Exploring XML Encryption, Part 2

Implement an XML Encryption engine

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

In this second installment, Bilal Siddiqui examines the usage model of XML Encryption with the help of a use case scenario. He presents a simple demo application, explaining how it uses the XML Encryption implementation. He then continues with his last implementation of XML Encryption and makes use of JCA/JCE classes to support cryptography. Finally, he briefly discusses the applications of XML Encryption in SOAP-based Web services.

In Part 1 of this series, I gave an introduction to XML Encryption and its underlying syntax and processing. I examined the different tags and their respective use in XML encryption with a simple example of secure exchange of structured data, proposed a Java API for XML Encryption based on DOM, and gave a brief overview of cryptography in Java (JCA/JCE).

I start my discussion in this part with an information exchange scenario, which demonstrates the use of XML encryption.

Information exchange scenario

Consider the process of information exchange between two enterprises. One is an online books-seller and the other is a publisher. When the books-seller wants to purchase books, it submits a purchase order to the publisher. At the publisher's end, the sales department receives this order, processes it, and forwards it to the accounts department. The two enterprises exchange information in the form of XML documents. Since some portion of the document needs to be secure and the rest can be sent insecurely, XML encryption is the natural approach for applying security to distinct portions of the document.

According to the books-seller's security policy, the payment information will only be revealed to the accounts department. The sales department will need to extract only the name of the book, its item ID and the quantity ordered; because this is insensitive information it can remain insecure. The accounts department will need to decrypt the payment information in the purchase order using a pre-exchanged secret key. (Note that XML Encryption is only about encryption and decryption of structured information and does not dictate any particular method of key exchange.) Mapping this policy, XML Encryption facilitates the concealment of payment information in the sales department and its disclosure in the accounts department.

Document-based security

At this point, it may be useful to ponder a bit on the concept of document-based security. With this security architecture, you can impose security at the document level. The context of a secure session is effectively preserved within the secure document. All the information that an authorized party may need to decrypt the document is available inside the document. A logical secure session is created that is flexible, has a long life, and allows numerous parties to be part of the same secure session. An upcoming protocol for Web services, the Business Transaction Protocol (BTP -- see Resources), relies on the same concept of preserving the context of a session within a transaction document; this prolongs the life of a transaction and enhances its flexibility.

The demo application

To fulfill the requirements of the secure data exchange scenario between the publisher and the books-seller described above, I have created a demo application class called `demoXmlEncApp` (see Listing 1). This class uses the XML Encryption sample implementation class `XmlEncryption` (see Listing 2).

Related content:

Exploring XML Encryption, Part 1
Subscribe to the developerWorks newsletter
More dW Web architecture resources

Also in the XML zone:

Tutorials
Tools and products
Code and components
Articles

What the demo application does

The `main` method of the `demoXmlEncApp` class first executes the processing at the books-seller's end by calling the `simulateBookSellersEnd` method, which reads the purchase order XML file, Order.xml (see Listing 3), and encrypts the sensitive information therein through one of the three encryption methods (i.e. complete XML file encryption, element encryption, or element content encryption) specified in XML Encryption. It then saves the XML-encrypted file to disk, as well as the key used for encryption.

Listing 3. Order.xml

```
<?xml version="1.0"?>
<!-- This Listing provides the sample XML File that will be encrypted. -->
<purchaseOrder>
        <Order>
                <BookName>Soccer For Dummies</BookName>
                <Id>123-958-74598</Id>
                <Quantity>500</Quantity>
        </Order>
        <Payment>
                <CardNo>4502-3456-3278-2011</CardNo>
                <CardType>VISA</CardType>
                <ValidDate>12-10-2004</ValidDate>
        </Payment>
</purchaseOrder>
```

Since file exchange and key exchange protocols are not a concern here, I therefore assume that they are made available to the publisher through any suitable method (for example, the XML-encrypted file and secret key are exchanged through HTTP and some public key encryption algorithm, respectively). The `main` method then assumes the role of the publisher by calling the `simulatePublishersEnd` method, which extracts the information needed by the sales department, and displays the received XML file on the console. Further, it decrypts the payment information for the accounts department and displays it on the console.

