Exploring XML Encryption, Part 1

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

XML Encryption provides end-to-end security for applications that require secure exchange of structured data. XML itself is the most popular technology for structuring data, and therefore XML-based encryption is the natural way to handle complex requirements for security in data interchange applications. Here in part 1 of this two-part series, Bilal explains how XML and security are proposed to be integrated into the W3C's Working Draft for XML Encryption.

Currently, Transport Layer Security (TLS) is the de facto standard for secure communication over the Internet. TLS is an end-to-end security protocol that follows the famous Secure Socket Layer (SSL). SSL was originally designed by Netscape, and its version 3.0 was later adapted by the Internet Engineering Task Force (IETF) while they were designing TLS. This is a very secure and reliable protocol that provides end-to-end security sessions between two parties. XML Encryption is not intended to replace or supersede SSL/TLS. Rather, it provides a mechanism for security requirements that are not covered by SSL. The following are a two important areas not addressed by SSL:

- Encrypting part of the data being exchanged
- Secure sessions between more than two parties

With XML Encryption, each party can maintain secure or insecure states with any of the communicating parties. Both secure and non-secure data can be exchanged in the same document. For example, think of a secure chat application containing a number of chat rooms with several people in each room. XML-encrypted files can be exchanged between chatting partners so that data intended for one room will not be visible to other rooms.

XML Encryption can handle both XML and non-XML (e.g. binary) data. We'll now demonstrate a simple exchange of data, making it secure through XML Encryption. We'll then slowly increase the complexity of the security requirements and explain the XML Encryption schema and the use of its different elements.

A simple example of secure exchange of XML data

Suppose you want to send the XML file in Listing 1 to a publishing company. This file contains details of a book that you want to purchase. In addition, it also contains your credit card information for payment. Naturally, you would like to use secure communication for this sensitive data. One option is to use SSL, which secures the whole communication. The alternative is to use XML Encryption. As already mentioned, XML Encryption is not an alternative to SSL/TLS. If the application requires that the whole communication be secure, you'll use SSL. On the other hand, XML Encryption is the best choice if the application requires a combination of secure and insecure communication (which means that some of the data will be securely exchanged and the rest will be exchanged as is).

Listing 1. The sample XML file to be encrypted

```
<purchaseOrder>
        <Order>
                <Item>book</Item>
                <Id>123-958-74598</Id>
                <Quantity>12</Quantity>
        </Order>
        <Payment>
                <CardId>123654-8988889-9996874</CardId>
                <CardName>visa</CardName>
                <ValidDate>12-10-2004</ValidDate>
        </Payment>
</purchaseOrder>
```

> **Note:** We have intentionally kept the XML file in Listing 1 very simple. This helps in keeping our focus on encryption-related issues. Real-world XML files in collaborative commerce or Web services will be similar in structure but more verbose. WSDL (Web Services Definition Language) and SOAP (Simple Object Access Protocol) are XML-based grammars that are frequently used in B2B integration. Both WSDL and SOAP can use XML Encryption to provide secure communication across the enterprise. Visit the W3C for details about them (see Resources).

Encrypting complete documents with XML Encryption
XML Encryption offers various options. Listing 2, Listing 3, and Listing 4 show the different encrypted results. Let's look at them in detail, one by one.

Listing 2 shows the resulting XML-encrypted file, in case you decide to encrypt the entire XML document in Listing 1. Notice the `<CipherData>` and `<CipherValue>` tags. The actual encrypted data appears as contents of the `<CipherValue>` tag. The complete `CipherData` element appears within an `EncryptedData` element. The `EncryptedData` element contains the XML namespace used for encryption. For example, your original data before encryption was XML and the official type definition by the Internet Assigned Numbers Authority (IANA) for XML is http://www.isi.edu/in-notes/iana/assignments/media-types/text/xml. This appears as the value of the `Type` attribute. XML Encryption uses the type definitions by IANA for various popular data formats such as RTF, PDF, and JPG. Refer to their Web site for complete details (see Resources). If you have special application data types (perhaps your own DTDs or XSDs that belong to your company's content management system), you can specify them in the `Type` attribute of `EncryptedData` element. The other attribute, xmlns, specifies the XML Encryption namespace that we used to encrypt the XML data.

Encrypting a single element with XML Encryption
You may want to encrypt only one element in Listing 1 -- for example, the `Payment` element. In this case, the result is illustrated in Listing 3. Compare Listing 2 and Listing 3 and you'll find the following differences:

1. Listing 2 contains only XML Encryption's schema, while Listing 3 contains both XML Encryption as well as elements from the original data in Listing 1. In Listing 3, the XML Encryption is embedded inside the user's XML.
2. Listing 3 also has a `Type` attribute in `<EncryptedData>`, but its value is http://www.w3.org/2001/04/xmlenc#Element. We are no longer using the IANA type; instead, we are using the type that XML Encryption has specified.
3. Note particularly the fragment #Element at the end that means EncryptedData -- this represents one element.

Encrypting the content of an element
Listing 4 will be the result if you want to encrypt only the content in `CardId`, an element in Listing 1. This time, we have used http://www.w3.org/2001/04/xmlenc#Content as the `Type` attribute value. We use this value whenever we have to encrypt only the content.

