CHAPTER 3

XML SIGNATURE AND ENCRYPTION

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 (Sun & Li 2008, Knap & Ml´ynkov´a 2009a, Yue-sheng et al 2010, and Hao-yu et al 2011):

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 chapter 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.

3.1 XML SIGNATURE

The XML signature, called XMLDsig, XML-DSig, or XML-Sig, defines the XML syntax for digital signatures, and is defined by W3C and the Internet Engineering Task Force (IETF) in Bartel et al (2013). The XML signature creates a highly extensible signature syntax, which is integrated tightly with the existing XML technologies. The XML signature is a digital signature obtained by applying a digital signature operation to XML resources. Moreover, the XML signature can be applied to any digital data or binary data, such as a JPEG-file. Its functionality is similar to that of the PKCS\#7 (Kaliski 1998).

The existing technologies allow us to sign only a whole XML document. However, the XML signature provides a means to sign the entire document, parts of a document, or 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. The XML
signature has been used to solve security problems, such as falsification, spoofing, and repudiation, by ensuring confidentiality, integrity, authenticity, and non-repudiation.

### 3.1.1 Syntax

The structure of the XML signature document is depicted in Figure 3.1, as defined by Bartel et al (2013):

```xml
<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>

Figure 3.1 XML Signature's Structure
In Figure 3.1, 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 (Bartel et al 2013).

### 3.1.1.1 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.
3.1.1.2 Signature Value

The required <SignatureValue> includes the encoded value, using base 64 of the digital signature.

3.1.1.3 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 <KeyInfo> is optional for the following reasons:

1. The signer may not want to disclose the key.
2. The key information may exist implicitly in the application's context.

If the signer needs to attach the key information to the signature, a <Reference> can get and contain the <KeyInfo> as part of the signature. The <Reference> is used, because the <KeyInfo> is outside of the <SignedInfo>. 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.

3.1.1.4 Object

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.

3.1.2 XML Signature Types

There are three types of XML signature, namely, enveloping, enveloped, or detached signature (Qiao 2007, Bartel et al 2013). In the enveloping signature, the <Signature> contains the signed data; the signed data is included within an <Object> or a <SignatureProperties>. The URI attribute of the <Reference> identifies the data by referring to the Id attribute of the <Object> or of the <SignatureProperties>. An example of the enveloping signature is depicted in Figure 3.2:

```xml
<Signature Id = "MySignature"
         xmlns = "http://www.w3.org/2000/09/xmldsig#" >

<SignedInfo>
  <CanonicalizationMethod
   Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
   Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference URI = "#obj">
    <DigestMethod
```

Figure 3.2 (Continued)
Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1" />

<DigestValue>
dGhpcyBpcyBub3QgYSBzaWdfhksaksm
</DigestValue>

</Reference>
</SignedInfo>

<SignatureValue>
C0CFFrVLtR
</SignatureValue>

<KeyInfo>... </KeyInfo>

<Object Id = "obj">
  XML security
  <SignatureProperties>
    <SignatureProperty Id = "TimeStamp" Target = "#MySignature">
      <TimeStamp xmlns = "http://www.ietf.org/rfcXXXX.txt">
        <date>4-3-2013</date>
        <time>15:43:30</time>
      </TimeStamp>
    </SignatureProperty>
  </SignatureProperties>
</Object>
</Signature>

Figure 3.2 Enveloping XML Signature

In Figure 3.2, the data referenced in the <Reference> by URI = "#obj" is the value of the Id attribute of the <Object> to get the data residing in it that have been signed.

In the enveloped signature, the signed data is an XML document, containing the <Signature> as its child element. The Signature calculation must be performed on the XML document except the content of the
<Signature>; thus the content of the <Signature> is excluded from the calculations of the data digest and signature value, by using the enveloped-signature transform whose identifier is "http://www.w3.org/2000/09/xmldsig#enveloped-signature". An example of the enveloped signature is depicted in Figure 3.3:

```
<Message>
 <To> OMAR </To>
 <Body> Take Care </Body>
 <From> Zizo </From>
 <Signature xmlns = "http://www.w3.org/2000/09/xmldsig#" >
    <SignedInfo>
       <CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
       <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       <Reference URI = ""/>
       <Transforms>
          <Transform Algorithm = "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
       </Transforms>
       <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>...</DigestValue>
    </SignedInfo>
    <SignatureValue>
       gjggjmdorldmfl
    </SignatureValue>
    <KeyInfo>...</KeyInfo>
 </Signature>
</Message>
```

Figure 3.3 Enveloped XML Signature
In Figure 3.3, the data referenced in the <Reference> by URI = "" is the entire <Message> (all the document) including the <Signature> itself. However, the value "http://www.w3.org/2000/09/xmldsig#enveloped-signature" of the Algorithm attribute of the <Transform> ensures that, the <Signature> and its content are excluded from the signature processing.

In the detached signature, the signature is carried out on external XML data outside the <Signature> or on local data objects that reside in the same XML document. The external data can be identified by the URI attribute of the <Reference>. For the local data objects, the <Signature> and local data objects will be two sibling elements in the same document. The following figures are two examples of the detached signature for external and local data:

```xml
<SignedInfo>
  <CanonicalizationMethod  Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod  Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference  URI = "http://www.w3.org/TR/2000/REC-xhtml1-2000112">
    <DigestMethod  Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue> flfjfldjfcyBpcyBub3QgYSBzaWduY </DigestValue>
  </Reference>
</SignedInfo>

SignatureValue KHGBNMLO </SignatureValue>
</Signature>
```

**Figure 3.4 Detached XML Signature for External Data**
In Figure 3.4, the data referenced in the <Reference> by URI = "http://www.w3.org/TR/2000/REC-xhtml1-2000112" is the external data that have been signed.

```xml
<Signature xmlns = "http://www.w3.org/2000/09/xmlsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
    <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
    <Reference URI = "#localdata">
      <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmlsig#sha1"/>
      <DigestValue> dfjkdhfkdfhjkdffhd </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue> jdhasjdjasdhjaskd </SignatureValue>
</Signature>