The `simulateBookSellersEnd` function in the `demoXmlEncApp` class first instantiates an `XmlEncryption` object and calls various setter methods of `XmlEncryption` to set the following attributes:

- `clearDoc`: The DOM object form of the XML file (see Listing 3) to be encrypted
- `encKey`: A key used for encryption
- `algoName`: Name of the encryption algorithm
- `keyName`: Name of the encryption key; this becomes the value of the child text node of the `KeyName` element during XML encryption as explained in part 1
- `encId`: The name given to the `EncryptedData` tag which is unique in the document

The `simulateBookSellersEnd` method of `demoXmlEncApp` class calls the different methods of the XML Encryption engine (`XmlEncryption` class) depending upon the command line arguments passed while invoking the application. Let's see what happens inside the `XmlEncryption` class, which is the main XML Encryption processing engine.

XML Encryption implementation details

I already introduced a portion of the proposed XML encryption API in Part 1. Recall that I had a method named `encryptCompleteXmlFile` in the class `XmlEncryption` (see Listing 11 of part 1) for encrypting a complete XML file. In addition to the `encryptCompleteXmlFile` method, I have now added the following two methods to service the encryption requirements of different types of data (encryption granularity):

- `encryptElementOfXmlFile` for encrypting a particular element in the XML file
- `encryptElementContentOfXmlFile` for encrypting the content of a particular element in the XML file

The books-seller can secure the sensitive information in the purchase order by employing any of the following three XML encryption methods of the `XmlEncryption` class:

1. Encrypt a complete XML file
2. Encrypt an element in an XML file
3. Encrypt an element's content in an XML file

1. Encrypt a complete XML file with XML Encryption
The books-seller can encrypt the entire Order.xml file to produce an XML-encrypted file, which can then be sent to the publisher's sales department. Although this provides relevant security through the end-to-end communication link, the books-seller's security policy is violated. This policy requires concealing the payment information in the sales department and revealing it in the accounts department. In this case, the whole XML document is decrypted by the sales department and the payment information is disclosed. Therefore this approach does not seem suitable, although it can be practical if you use super encryption (discussed later in this article).

If the books-seller decides to encrypt the entire Order.xml file, the `simulateBookSellersEnd` function in the class `demoXmlEncApp` generates a call to the public method `encryptCompleteXmlFile` of the class `XmlEncryption`.

This method first calls the `getString` private method to serialize the XML file to be encrypted into string form. It then calls the encryption method `getEncryptedData`, which returns a Base 64 encoded cipher text string. The cipher text string is then passed to the `getCipherDataDoc` private method. This method creates and returns the `CipherData` tag with the child `CipherValue` tag, which holds the Base 64 encoded cipher text string. Similarly the `EncryptionMethod` and `ds:KeyInfo` tags are created. These three tags -- namely `CipherData` (which carries its child `CipherValue` tag), `EncryptionMethod`, and `ds:KeyInfo` -- are subsequently added as child tags of the `EncryptedData` tag. The `EncryptedData` tag is actually loaded into a DOM document object, `encDataObj`, which is serialized and returned to the `simulateBookSellersEnd` method.

2. Encrypt an element in an XML file with XML Encryption
The books-seller can encrypt the payment information portion of the XML file with the accounts department's secret key, and keep the rest of the file content unencrypted for the sales department to view. This processing can be performed by encrypting the `Payment` element in the Order.xml file. The credit card information becomes secure as it resides in the child nodes of the encrypted `Payment` element. Since the security requirement dictates that the means of payment (such as credit card or bank check) must be hidden from unauthorized viewers, encrypting the `Payment` element pays off.