Encrypting non-XML data
What if you want to send, say, a JPEG file through XML Encryption? Listing 5 is a typical file that will result. The complete JPEG file in an encrypted sequence of bytes will appear as the content of the `CipherValue` element. Notice that there is only one difference between Listing 2 and Listing 5: the `Type` attribute of the `EncryptedData` element. Listing 5 includes the IANA type for the JPEG format. Similarly, you can encrypt any format by providing IANA values (refer to the IANA Web site, see Resources).

Keys for XML Encryption

In Listings 1 through 5, we have demonstrated encryption, which is not possible without keys (see the sidebar Public, private, and secret keys). With XML Encryption, all key-related issues are divided into two parts:

- Exchange of keys (asymmetric encryption)
- Using keys that were previously exchanged (symmetric encryption)

This way, users can exchange keys and use them later.

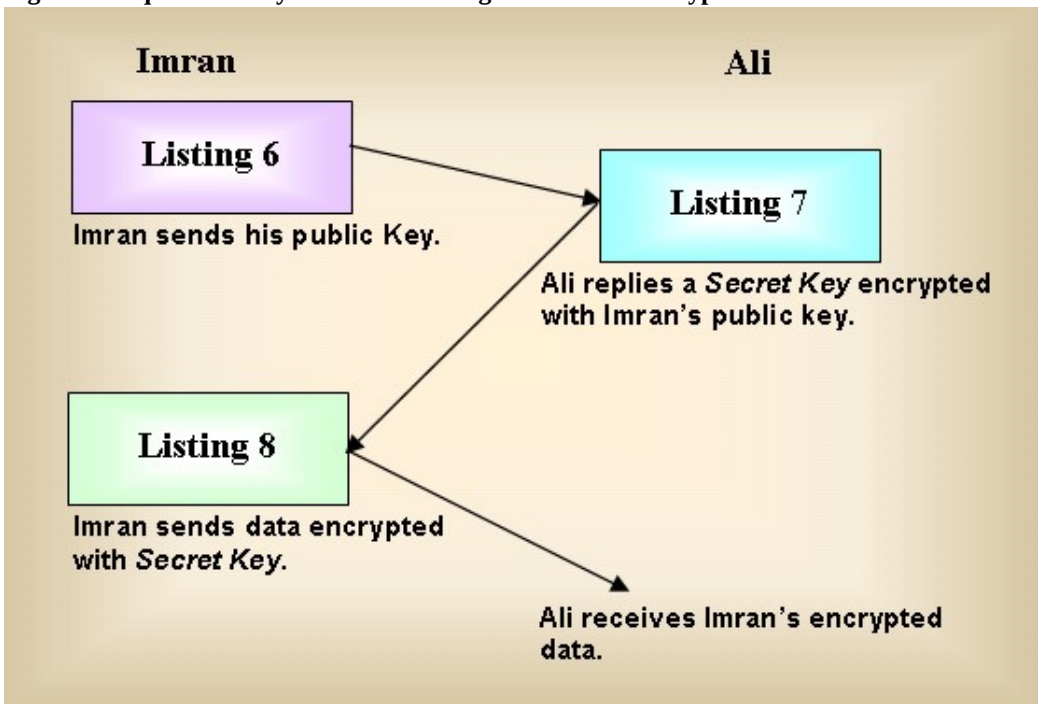
Asymmetric keys for exchange of secret keys
In this scenario, one party sends its public key to a second party. The second party uses this public key to encrypt its secret key. This exchange of data is shown in Listing 6 (request) and Listing 7 (response). We will imagine Imran and Ali as the first party and second party, respectively, communicating with each other. Imran initializes the public key exchange request and sends his public key in the element named KeyValue . The attribute `CarriedKeyName` represents the name of the key that is being transported. Note that the root element of this structure is `EncryptedKey`, which contains the `ds:KeyInfo` and `ds:KeyValue` elements. The `ds: KeyInfo` and `ds:KeyValue` elements belong to the XML Digital Signature (ds:) namespace. XML Encryption relies entirely on the XML Digital Signature specification for key exchange. Therefore, both `<ds:EncryptedKey>` and `<ds:KeyValue>` belong to the XML Digital Signature specification namespace. Listing 7 is what Ali sends in response. The `CipherValue` element in Listing 7 contains a newly generated secret key, which is encrypted with the public key of the first party. Looking closely at Listing 6 and Listing 7, you'll notice that both request and response contain an `EncryptedKey` element. The `ds:KeyInfo` and `ds:KeyValue` elements within the `EncryptedKey` element carry the public key (Listing 6). On the other hand, the `CipherData` and `CipherValue` elements inside the `EncryptedKey` element (Listing 7) will transport the secret (encrypted) keys. Also notice that the `EncryptedKey` element always contains a `CarriedKeyName` attribute to specify the name of the key it is carrying.

Using keys we have already exchanged in the past
In the previous section, we exchanged a secret key. We'll now use that key to encrypt data. We will assume that Imran sends an XML message (Listing 8) in response to Listing 7 (recall that Listing 7 contains an encrypted secret key whose name is "Imran Ali"). Imran will decrypt this secret key with his (Imran's own) private key (as Ali encrypted this secret key with Imran's public key). Imran can encrypt the data he wants to send to Ali using this secret key and placing it inside the `CipherValue` element in Listing 8.

