may be a child of the <ds:KeyInfo>, or it may be stored independently and be identified by a <ds:RetrievalMethod> with an attribute *Type* = ”http://www.w3.org/2001/04/xmlenc#EncryptedKey”.

3) Encrypt the data using the selected (symmetric/asymmetric) algorithm and the generated secret/public key.

4) Store the encrypted data either in a <CipherValue> or externally, and refer to the encrypted data by a <CipherReference>.

5) Construct a <CipherData> from the <CipherValue> or <CipherReference>.

6) Add the <EncryptionMethod>, <ds:KeyInfo> (optional) , and <CipherData> to an <EncryptedData>, which is the final encryption result. **The following steps decrypt the encrypted data:**

1) Investigate the <EncryptedData> to determine the used encryption algorithm, and the <ds:KeyInfo> to determine the decryption key (the private key of the recipient or secret key) that will be used for

decrypting the cipher data. used.

2) If the decryption key is an encrypted symmetric key, investigate the <ds:KeyInfo> of the <EncryptedKey> to determine the decryption key (in general, it is the private key of the recipient) that will be used for decrypting the encrypted symmetric key.

3) Decrypt the data, which is contained in the <CipherData>.

4. Conclusion

The paper shows the background information about the XML signature and encryption in detail. For the XML signature, it explains the XML signature types, syntax of the XML signature, and canonicalization. Moreover, it shows the steps of the XML signature and verification processes. For XML encryption, it explains the syntax of XML encryption. In addition, it shows the steps of the XML encryption and decryption processes.

References

[1] Yi, Z., Yuan, D., *et al.* (2011). XML-based model for data communication encryption. *Proceedings of the 3rd International Conference Computer Research and Development* (pp. 258–260).

[2] Knap, T., *et al.* (2009). Towards more secure web services – exploiting and analysing XML signature security issues. *Proceedings of the 3rd IEEE International Conference on Research Challenges in Information Science (RCIS)* (pp. 49–58).

[3] Yong, G., *et al.* (2010). Web services security based on XML signature and XML encryption. *Proceedings of Journal of Networks* (pp. 1092–1097).

[4] Nordbotten, N. A. (2009). XML and web services security standards. *IEEE Communications Surveys & Tutorials*, 4–21.

[5] Bartel, M., Boyer, J., Fox, B., LaMacchia, B., & Simon, E. (12 Feb., 2002). XML-Signature Syntax and Processing. *W3C Recommendation*.

[6] Kaliski, B. (Mar. 1998). PKCS#7: Cryptographic Message Syntax Standard, Version 1.5, IETF RFC 2315.

[7] Liu, B., & Chen, H. (July 2011). Implementation a prototype of XML security for certificate management. *Proceedings of the International Conference on Control, Automation and Systems Engineering* (pp. 1–6, 30-31).

[8] Imamura, T., Dillaway, B., Yiu, K., & Nystrm, M. (24 Dec., 2002). XML encryption syntax and processing. W3C Recommendation.

[9] Hashizume, K., & Fernandez, E. B. (28-30 Aug., 2009). Symmetric encryption and XML encryption patterns. *Proceedings of the 16th Conference on Pattern Languages of Programs* (pp. 1–8), Chicago, IL, USA.

[10] Qiao, J. (2007). Research on XML united-signature technology and its implementation. *Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* (pp. 979–983).

A. A. Abd El-Aziz has completed his Ph.D. degree in June 2014 in information science & technology from Anna University, Chennai-25, India. He had received the B.Sc., and master of computer science in 1995 and 2006 respectively from the Faculty of Science, Cairo University. Now, he is an assistant professor in the ISSR, Cairo University, Egypt. He has 12 years' experience in teaching at Cairo University, Egypt. His research interests include database system, database security, XML security, cloud computing, big data, data mining, and social network analysis. He has published about 30 research papers in international conferences and journals.