In this case, the `simulateBookSellersEnd` function in the class `demoXmlEncApp` generates a call to the public method `encryptElementOfXmlFile` of the class `XmlEncryption`. The private method `getElement` returns the element node which is then serialized and encrypted to produce the `CipherData` tag. The `EncryptedData` tag creation process is the same as before, except that this method replaces the `Payment` element in `clearDoc` with the `EncryptedData` element by making a call to the private method `replaceElement`. After the replacement has taken place, `clearDoc` is serialized and returned.

3. Encrypt an element's content in an XML file with XML Encryption
The third encryption option the books-seller can exercise is to encrypt only the credit card number in Order.xml. The element content encryption method is invoked, which encrypts only the textual content of the `CardNo` element. This raises an important question: Why do you need to come up with content encryption when the same can be accomplished using element encryption? The use of either method depends on the security policy for the document; if there is a specific need to disclose the name of the element or its attributes, while keeping its content secure, content encryption comes in handy.

If the books-seller wants to hide the credit card number only, the textual content of the element `CardNo` in Order.xml is encrypted. The `simulateBookSellersEnd` method calls the public method `encryptElementContentOfXmlFile`, which differs from the encryptElementOfXmlFile method only in the sense that it operates on the element's content and not on the element itself.

Using XML Encryption to decrypt an XML-encrypted file
When the publisher's end receives an XML-encrypted file, it will need decryption. For all three encryption methods described, the method for decryption is `getDecryptedData`. The `simulateBookSellersEnd` method in `demoXmlEncApp` serializes the XML-encrypted file into a text string and passes it to the `getDecryptedData` method for decryption.

The `getDecryptedData` method identifies the `EncryptedData` tags and extracts the Base 64 encoded cipher value. All the information necessary for decryption is present within the XML Encryption tags: the name of the encryption algorithm, the type of data that was encrypted, and the name of the encryption key.

Note: The XML Encryption specification does not dictate that all this information *should* be present within the XML Encryption tags. The presence of these attributes is optional. The application may supply this information through some other means, but the demo application and sample XML Encryption implementation assume the presence of all this information within the XML Encryption tags.

The `getDecryptedData` method now generates a decryption key from the key and the algorithm names. It then passes the extracted Base 64 encoded cipher value, the decryption key, and the algorithm name to a method named `Decrypt` (examined in

Cryptography).

The `Decrypt` method returns the decoded and decrypted data as a text string. This text string is manipulated in accordance with the type attribute of the `EncryptionData` element which may be any one of the following:

- A complete XML file that is saved to disk as a new XML file
- An element that replaces the `EncryptedData` tag to result in the original XML file
- The content of an element, which replaces the `EncryptedData` tag to result in the original XML file

Super encryption
When you use super encryption, you can encrypt just the payment information with the accounts department's secret key to produce an element-encrypted XML file. This resultant file is then completely encrypted using the sales department's secret key, thus resulting in a super-encrypted XML file.

Here I must point out that, with the XML Encryption specification, you can re-encrypt an XML-encrypted file which results in a super-encrypted XML file. But, you cannot encrypt a particular child of the `EncryptedData` or `EncryptedKey` elements using XML encryption. In other words, an `EncryptedData` element cannot be the parent or child of another `EncryptedData` element.

According to the XML Encryption specification, you can also encrypt arbitrary data (for example, a .jpg image or practically anything on the Web). This is nearly identical to the encryption of a complete XML file; the only difference is the value of the type attribute for the `EncryptedData` element (see Resources).

Cryptography
The XML Encryption specification lists a number of required and optional cryptographic algorithms including:

- Block encryption
- Stream encryption
- Key transfer
- Key agreement
- Symmetric key wrap
- Message digest
- Message authentication

The Base 64 encoding algorithm (see Resources) is also required. Every cipher must be Base 64 encoded before it can be inserted into an XML document. In this demonstration, I use the TripleDES block encryption algorithm of the SunJCE cryptography provider. The mode of encryption is CBC (Cipher Block Chaining) with 8-byte block size.

IANA values

In Part 1 of this series, I used Internet Assigned Numbers Authority (IANA) type definitions as the value of the `Type` attribute in the `EncryptedData` tag. Part 1 was based on the XML Encryption W3C Working Draft dated 18 October 2001. Recently, XML Encryption has become a W3C Candidate Recommendation, which no longer uses IANA values (see Resources).