<LocalData Id = "localdata">
  <title> XML security </title>
  <author> Prof. A. Kannan </author>
</LocalData>
```

**Figure 3.5 Detached XML Signature for Local Data**

In Figure 3.5, the <Signature> and the <LocalData>, which is the local signed data, are sibling elements in the XML document.

### 3.1.3 Canonicalization

The contents of digital signatures must be identical on signature application and verification. Otherwise, the digital signatures will be invalid.
In the XML digital signature, slight changes to the structure of the signed XML document are acceptable, as long as the document's contents keep identical to an XML parser. For example, XML comments or whitespaces between an element's attributes are not considerable to an XML parser. Therefore, these modifications will not affect the validation of the signature. This flexibility of XML signature is due to the concept of canonicalization.

The canonicalization is a set of transformations that are applied to the XML contents before digesting. The purpose of canonicalization is to hide the irrelevant modifications to the XML document, such as removing insignificant whitespaces, sorting attributes, sorting namespace declarations of each element, and adding fixed attributes according to the DTD file that will not affect the verification of the signature (Jensen et al 2009, Knap & Ml´ynkov´a 2009b).

In other words, XML canonicalization is significant, because two XML documents may be logically equivalent, but differ in their physical representations. The differences in their physical representations may be due to syntactic changes, permitted by XML and namespaces in XML. These differences in their physical representations will lead to different digest values of such XML documents, although they have the same content. Therefore, the canonicalization methods define a physical representation, namely, the canonical form, and the logically equivalent documents will be transformed to the canonical form, to have the same physical representation (Nordbotten 2009). For example, let there be two XML documents, namely, Doc1 and Doc2 containing an element B, which has two attributes, A1 and A2. Doc1 has a comment element, and defines attributes A1 and A2 respectively. However, Doc2 defines the attributes in the reverse order; i.e., attribute A2 and then attribute A1. The physical form of these two XML documents is different, but the logical form is the same. Hence, the canonicalization process
will transform these two logically equivalent XML documents to the same physical form. Therefore, the digesting process of the two XML documents will return the same values (Knap & Ml’ynkov´a 2009b).

There are three W3C Recommendations for the normalization process, namely, XML Canonicalization 1.0 (Can1.0) (Boyer 2001), XML Canonicalization 1.1 (Can1.1) (Boyer & Marcy 2008), and Exclusive XML Canonicalization 1.0 (ECan) (Boyer et al 2002). ECan is based either on Can1.0 (denoted as ECan1.0) or on Can1.1 (denoted as ECan1.1). Can1.0 is the original W3C Recommendation, which defines the whole normalization process. Can1.1 is an enhanced version of Can1.0, which addresses the issues with the processing of the attributes, namely, id and base from the XML namespace. Can1.1 and ECan1.1 differ in the processing of the declared namespaces. Can1.1 includes the result of the normalization of an XML element that will be signed by all the namespace declarations in its parent element, although its parent element will not be signed. However, ECan1.1 contains only those namespace declarations, which are required for parsing the signed contents, and to have only their namespace declarations become embedded in the signed document. Hence, in ECan1.1, the namespace declarations of the enclosing XML element are not involved in the result of the normalization process, unless they are used in the signed XML element.

### 3.1.4 Processing and Implementations

In this work, XML documents are signed and verified by applying the following steps (Liu & Chen 2011):

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>, <DigestMethod>, 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 canonicalization process to an Algorithm attribute of a <CanonicalizationMethod>.

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

12. Add the <SignatureMethod> and <CanonicalizationMethod> to the <SignedInfo>.

13. Apply the selected canonicalization process to the <SignedInfo>.
14. 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.

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

16. Add the signature value in a <SignatureValue>.

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

18. Add the <SignedInfo>, <SignatureValue>, and <KeyInfo> to a <Signature>, which is the final resulting signature.

To verify a signature, we 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 be sure 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.

The abstract algorithms, such as MD5 or SHA-1, are used to calculate the hash value for a given input. In this work, the signature process is applied to the digest value, since the digest value is less than the original data. Therefore, applying the signature process has reduced the signing time and the space of the memory required for the data. Hence, the transmission of the data is effective. The digest value used in this work represents the fingerprint of the original data. If any slight modifications occurred to the original data, the digest value will have a huge change, because of the avalanche effect property of the abstract algorithm. After the digest value has been signed, the XML documents cannot be changed. Hence, the integrity of the documents can be fulfilled (Yue-sheng et al 2009).

Many library function tools assist researchers to implement the XML signature. They are (Haron et al 2010):

1. Java XML Digital Signature API: Sun provides a Java development kit for the developer to apply W3C recommendations for XML security technology specification,
by translating XML security to Java Specification Request 105 (JSR 105), which is Java APIs. The JSR 105 is a collection of technical specifications developed in Java and designed to conform to the XML signature specifications.

2. Apache XML Security Library: Apache XML security (http://santuario.apache.org) is an open source initiative for XML security launched under Apache Software license. It is developed in C++ and Java and provides the W3C specifications for the XML signature.

3. XML Security Library: The XML Security Library known as the XMLSec library (Sanin et al 2013) is an open source initiative to enable the integration of XML security inside a third party solution. It is developed in C++, launched under MIT license, and provides the W3C specifications for the XML signature in both Windows and Linux operating systems.

4. XML Digital Signature Tool Application: XML Digital Signature Tool (Mazumdar 2007) is a Firefox extension for applying the digital signature to XML documents. The XML interface is built on top of the Apache XML Security library. It is developed in C++ and Java Script languages and supports the W3C Specification for the XML signature.

5. XML Security Tool Application: The XML security tool plug-in (Schadow 2005) for Eclipse permits the users to utilize the XML security features. This application is launched under Eclipse Public License. This tool aims to ease the exploiting of the XML technology. The XML interface is built on top of the Apache XML Security library using Java.

7. **JVNRSS XML Signature: SIG rdf (RDF with XML signature)** (Terada et al 2006) is an application to promote the use of the XML signature within a security information exchange.

### 3.1.5 Comparison with Traditional Digital Signature

The limitations of the traditional digital signature are as follows (Chen-xi et al 2010, Yue-sheng et al 2010):

1. The size of the signature is very large, and the unit is only a document.

2. The traditional digital signatures do not provide electronic documents, multiple signatures, and signatures of parts of the document.

3. The tradition digital signature returns a string primitive or binary data.

4. The traditional digital signature sends the signature value and the original message to the verifier for authentication, which affects the capacity of the network and affects the efficiency of signature confirmation.

5. The traditional digital signature uses the X.509 certificate, to express the type and value of the signature key.
On the other hand, the XML digital signature solves the above limitations by the following:

1. The size of the XML signature is very small, and it applies the digital signatures to element-level granularity.
2. The XML signature provides a means to sign the entire document, parts of a document, or multiple signatures written in the same document.
3. The XML signature returns a <Signature>, which is an XML element.
4. The XML signature is suitable for the distributed network.
5. The XML signature uses the <KeyInfo> to present easily and explicitly all the information of the signature key.

### 3.2 XML ENCRYPTION

XML encryption is a specification developed by the W3C Consortium (Imamura et al 2013). XML encryption (Selkirk 2001b, Ardagna et al 2007, and Yue-sheng et al 2010) describes a process for encrypting and decrypting data, and representing the result of the encryption, encryption algorithm, and encryption key, using the syntax of XML. The input data to the XML encryption process is an XML document, an XML element, an XML element contents, or non-XML data, such as binary 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. 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 an <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 a <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.
3.2.1 XML Encryption Types

XML Encryption provides two types of encryption algorithms, namely, symmetric and asymmetric (Hashizume & Fernandez 2009, Shahgholi et al 2011, and Hao-yu et al 2011). The symmetric encryption algorithms, such as the Data Encryption Standard (DES) or Advanced Encryption Standard (AES) use a common key, called the symmetric (secret, single, or shared) key for both encryption and decryption.

In symmetric encryption, both the sender and receiver must know the same secret key. The sender uses the secret key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver uses the same secret key to decrypt the message and returns the plaintext. Although the symmetric encryption is fast, the distribution of the key is a hard problem, because it requires a secure channel to exchange the secret key between the sender and receiver.

The asymmetric encryption algorithms, such as the RSA use the key pair (public key and private key) of the recipient. These public and private keys are linked mathematically, and the private key cannot be derived from its public key.

In asymmetric encryption, the sender encrypts a message using the receiver's public key, and the receiver uses his private key to decrypt the encrypted message. In asymmetric encryption, there is no need for key distribution, because the public keys are available to all the users, which makes the system strong. However, asymmetric encryption is slower than symmetric encryption.

When asymmetric and symmetric encryptions are used together, the encryption will be more effective. The symmetric key is used to encrypt the content, and then it is encrypted, using an asymmetric encryption algorithm.
Both the encrypted content and the encrypted symmetric key are sent to the recipient. The recipient will use his private key to decrypt the symmetric key, which itself is used to decrypt the content. This integration of the advantages of symmetric and asymmetric encryptions can conquer the key distribution problem of symmetric encryption, and the long time problem of asymmetric encryption. In both types of encryption, only recipients who own the shared key or the private key, that matches the public key used in the encryption process, can read the encrypted message after the decryption.

3.2.2 Syntax

The structure of the XML encryption document is depicted in Figure 3.6 (Hashizume & Fernandez 2009, Nordbotten 2009, Liu & Chen 2011, and Imamura et al 2013):