The `ds:KeyInfo` element in Listing 8 contains a `KeyName` element. This combination refers to the name of the key that Imran uses for data encryption.

Figure 1 is a visual diagram showing this exchange of XML files for secure data exchange.

Figure 1. Sequence of key and data exchange with XML Encryption



Referring external encrypted data from our XML Encryption file

In Listings 5 and 7, the `CipherData` element can appear within an `EncryptedData` element or an `EncryptedKey` element. We use a `CipherData` element to refer to either the encrypted data (when it appears inside an `EncryptedData` element) or the encrypted key (when it appears inside an `EncryptedKey` element). In both Listings 5 and 7, there is a `CipherValue` child element inside the `CipherData` element that contains the actual encrypted data.

We can also refer to external encrypted data or encrypted keys. This means that actual encrypted data or keys will be present somewhere else (perhaps somewhere on the Internet) and not inside our XML Encryption file. In this case we will use `CipherReference` instead of the `CipherValue` child element inside `CipherData`. We'll refer to the actual encrypted data through a URI. This is shown in Listing 9.

Referencing a particular element of an external XML file

Listing 10 illustrates a variation of referring external XML files. Here we have referenced only a portion of the external file that the URI is pointing to. There is a `Transforms` child element inside the `CipherReference` element. This `Transforms` element may contain a number of `Transform` elements, each of which will contain a single XPath element. This XPath element specifies an XPath expression that refers to a particular node of the external XML document.

The DOM structure of our API

We have already demonstrated how to author XML Encryption files and exchange encrypted data. We will now propose a Java API for XML Encryption and provide a sample implementation. We will use DOM for this purpose.

Our DOM implementation consists of a set of classes (Listings 11 to 16). The `XmlEncryption` class (Listing 11) is a wrapper for the rest of the classes, which means users of our API will only need to interact with this class. It uses the functionality of other classes internally.

Listing 11 is a wrapper class that can generate a complete XML encrypted file.

Listing 12 authors the `EncryptedData` element.

Listing 13 authors the `EncryptionMethod` element.

Listing 14 authors the `KeyInfo` element.

Listing 15 authors the `CipherData` element.

Listing 16 contains names of Algorithms as static integers and their corresponding namespaces as strings.

The `XmlEncryption` class (Listing 11) contains various public Get/Set methods. The user will call Set methods to specify encryption parameters, which include the following:

1. Name of the file to be encrypted
2. Name of the resulting XML Encryption file
3. Name of the algorithm for encryption
4. Name of the key that we will use for encryption
5. An ID for identification of `<EncryptedData>` structure

We have demonstrated the use of the `XmlEncryption` class (Listing 11) through a `main ()` method. In the `main ()` method, we have created an instance of this class. The constructor instantiates DOM so that all underlying classes will use the same object.

This implementation only supports encryption of complete files, as illustrated in Listing 2. The `EncryptCompleteXmlFile ()` method will do this job by calling the following methods in a sequence:

1. `GetEncryptedDataDoc()` returns the object of the `EncryptedData` class (Listing 12). It contains the structure of the `EncryptedData` element.
2. `GetEncryptionMethodDoc()` returns the Document object, which contains the XML structure corresponding to the `EncryptionMethod` element. `GetEncryptionMethodDoc()` uses `EncryptionMethod` class (Listing 13) to author XML.
3. `GetKeyInfoDoc()` returns the `Document` object, which contains the XML structure corresponding to KeyInfo element.

`GetKeyInfoDoc()` uses the object of `GenericKeyInfo` class (Listing 14) to author the XML. This class only provides the minimum necessary functionality (support for `KeyName` and `KeyValue` elements) you will inherit from `GenericKeyInfo` class to provide the complete functionality, which includes support for X509 Certificates, PGP Data, etc.

4. `ReadFile()` fetches the data (complete XML file) that we want to encrypt.
5. `GetEncryptedData()` for the time being is not doing anything. We'll implement this method in the next part of this article. It is supposed to create the encrypted form of XML data that we fetched in step 4. We have briefly discussed our encryption strategy in the last section (Java Cryptographic Architecture).
6. `GetCipherDataDoc()` takes the encrypted data as an argument and returns the Document Object containing the `CipherData` element. `GetCipherDataDoc()` uses the Object of `CipherData` class (Listing 12) to author XML.
7. At the end, `addChild()` method of Object of `EncryptedData` (Listing 15) is called thrice, which will take the Document Objects of steps 2, 3, and 6 and adds them to the `<EncryptedData>` structure, which is the parent of all of them.
8. `SaveEncryptedFile()` saves the completed XML Encryption file.

`AlgoNames` (Listing 16) is a helper class that only specifies namespace declarations required by XML Encryption.

The `XmlEncryption` class (Listing 11) can also be used as a server-side component. In the next part of this series, we'll demonstrate its use inside independent as well as server-side applications.

The set of classes that we have developed only performs DOM-based XML authoring. We need to implement cryptographic functionality as well. We will now try to form a strategy for cryptographic support. For this purpose, we need to study the Java Cryptographic Architecture (JCA).

The Java Cryptographic Architecture (JCA)
Java offers complete support for cryptography. For this purpose, there are several packages inside J2SE, covering all the main features of security architecture such as access controls, signatures, certificates, key pairs, key stores, and message digests.