The `XmlEncryption` class employs JCA/JCE cryptography, implemented by following methods:

- **`getEncryptedData`:** This method accepts a text string and encrypts it using the TripleDES algorithm. For symmetric encryption (see Resources), you need an encryption key in addition to the clear data being encrypted. This key is available as a private attribute, `encKey`, of the class `XmlEncryption`. The input text string is converted into an array of bytes and is padded. I use a block cipher algorithm for encryption that requires the clear data bytes to be in blocks of 8 bytes, so I pad the last incomplete block with some characters. The sample implementation uses a blank space (" ") as the character for padding.

  Next I create a `Cipher` class object and initiate it in encryption mode. I then extract the initialization vector (IV) into a byte array, block encrypt the clear data, and prefix the IV bytes to the encrypted data bytes. When I use the method `getBase64Encoded` of the class `XmlEncryption`, the resultant byte array is then transformed into a Base 64 encoded string and is subsequently returned.

- **`Decrypt`:** This method is essentially the opposite of the `getEncryptedData` method. It accepts a Base 64 encoded cipher byte array, and decodes and decrypts it to return the clear data string. The IV is then separated from the actual cipher byte array. A `Cipher` class object is created and initiated in the decryption mode. The cipher byte array is then decrypted, and the decrypted byte array is returned as a text string.

I have provided support for the TripleDES block cipher algorithm only for demonstration. With a few modifications, you can also

use other algorithms such as AES.

This simple implementation demonstrates how XML Encryption can be a viable security framework for larger and more complex business applications.

XML Encryption and SOAP

Simple Object Access Protocol, or SOAP (see Resources), is a lightweight XML-based protocol for data exchange. It facilitates the transfer of data resulting from remote procedure calls and responses. It is designed for use in distributed and remote applications, and is a primary component of Web services. SOAP provides an envelope for containing a message and its processing information. Since the content of this envelope can be confidential, security is an issue that you must address. XML Encryption provides a seamless solution to this problem.

SOAP itself is XML, which gives you the freedom to handle encryption issues in any suitable fashion using XML Encryption. For example, you may decide to encrypt the entire SOAP body or a part of the body. However, XML-SEC is a W3C effort aimed at standardizing the ways of implementing security in SOAP-based Web services. At the moment, it does not include XML Encryption; I expect XML Encryption or a similar encryption methodology to be part of it in the future.

Summary

In Part 1, I introduced XML Encryption and discussed its use for achieving secure data exchange. I elaborated on the use of different XML Encryption tags and proposed a Java API for XML Encryption based on DOM.

Here in Part 2, I have examined a typical data interchange scenario where you can employ XML encryption concepts. I also demonstrated the various types of XML encryption: encrypting a complete XML file, an element of an XML file, or the contents of an XML element. I also discussed decryption, super encryption, and the application of XML encryption concepts in SOAP-based Web services.

Resources

- Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.
- Read "Exploring XML Encryption, Part 1" of this two-part series of articles on XML Encryption by Bilal Siddiqui (*developerworks*, March 2002).
- Download the complete working code for this demonstration. The following code referenced in this article is included:
  - Readme.txt tells you how to set up the environment and run the demo
  - demoXmlEncApp.java (Listing 1)
  - XmlEncryption.java (Listing 2)
  - Order.xml (Listing 3)
  - AlgoNames.java
  - CipherData.java
  - EncryptedData.java
  - EncryptionMethod.java
  - GenericKeyInfo.java
  - Transforms.java
- Check out the W3C XML Encryption Specification Candidate Recommendation, which I have followed in this article.
- Explore Business Transaction Protocol (BTP), a flexible architecture for business transactions that is still under development.
- Read XML Digital Signature (XML-DSIG), a protocol designed for key exchange.
- Learn about Gokul Singh's Base 64 encoding/decoding classes which are used in the demonstration for this article.
- Check out the SOAP Specification which defines a protocol for data exchange in a distributed environment.