```xml
<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>
```

Figure 3.6 XML Encryption's Structure
In Figure 3.6, 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 "image".

3.2.2.1 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.
3.2.2.2 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.

### 3.2.2.3 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.

### 3.2.2.4 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.

Figures 3.8, 3.9, 3.10, and 3.11 present examples of an XML encryption's structure:
Figure 3.7 XML Document for Students' Courses

Figure 3.7 presents the courses and grades of students. Because the grade information is sensitive, the <Course> is encrypted, and is depicted in Figure 3.8:

```xml
<? xml version = "1.0"?>
<Student>
  <Name> Abd El-Aziz </Name>
  <Semester>
    <Number> I </Number>
    <Course>
      <Name> XML security </Name>
      <Grade> A </Grade>
    </Course>
  </Semester>
</Student>
```

**Figure 3.8 Encryption of the Course Element**
In Figure 3.8, the Triple DES, which is a symmetric algorithm encrypts the <Course>; the name of the symmetric key is Omar (i.e., the Key with the name Omar is used for the encryption and decryption), and the cipher value is B34C5ACD6. The encryption of the <Course>’s contents, which are of type Content is depicted in Figure 3.9:

```xml
<? xml version = "1.0">
<Student>
  <Name> Abd El-Aziz </Name>
  <Semester>
    <Number> I </Number>
    <Course>
      <EncryptedData
        Type = "http://www.w3.org/2001/04/xmlenc#Content"
        xmlns = "http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
          Algorithm = "http://www.w3.org/2001/04/xmlenc#
                      tripledes-cbc"/>
        <ds:KeyInfo
          xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
          <ds:KeyName> Omar </ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
          <CipherValue> AC3F4C5AC </CipherValue>
        </CipherData>
      </EncryptedData>
    </Course>
  </Semester>
</Student>
```

Figure 3.9 Encryption of the Course Element's Contents
The encryption of the `<Grade>`'s contents, which are character values, is depicted in Figure 3.10:

```xml
<?xml version="1.0"?>
<Student>
    <Name> Abd El-Aziz </Name>
    <Semester>
        <Number> I </Number>
        <Course>
            <Name> XML security </Name>
            <Grade>
                <EncryptedData
                    Type = "http://www.w3.org/2001/04/xmlenc#Content"
                    xmlns = "http://www.w3.org/2001/04/xmlenc#">
                    <EncryptionMethod
                        Algorithm = "http://www.w3.org/2001/04/xmlenc#
                        tripledes-cbc"/>
                    <ds:KeyInfo
                        xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
                        <ds:KeyName> Omar </ds:KeyName>
                    </ds:KeyInfo>
                    <CipherData>
                        <CipherValue> AC3GRTY45C </CipherValue>
                    </CipherData>
                </EncryptedData>
            </Grade>
            </Course>
        </Semester>
    </Course>
</Student>
```

Figure 3.10 Encryption of the Grade Element's Contents
When an asymmetric encryption algorithm encrypts the symmetric key used to encrypt the grade value, XML encryption will be given, as in Figure 3.11:

```xml
<? xml version = "1.0" >
<Student>
  <Name> Abd El-Aziz </Name>
  <Semester>
    <Number> I </Number>
  </Semester>
  <Course>
    <Name> XML security </Name>
    <Grade>
      <EncryptedData  Id = "ED"
        xmlns = "http://www.w3.org/2001/04/xmlenc#content"
        xmlns:xml = "http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
          Algorithm = "http://www.w3.org/2001/04/xmlenc#
            aes128-cbc"/>
        <ds:KeyInfo
          xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
          <RetrievalMethod
            URI = "#EK"
            xmlns = "http://www.w3.org/2001/04/xmlenc#
            EncryptedKey"/>
          <ds:KeyName> Omar </ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
          <CipherValue> AC3GRTY45C </CipherValue>
        </CipherData>
      </EncryptedData>
      <EncryptedKey  Id = "EK"
```

*Figure 3.11 (Continued)*
In Figure 3.11, the content of the <Grade> is encrypted with a symmetric key associated with the name Omar. The symmetric key is encrypted using the RSA, which is an asymmetric encryption algorithm, and it will be decrypted, using the private key associated with the name Ahmed. The <ReferenceList> refers to the <EncryptedData Id = "ED"/> encrypted using the symmetric key, named Omar. The reference is done using the <DataReference>. The <CarriedKeyName> points to the symmetric key
Omar used to encrypt the data in the <EncryptedData Id = "ED">. Since the symmetric key, named Omar is used for the encryption and decryption, the value of the <CarriedKeyName> is similar to the value of the <KeyName>.

3.2.3 Processing and Implementations

In this work, the XML documents are encrypted by applying the following steps (Liu & Chen2011):

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 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.

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>.

Many library function tools assist researchers to implement the XML encryption. They are (Haron et al 2010):

1. Apache XML Security Library: Apache XML security (http://santuario.apache.org) is an open source initiative for XML security launched under Apache Software license. It is
developed in C++ and Java and supports the W3C specifications for XML encryption.

2. XML Security Library: The XML Security Library or known as the XMLSec library (Sanin et al 2013) is an open source initiative to enable the integration of XML security inside a third party solution. It is developed in C++, launched under MIT license, and supports the W3C specifications for XML encryption in both Windows and Linux operating systems.

3. XML Security Tool Application: The XML security tool plug-in (Schadow 2005) for Eclipse allows the users to exploit the XML security features. This application is released under Eclipse Public License and is intended to bridge the lack of user acceptance in the XML technology. The XML interface is built on top of the Apache XML Security library using Java.


3.2.4 Comparing with Traditional Encryption

The limitations of traditional encryption are as follows: (Yue-sheng et al 2010):

1. The traditional encryption algorithms apply the encryption to data using a single key.
2. The SSL provides Point-to-Point security. Therefore, it is not suitable for the transmission modes of web services, such as the TCP, the FTP, and messages formation.

3. The SSL can execute the encryption for the complete information, but it cannot do it for the partial information.

On the other hand, XML encryption solves the above limitations by the following:

1. The XML encryption algorithm uses many symmetrical or asymmetrical keys to achieve the element level encryption.

2. XML encryption provides End-to-End security for the application procedure. Hence, it is flexible for the applications, especially, of mobile web services that demand more flexible, customizable, and better-optimized security schemes.

3. XML encryption provides the encryption of specific parts of an XML document and multiple encryptions, which encrypts multiple parts of an XML document.

3.3 SUMMARY

This chapter 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, lists some of the function tools for the XML signature, and compares the XML and traditional signatures.
For XML encryption, it explains the XML encryption types, syntax of XML encryption. In addition, it shows the steps of the XML encryption and decryption processes, lists some of the function tools for XML encryption, and compares the XML and traditional encryption processes.