Public, private, and secret keys

We have used three technical terms related to keys (public, private, and secret keys). Although these terms are well known to developers working with end-to-end security, XML developers might not be familiar with them. Let's clarify these terms:

Public and private keys: We use them as a pair. Some algorithms generate a pair of public and private keys. We send the public key to anyone who wants to exchange encrypted data with us. With a public key, we can only encrypt data of limited size. Our communicating partner encrypts the data with our public key and sends the encrypted data to us. We then decrypt the data with our private key. This is asymmetric encryption.

Secret key: We use public and private keys to exchange a secret key. We normally generate secret keys randomly. Once we have exchanged a secret key with our communicating partner through asymmetric encryption, we then use this key for encrypting data at both ends. This is symmetric encryption.

The primary principle of JCA design is to separate cryptographic concepts from algorithmic implementations, so that different vendors can offer their tools within the JCA framework.

JCA Engine classes
JCA defines a series of Engine classes, where each Engine provides a cryptographic function. For example, there are several different standards of MD (Message Digest) algorithm. All these standards differ in the implementation, but at the Engine API level they are all the same. Different vendors are free to provide implementations of specific algorithms.

Java Cryptographic Extension (JCE)
All independent (third party) vendor implementations of cryptographic algorithms are called Java Cryptographic Extensions (JCEs). Sun Microsystems has also provided an implementation of JCE. Whenever we use JCE, we need to configure it with JCA. For this, we need to do the following:

1. Add the address of the jar file to configure the provider (all JCE implementations are called providers) in the `CLASSPATH` environment variables.

2. Configure the provider in the list of your approved providers by editing the java.security file. This file is located in JavaHome/jre/lib/security folder. The following is the syntax to specify the priority: `security.provider.<n>=<masterClassName>`. Here, n is the priority number (1, 2, 3, etc.). `MasterClassName` is the name of master class to which the engine classes will call for a specific algorithm implementation. The provider's documentation will specify its master class name. For example, consider the following entries in a java.security file:

- `security.provider.1=sun.security.provider.Sun`
- `security.provider.2=com.sun.rsajca.Provider`
- `security.provider.3=com.sun.net.ssl.internal.ssl.Provider`

These entries mean that the engine class will search for any algorithm implementation in the above mentioned order. It will execute the implementation found first. After these simple steps, we are all set to use JCA/JCE in our XML Encryption application.

Using JCA and JCE in our implementation of XML Encryption

The `GetEncryptedData()` function in our wrapper class, `XmlEncryption` (Listing 11), is the place to handle all JCA/JCE-related issues. Currently this method only returns the string "This is Cipher Data". We have not yet written JCA/JCE-related classes. This method takes the unencrypted data and returns it as an encrypted string. We will handle all the algorithm- and key-related issues in this method after writing the wrapper classes for JCA/JCE.

Next time: In our next installment of this series of articles, we will discuss and implement the details of cryptography. We'll demonstrate the working of encryption and decryption classes and their interaction with parsing logic, and present applications of XML Encryption in Web services.

Resources

- Visit W3C to see official details of XML Encryption Requirements and XML Encryption Processing and Syntax.
- Check out news about XML Encryption at OASIS/Robin Cover's XML Cover Pages.
- IBM's toolkits for XML Encryption is available for download at alphaWorks.
- KeyTools XML is another Security tool from Baltimore.
- Check what's happening at SUN's community process for XML Encryption.
- We mentioned XPath in this article. Here is a chapter on XPath from an XML book *XML in a Nutshell* by O'Reilly.
- We mentioned IANA in this article. Visit IANA's Web site to check type definitions for popular data formats.
- Of course, you'll find plenty more security-related resources on the *developerWorks* Security special topic area.
- IBM Security Services can help you determine what your risks are, and then design a security program to address them.
- IBM WebSphere Studio Application Developer is an easy-to-use, integrated development environment for building, testing, and deploying J2EE (TM) applications, including generating XML documents from DTDs and schemas.

About the author

XML consultant Bilal Siddiqui received a degree in Electronics Engineering from the University of Engineering and Technology, Lahore, in 1995. He then began designing software solutions for industrial control systems. Later he turned to XML and used his experience programming in C++ to build Web- and WAP-based XML processing tools, server-side parsing solutions, and service applications. You can e-mail Bilal for working copies of the code files contained in this article at wap_monster@yahoo.com.

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 2. Encrypting the entire file

```
<?xml version='1.0' ?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
               Type='http://www.isi.edu/in-notes/iana/assignments/media-
types/text/xml'>
        <CipherData>
                <CipherValue>A23B45C56</CipherValue>
        </CipherData>
</EncryptedData>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 3. Encrypting only the <Payment> element

```
<?xml version='1.0' ?>
<PurchaseOrder>
        <Order>
                <Item>book</Item>
                <Id>123-958-74598</Id>
                <Quantity>12</Quantity>
        </Order>
        <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'

xmlns='http://www.w3.org/2001/04/xmlenc#'>
                <CipherData>
                        <CipherValue>A23B45C564587</CipherValue>
                </CipherData>
        </EncryptedData>
</PurchaseOrder>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 4. Encrypting only the content in the CardId element

```
<?xml version='1.0' ?>
<PurchaseOrder>
        <Order>
                <Item>book</Item>
                <Id>123-958-74598</Id>
                <Quantity>12</Quantity>
        </Order>
        <Payment>
                <CardId>
                        <EncryptedData
Type='http://www.w3.org/2001/04/xmlenc#Content'

xmlns='http://www.w3.org/2001/04/xmlenc#'>
                                <CipherData>
                                        <CipherValue>A23B45C564587</CipherValue>
                                </CipherData>
                        </EncryptedData></CardId>
                <CardName>visa</CardName>
                <ValidDate>12-10-2004</CardName>
        </Payment>
</PurchaseOrder>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 5. Encrypting arbitrary non-XML data (JPEG)