- Take a look at Doug Tidwell's developerWorks tutorial Web services -- the Web's next revolution (*developerworks*, November 2000). It offers an overview of Web services, a solution for building platform- and language-independent, interoperable distributed applications.

- Find more XML resources on the developerWorks XML technology zone.

- Get IBM WebSphere Studio Application Developer, an easy-to-use, integrated development environment for building, testing, and deploying J2EE applications, including generating XML documents from DTDs and schemas.

- Find out how you can become an IBM Certified Developer in XML and related technologies.

About the author

XML consultant Bilal Siddiqui received a degree in Electronics Engineering from the University of Engineering and Technology, Lahore, Pakistan 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 contact him at wap_monster@yahoo.com.

Implement an XML Encryption engine

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

In this second installment, Bilal Siddiqui examines the usage model of XML Encryption with the help of a use case scenario. He presents a simple demo application, explaining how it uses the XML Encryption implementation. He then continues with his last implementation of XML Encryption and makes use of JCA/JCE classes to support cryptography. Finally, he briefly discusses the applications of XML Encryption in SOAP-based Web services.

Related content:
Subscribe to the developerWorks newsletter
More dW Web architecture resources

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

Listing 1. The demoXmlEncApp class

```
 /*
        DW/BS
        20020521
        demoXmlEncApp.java
        Listing 1
        A demo application class that demonstrates XML encryption.
        It uses the class XmlEncryption.java.
 */


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


class demoXmlEncApp {

        // This main method is devised to demonstrate XML Encryption functionality.
        public static void main (String args[]) {
                if (args.length != 1) {
                        System.out.println("USAGE: java demoXmlEncApp <n>");
                        System.out.println("\t\t\tn = 1 : Complete XML-File
Encryption");
                        System.out.println("\t\t\tn = 2 : Element Encryption");
                        System.out.println("\t\t\tn = 3 : Element-Content
Encryption\n");
                }
                else {
                        // Testing the provider for TripleDES encryption support.
                        try {
                                Cipher testCipher = Cipher.getInstance("DESede");
```

```
                        }//try
                        catch(Exception e) {
                                // JCE provider not installed on this system.
                                // Installing the JCE provider.
                                System.err.println("INSTALLING PROVIDER: SunJCE");
                                Provider sunjceprov = new
com.sun.crypto.provider.SunJCE();
                                Security.addProvider(sunjceprov);
                                System.err.println("PROVIDER INSTALLED...
CONTINUING");
                }//catch

                        // Create an instance of this class.
                  demoXmlEncApp demoApp = new demoXmlEncApp();

                        // Demonstrate what happens
                        // at the book seller's (sender) end.
                  demoApp.simulateBookSellersEnd(args);

                        // Demonstrate what happens
                        // at the publisher's (recepient) end.
                  demoApp.simulatePublishersEnd();
                }// End else
        }// End main()


        void simulateBookSellersEnd(String args[]) {
                try{
                        //******BEGINNING OF PROCESSING AT THE BOOKS_SELLER'S
END******

                        // Reading the source file from disk.
                        FileInputStream fis = new FileInputStream("Order.xml");
                        byte [] aFile = new byte [fis.available()];
                        fis.read(aFile);
                        // The source file is converted into a String.
                        String aFileString = new String(aFile);


                        // The XML Encrypted file string
                        String aEncString = null;
                        // The encryption key
                        Key theKey = null;
                        // Generating the key for encryption
                        KeyGenerator KG = KeyGenerator.getInstance("DESede");
                        theKey = KG.generateKey();


                        // Creating a new XmlEncryption Object for encryption:
                        XmlEncryption aXmlEnc = new XmlEncryption();
                        // Setting the clear Document Object
                        aXmlEnc.setClearDoc(aFileString);
                        // Setting the encryption key