```
<?xml version='1.0' ?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
                        Type='http://www.isi.edu/in-notes/iana/assignments/media-
types/jpeg' >
        <CipherData>
                <CipherValue>A23B45C56</CipherValue>
        </CipherData>
</EncryptedData>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 6. One party sends its public key to a second party for key exchange

```
<?xml version='1.0' ?>
<SecureCommunicationDemonstration>
        <EncryptedKey  CarriedKeyName="Muhammad Imran"
xmlns='http://www.w3.org/2001/04/xmlenc#'>
                <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
                        <ds:KeyValue>1asd25fsdf2dfdsfsdfds2f1sd23</ds:KeyValue>
                </ds:KeyInfo>
        </EncryptedKey>
</SecureCommunicationDemonstration>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 7. Second party sends back the randomly generated secret key encrypted with public key of first party

```
<?xml version='1.0' ?>
<SecureCommunicationDemonstration>
        <EncryptedKey  CarriedKeyName="Imran Ali"
xmlns='http://www.w3.org/2001/04/xmlenc#'>
                  <EncryptionMethod   Algorithm= "http://www.w3.org/2001/04/xmlenc#rsa-
1_5"/>
                        <CipherData>
                                <CipherValue>xyza21212sdfdsfs7989fsdbc</CipherValue>
                        </CipherData>
        </EncryptedKey>
</SecureCommunicationDemonstration>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 8. Data is encrypted with a secret key and placed in the <CipherValue> element

```
<?xml version='1.0' ?>
<<SecureCommunicationDemonstration>
        <Order>
        <Item>book</Item>
              <Id>123-958-74598</Id>
        <Quantity>12</Quantity>
              <CardName>Visa</CardName>
              <ExpDate>10-10-2005</ExpDate>

                <EncryptedData  Type='http://www.w3.org/2001/04/xmlenc#Element'

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>Imran ali</ds:KeyName>
                        </ds:KeyInfo>
                        <CipherData>
                                <CipherValue>A23B45C564587</CipherValue>
                        </CipherData>
                </EncryptedData>
        </Order>
</SecureCommunicationDemonstration>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 9. Using the <CipherReference> element to refer to external encrypted data

```
<?xml version='1.0' ?>
<EncryptedData   xmlns='http://www.w3.org/2001/04/xmlenc#'
                                Type='http://www.w3.org/2001/04/xmlenc#Element'>
        <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
                <ds:KeyName>Imran ali</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
                <CipherReference URI="www.waxsys.com/secureData/waxFile.txt"/>
        </CipherData>
</EncryptedData>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 10. Using the <Transforms> element and XPath to refer to a particular node of the external XML File

```
<?xml version='1.0' ?>
<EncryptedData  ID="Enc-Data" xmlns='http://www.w3.org/2001/04/xmlenc#'

Type='http://www.w3.org/2001/04/xmlenc#Element' >
        <CipherReference URI="http://www.waxsys.com/EncFile.xml" >
                <Transforms  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" >
                <ds:Transform   Algorithm="http://www.w3.org/TR/1999/REC-xpath-
19991116">
                        <wax:XPath xmlns:wax="http://www.waxsys.com/xpathNS">
                                PruchaseOrder/EncryptedData [@Id="Imran-Enc-Data"]
                        </wax:XPath>
                        </ds:Transform>
          </Transforms>
        </CipherReference>
</EncryptedData>
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 11. XmlEncryption.java

```
 /*
        DW/BS
        20020204
        XmlEncryption.java
        Listing 11
        A wrapper class that can generate complete XML encrypted file.
        It uses all the other classes.
        Users of our XML Encryption Engine will only need to interact with
        this class.
 */

import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.crimson.tree.XmlDocument;

public class XmlEncryption {

        // Source and Result file names.
        private String fileSource = null;
        private String fileResult = null;

        // Name of Algorithm which will be used to encrypt data.
        private String algoName = null;

        // Name of Secret key which was previously agreed upon
        // and saved with the given name.
        private String keyName = null;

        // Id attribute of Main structure
        private String encId = null;

        // It will be used to get New Document Objects.
        private DocumentBuilder docBuilder = null;

        // Default Constructor
        public XmlEncryption() {
                // Create DocumentBuilder object from DocumentBuilderFactory.
                try {
                        docBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
```

```
                } catch (ParserConfigurationException e){ docBuilder = null; }
        }

        // Get the new Document object for DocumentBuilder
        private Document getNewDocument() {
                if (docBuilder != null)
                        return docBuilder.newDocument();
                else
                        return null;
        }

        // Generate Complete XML Encrypted File.
        public void encryptCompleteXmlFile(){
                // Take an Object of EncryptedData Class.
                // It represents EncryptedData Element.
                EncryptedData encDataObj = this.getEncryptedDataDoc(this.encId,
"DOCUMENT");

                // Get XML Structure for EncryptionMehtod element.
                Document encMethodDoc = this.getEncryptionMethodDoc(this.algoName);

                // Get XML Structure for KeyInfo element.
                Document encKeyInfoDoc = this.getKeyInfoDoc(this.keyName);

                // Read the given file data which will be encrypted.
                String plainData = this.readFile(fileSource);

                // Use of JCA/JCE to get encrypted data.
                String cipherData = this.getEncryptedData(plainData);

                // Get XML Structure for CipherData element.
                Document cipherDataDoc = this.getCipherDataDoc(cipherData);


                // Join these XML Structures.
                encDataObj.addChild(encMethodDoc);
                encDataObj.addChild(encKeyInfoDoc);
                encDataObj.addChild(cipherDataDoc);

                // Now Save this Document as an XML File
                this.saveEncryptedFile(this.fileResult, encDataObj.getEncData());
        }// End encryptCompleteXmlFile()


        //********** All Set/Get methods to related fields.

        public void setFileSource(String file) {
                this.fileSource = file;
        }// End setFileSource()

        public String getFileSource() {
                return this.fileSource;
        }// End getFileSource()

        public void setFileResult(String file) {
                this.fileResult = file;
        }// End setFileResult()

        public String getFileResult() {
                return this.fileResult;
        }// End getFileResult()

        public void setAlgoName(String name) {
                this.algoName = name;
        }// End setAlgoName()
```

```
	public String getAlgoName() {
		return this.algoName;
	}// End getAlgoName()

	public void setKeyName (String key) {
		this.keyName = key;
	}// End setKeyName()

	public String getKeyName() {
		return this.keyName;
	}// End getKeyName()

	public void setEncId (String id) {
		this.encId = id;
	}// End setEncId()

	public String getEncId() {
		return this.encId;
	}// End setEncId()

	//**************

	// Reads the given file and returns it as string.
	public String readFile(String fileName){
		String xml = "";
		try {
			FileInputStream in = new FileInputStream(fileName);
			byte [] data = new byte[in.available()];
			in.read(data);
			xml = new String(data);
		} catch (IOException e) { }
		return xml;
	}// End readFile()

	// Saves the given document as an XML (Text) file with given name.
	public void saveEncryptedFile (String fileName, Document doc) {
		XmlDocument xmlDoc = (XmlDocument)doc;
		try {
			OutputStream out = new FileOutputStream(fileName);
			xmlDoc.write(out);
			out.close();
		} catch (IOException e) { }

	}// End saveEncryptedFile()

	// Returns the EncryptedData Object.
	public EncryptedData getEncryptedDataDoc(String Id, String encType) {
		EncryptedData ed = new EncryptedData(this.getNewDocument());
		ed.setId(Id);
		if (encType.equals("DOCUMENT"))
			ed.setType(AlgoNames.DOCUMENT);

		return ed;
	}// End getEncryptedDataDoc()

	// Returns the EncryptionMehtod Structure.
	public Document getEncryptionMethodDoc (String algoName) {
		EncryptionMethod em = new EncryptionMethod(this.getNewDocument());
		if (algoName.equals("TripleDes-cbc"))
			em.setAlgorithm(AlgoNames.TRIPLE_DES);
		return em.getEncMethod();
	}// End getEncryptionMethodDoc()

	// Returns the KeyInfo Structure.
```

```
        public Document getKeyInfoDoc (String keyName) {
                GenericKeyInfo ki = new GenericKeyInfo(this.getNewDocument(),"ds",
AlgoNames.XML_DSIG);
                ki.setKeyName(keyName);
                return ki.getKeyInfo();
        }// End getKeyInfoDoc()

        // Returns the CipherData Structure.
        public Document getCipherDataDoc (String data) {
                CipherData cd = new CipherData(this.getNewDocument());
                cd.setValue(data);
                return cd.getCipherData();
        }// getCipherDataDoc()

        // In the future, all JCA/JCE related classes will be used here.
        // It will take plain text and return its encrypted form.
        // All necessary Infromation about keys and algos will be
        // taken from the fields representing them.
        // For the time being it is not doing any thing.
        public String getEncryptedData(String data) {
                return "This is Cipher Data";
        }// End getEncryptedData()