                        aXmlEnc.setEncKey(theKey);
                        // Setting the required algorithm
                        aXmlEnc.setAlgoName("DESede");
                        // Setting the name of the encryption key
                        aXmlEnc.setKeyName("theKey");
                        // Setting the EncryptedData tag ID
                        aXmlEnc.setEncId("Test");
                        // Selecting the XML Encryption method
                        if(args[0].equals("1"))
                                // XML Encryption of the complete XML file
                                aEncString = aXmlEnc.encryptCompleteXmlFile();
```

```
                        if(args[0].equals("2"))
                                // XML Encryption of an Element in the XML file
                                aEncString =
aXmlEnc.encryptElementOfXmlFile("CardNo");
                        if(args[0].equals("3"))
                                // XML Encryption of the content(textual)
                                // of an Element in the XML file
                                aEncString =
aXmlEnc.encryptElementContentOfXmlFile("CardNo");
                        if(args[0].equals("4"))
                                // XML Encryption of the content(Elements)
                                // of an Element in the XML file
                                aEncString =
aXmlEnc.encryptElementContentOfXmlFile("Payment");

                        // Writing the XML Encrypted file to disk
                        try {
                                byte[] aToFile = aEncString.getBytes();
                                FileOutputStream aFOS = new
FileOutputStream("aEnc.xml");
                                aFOS.write(aToFile);
                                aFOS.close();
                        }
                        catch(Exception e) {
                                System.out.println("Unable to create 'aEnc.xml'.");
                        }

                        //Writing the encryption key to a file on disk
                        try {
                                FileOutputStream keyFOS = new
FileOutputStream("theKey");
                                keyFOS.write(theKey.getEncoded());
                                keyFOS.close();
                        }
                        catch(Exception e) {
                                System.out.println("Unable to save the encryption
key.");
                        }

                                //******END OF PROCESSING AT THE BOOKS_SELLER'S
END******

                }//try
                catch(Exception e) {
                        System.out.println("Unable to read the input XML file.");
                }//catch
      }//simulateBookSellersEnd


        void simulatePublishersEnd() {
                //******BEGINNING OF PROCESSING AT THE PUBLISHER'S END******

                // The XML Encrypted file String
                String bEncString = null;
                // The decrypted file String
                String bFileString = null;
                // Reading the XML Encrypted file from disk.
                try {
                        FileInputStream bFIS = new FileInputStream("aEnc.xml");
                        byte[] bFromFile = new byte[bFIS.available()];
                        bFIS.read(bFromFile);
                        // Converting the XML Encrypted file to String
                        bEncString = new String(bFromFile);
                        bFIS.close();
```

```
                }//try
                catch(Exception e) {
                        System.out.println("Unable to read the encrypted file.");
                }//catch

                // Printing the received XML Encrypted file to the console:
                System.out.println("Order.xml AS VIEWED BY THE SALES DEPARTMENT:\n\n"
                        +bEncString);
                // Creating a new XmlEncryption Object for decryption
                XmlEncryption bXmlEnc = new XmlEncryption();
                // Decrypting the XML Encrypted file String
                bFileString = bXmlEnc.getDecryptedData(bEncString);

                // Printing the decrypted file to the console:
                System.out.println(
                        "Order.xml AS VIEWED BY THE ACCOUNTS
DEPARTMENT:\n\n"+bFileString);

                //******END OF PROCESSING AT THE PUBLISHER'S END******

      }//simulatePublishersEnd

}// End class demoXmlEncApp
```

Implement an XML Encryption engine

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

In this second installment, Bilal Siddiqui examines the usage model of XML Encryption with the help of a use case scenario. He presents a simple demo application, explaining how it uses the XML Encryption implementation. He then continues with his last implementation of XML Encryption and makes use of JCA/JCE classes to support cryptography. Finally, he briefly discusses the applications of XML Encryption in SOAP-based Web services.