        // This main method is only included to demonstrate functionality.
        public static void main (String args[]) {
                XmlEncryption xmlEnc = new XmlEncryption();
                xmlEnc.setFileSource("Order.xml");
                xmlEnc.setFileResult("EncryptedOrder.xml");
                xmlEnc.setAlgoName("TripleDes-cbc");
                xmlEnc.setKeyName("ImranAli");
                xmlEnc.setEncId("Test");
                xmlEnc.encryptCompleteXmlFile();
        }// End main()
}// End Class DemoApplication
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 12. Authors the EncryptedData element

```
 /*
        DW/BS
        20020204
        EncryptedData.java
        Listing 12
        Authors the EncryptedData element which is the parent of
        all XML Encryption structures except EncryptedData element.
 */

import org.w3c.dom.*;

public class EncryptedData{

        // Element to hold complete XML Structure.
        Element encData = null;
        Document doc = null;

        // Element will be appended in Document only once.
        private boolean elementAppendedToDoc = false;

        // Default Constructor
        public EncryptedData(Document document) {
                doc = document;
                encData = doc.createElement("EncryptedData");

                // Add the namespace attribute
                encData.setAttribute("xmlns","http://www.w3.org/2001/04/xmlenc#");
        }// End EncryptedData()

        // Specify the ID of this EncryptedData as there
        // can be many EncryptedData elements in the same XML file.
        public void setId(String id){
                encData.setAttribute("Id", id);
        }// End setId()

        // There can be three types:
        // 201 for ELEMENT
        // 202 for CONTENT
        // 303 for DOCUMENT
        public void setType(int type){
```

```
                encData.setAttribute("Type", AlgoNames.getAlgoNSValue(type));
        }// end setType()


        // If an Arbitrary type is to be Encrypted, specify its mimeType.
        public void setArbitraryType(String mimeType){
                String typeValue = "http://www.isi.edu/in-
notes/iana/assignments/media-types/" + mimeType;
                encData.setAttribute("Type", typeValue);
        }// End setArbitraryType()

        // Add any of these XML Documents KeyInfo | CipherData | Transforms |
EncryptionMethod.
        public void addChild(Document document){
                NodeList nList = document.getChildNodes();

                for (int i=0 ; i<nList.getLength(); i++){
                        Node tempNode = nList.item(i);
                        // If it is a Root Element
                        if (tempNode.getNodeType() == Node.ELEMENT_NODE) {
                                Node importedNode = (Node)doc.importNode(tempNode,

true /* import all childern*/);
                                encData.appendChild((Element)importedNode);
                                return;
                        }// end if (tempNode.getNodeType() == Node.ELEMENT_NODE)
                }// end for (int i=0 ; i<nList.getLength(); i++)
        }// End addChild()

        // Return the complete XML structure as a Document Object.
        public Document getEncData(){
                if (elementAppendedToDoc == false) {
                        doc.appendChild(encData);
                        elementAppendedToDoc = true;
                }
                return doc;
        }// end getEncData()
}// End class EncryptedData
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 13. Authors the EncryptionMethod element

```
 /*
        DW/BS
        20020204
        EncryptionMethod.java
        Listing 13
        Authors the EncryptionMethod element.
 */

import org.w3c.dom.*;
import javax.xml.parsers.*;

public class EncryptionMethod{

        // Element to hold complete XML Structure.
        private Element encMethod = null;
        private Document doc = null;

        // Element will be appended in Document only once.
        private boolean elementAppendedToDoc = false;

        // Constructor
        public EncryptionMethod (Document document) {
                doc = document;
                encMethod = doc.createElement("EncryptionMethod");
        }// End EncryptionMethod()

        // Set the name of Algorithm.
        public void setAlgorithm(int algoNo){
                encMethod.setAttribute("Algorithm",
AlgoNames.getAlgoNSValue(algoNo));
        }// End setAlgorithm()

        // Get the complete Document Sturcture.
        public Document getEncMethod() {
                if (elementAppendedToDoc == false) {
                        doc.appendChild(encMethod);
                        elementAppendedToDoc = true;
                }
                return doc;
        }// End getEncMethod()
```

```
}// End Class EncryptionMethod
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 14. Authors the KeyInfo element

```
 /*
        DW/BS
        20020204
        GenericKeyInfo.java
        Listing 14
        Authors the KeyInfo element.
 */

import org.w3c.dom.*;

public class GenericKeyInfo {

        // Element to hold complete XML Structure.
        protected Element keyInfo = null;
        protected Document doc = null;

        // Element will be appended in Document only once.
        protected boolean elementAppendedToDoc = false;


        // Default constructor.
        public GenericKeyInfo (Document document, String nsQlif, int typeNo) {
                doc = document;

                keyInfo = doc.createElement(nsQlif + ":KeyInfo");

                // Add the namespace attribute.
                keyInfo.setAttribute("xmlns:" + nsQlif,
AlgoNames.getAlgoNSValue(typeNo));
        }// End GenericKeyInfo

        // Add KeyName child in KeyInfo
        public void setKeyName(String keyName) {
                Element tempElem = doc.createElement("KeyName");
                tempElem.appendChild(doc.createTextNode(keyName));
                keyInfo.appendChild(tempElem);
        }// End setKeyName()

        // Add KeyValue child in KeyInfo
        public void setKeyValue (String keyValue){
```

```
                Element tempElem = doc.createElement("KeyValue");
                tempElem.appendChild(doc.createTextNode(keyValue));
                keyInfo.appendChild(tempElem);
        }// End setKeyValue()

        // Set retrieval Method URI and and Its type.
        public void setRetrievalMethod (String uriValue, int typeNo) {
                Element tempElem = doc.createElement("RetrievalMethod");
                tempElem.setAttribute("URI", uriValue);
                tempElem.setAttribute("Type", AlgoNames.getAlgoNSValue(typeNo));
                keyInfo.appendChild(tempElem);
        }// End setRetrievalMethod()

        // Return the complete XML structure as a Document Object.
        public Document getKeyInfo() {
                if (elementAppendedToDoc == false ) {
                        doc.appendChild(keyInfo);
                        elementAppendedToDoc = true;
                }
                return doc;
        }// End getKeyInfo()
} // end class KeyInfo
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 15. Authors the CipherData element

```
/*
        DW/BS
        20020204
        CipherData.java
        Listing 15
        Authors the CipherData element.
 */

import org.w3c.dom.*;

public class CipherData {

        // We will create child elements within this document.
        private Document doc = null;

        // The main structure.
        private Element cipherData = null;

        // Element will be appended in Document only once.
        private boolean elementAppendedToDoc = false;

        // Constructor
        public CipherData(Document document){
                doc = document;
                cipherData = doc.createElement("CipherData");

                elementAppendedToDoc = false;
        }// End CipherData()

        // Sets the encrypted Data inside CipherValue Child tag.
        public void setValue(String value){
                Element tempElem = doc.createElement("CipherValue");
                tempElem.appendChild(doc.createTextNode(value));
                cipherData.appendChild(tempElem);
        }// End setValue()

        // Adds CipherReference element inside CipherData element.
        // Its Attribute URI is passed as parameter.
        // Transform element is optional.
        // If want to set only URI then pass NULL in place of transforms Element
```

```
        public void setCipherReference (String uriValue, Element transforms) {
                Element tempElem = doc.createElement("CipherReference");
                tempElem.setAttribute("URI", uriValue);

                if ( transforms != null )
                        tempElem.appendChild(transforms);

                cipherData.appendChild(tempElem);
        }// End setCipherReference()

        // Retuns the completed CipherData structure.
        public Document getCipherData() {
                if (elementAppendedToDoc == false ) {
                        doc.appendChild(cipherData);
                        elementAppendedToDoc = true;
                }
                return doc;
        }// End getCipherData()
}// end Class CipherData
```

Demonstrating the secure exchange of structured data

Bilal Siddiqui (wap_monster@yahoo.com)
CEO, WAP Monster
March 2002

Related content:
Subscribe to the developerWorks newsletter

Also in the XML zone:
Tutorials
Tools and products
Code and components
Articles

Listing 16. AlgoNames.java

```
 /*
        DW/BS
        20020204
        AlgoNames.java
        Listing 16
        This class is only for convenience of use.
        It contains names of Algorithms as static integers
        and their corresponding namespaces as strings.
 */

public class AlgoNames {

        public static final int TRIPLE_DES    =  1;
        public static final int AES_128       =  2;
        public static final int AES_256       =  3;
        public static final int RSA_V1_5      =  4;
        public static final int RSA_OAEP      =  5;
        public static final int DH                    =  6;
        public static final int KW_TRIPLE_DES =  7;
        public static final int KW_AES_128    =  8;
        public static final int KW_AES_256    =  9;
        public static final int SHA1          = 10;
        public static final int SHA256        = 11;

        public static final int XPATH         = 100;
        public static final int BASE_64           = 101;
        public static final int ENC_KEY           = 102;
        public static final int XML_DSIG          = 103;

        // Fields to represent what type of Data to be encrypted.
        public static final int ELEMENT           = 201;
        public static final int CONTENT           = 202;
        public static final int DOCUMENT          = 203;



        public static String getAlgoNSValue(int algoNo){
                String algoNS = "";
                switch (algoNo){
                        case TRIPLE_DES :  algoNS =
"http://www.w3.org/2001/04/xmlenc#tripledes-cbc"; break;
```

```
                        case AES_128    :  algoNS =
"http://www.w3.org/2001/04/xmlenc#aes128-cbc";    break;
                        case AES_256    :  algoNS =
"http://www.w3.org/2001/04/xmlenc#aes256-cbc";    break;
                        case RSA_V1_5   :  algoNS =
"http://www.w3.org/2001/04/xmlenc#rsa-1_5";       break;
                        case RSA_OAEP   :  algoNS =
"http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p";break;
                        case DH                 :  algoNS =
"http://www.w3.org/2001/04/xmlenc#dh";                    break;
                        case KW_TRIPLE_DES:algoNS =
"http://www.w3.org/2001/04/xmlenc#kw-tripledes";  break;
                        case KW_AES_128 :  algoNS =
"http://www.w3.org/2001/04/xmlenc#kw-aes128";     break;
                        case KW_AES_256 :  algoNS =
"http://www.w3.org/2001/04/xmlenc#kw-aes256";     break;
                        case SHA1               :  algoNS =
"http://www.w3.org/2000/09/xmldsig#sha1";                 break;
                        case SHA256             :  algoNS =
"http://www.w3.org/2000/09/xmldsig#sha256";               break;
                        case XPATH              :  algoNS =
"http://www.w3.org/TR/1999/REC-xpath-19991116";   break;
                        case BASE_64    :  algoNS =
"http://www.w3.org/2000/09/xmldsig#base64";               break;
                        case ENC_KEY    :  algoNS =
"http://www.w3.org/2001/04/xmlenc#EncryptedKey";  break;
                        case XML_DSIG   :  algoNS =
"http://www.w3.org/2000/09/xmldsig#";                     break;
                        case ELEMENT    :  algoNS =
"http://www.w3.org/2001/04/xmlenc#Element";               break;
                        case CONTENT    :  algoNS =
"http://www.w3.org/2001/04/xmlenc#Content";               break;
                        case DOCUMENT   :  algoNS = "http://www.isi.edu/in-
notes/iana/assignments/media-types/text/xml";  break;

                } // end switch(algoNo)
                return algoNS;
        }// End getAlgoNsValue()
}// End Class
```