Related content:
Subscribe to the developerWorks newsletter
More dW Web architecture resources

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

Listing 2. The XmlEncryption class

```
 /*
        DW/BS
        20020521
        XmlEncryption.java
        Listing 2
        A wrapper class that can generate complete XML encrypted file.
        It uses all the other classes.
        Users of the XML Encryption Engine will only need to interact with
        this class.
 */

import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.crimson.tree.XmlDocument;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import gokul.java.io.base64.*;

public class XmlEncryption {


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

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

        // ID attribute of the main encryption type structure
        private String encId = null;

        // It will be used to get a new Document objects.
        private DocumentBuilder docBuilder = null;
```

```
        // The Document object holding the XML file
        private Document clearDoc = null;

        //The encryption key
        private Key encKey = null;

        //The decryption key
        private SecretKey decKey = null;


        // The default constructor
        public XmlEncryption() {
                // Create DocumentBuilder object from DocumentBuilderFactory.
                try {
                        docBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
                }
                catch (ParserConfigurationException e) {
                        docBuilder = null;
                }
        }


        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()


        public void setClearDoc(String fileString) {
                try {
                        ByteArrayInputStream bais = new ByteArrayInputStream
                                (fileString.getBytes());
                        this.clearDoc = docBuilder.parse(bais);
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }// End setClearDoc()
```

```
        public Document getClearDoc() {
                return this.clearDoc;
        }// End getClearDoc()


        public void setEncKey(Key encKey) {
                this.encKey = encKey;
        }// End setEncKey()


        public Key getEncKey() {
                return this.encKey;
        }// End getEncKey()


        public void setDecKey(SecretKey decKey) {
                this.decKey = decKey;
        }// End setDecKey()


        public Key getDecKey() {
                return this.decKey;
        }// End getDecKey()


        // Generate a Complete XML Encrypted File.
        public String encryptCompleteXmlFile() {
                // Take an Object of EncryptedData Class.
                // It represents an <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.getString((XmlDocument)this.clearDoc);
                // 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);
                //return the resultant in a file
                return getString((XmlDocument)encDataObj.getEncData());
        }// End encryptCompleteXmlFile()


        // Generate element encrypted XML file.
        public String encryptElementOfXmlFile(String elementName) {

                // Take an Object of EncryptedData Class.
                // It represents <EncryptedData> Element.
                EncryptedData encDataObj = this.getEncryptedDataDoc(this.encId,
"ELEMENT");
                // Get XML Structure for <EncryptionMehtod> element.
                Document encMethodDoc = this.getEncryptionMethodDoc(this.algoName);
                // Get XML Structure for <KeyInfo> element.
                Document encKeyInfoDoc = this.getKeyInfoDoc(this.keyName);
                //The node object is transformend into a string.
                String plainData = this.getClearNode(elementName).toString().trim();
                // Method call to get encrypted and base 64 encoded data.
                String base64cipherData = this.getEncryptedData(plainData);
```

```
                // Get XML Structure for <CipherData> element.
                Document cipherDataDoc = this.getCipherDataDoc(base64cipherData);
                // Join these XML Structures.
                encDataObj.addChild(encMethodDoc);
                encDataObj.addChild(encKeyInfoDoc);
                encDataObj.addChild(cipherDataDoc);
                // Return the encrypted content as a Document type object.
                Document encElementDoc = encDataObj.getEncData();
                //Replace the specified element with its encrypted form
                boolean repstatus = false;
                repstatus = replaceElement(encElementDoc,elementName);
                return getString((XmlDocument)clearDoc);
        }// End encryptElementOfXmlFile()


        // Generate element content encrypted XML file.
        public String encryptElementContentOfXmlFile(String elementName) {
                // Take an Object of EncryptedData Class.
                // It represents <EncryptedData> Element.
                EncryptedData encDataObj = this.getEncryptedDataDoc(this.encId,
"CONTENT");
                // Get XML Structure for <EncryptionMehtod> element.
                Document encMethodDoc = this.getEncryptionMethodDoc(this.algoName);
                // Get XML Structure for <KeyInfo> element.
                Document encKeyInfoDoc = this.getKeyInfoDoc(this.keyName);
                //The node object is transformed into a string.
                String plainData = this.getElementContent(elementName).trim();
                // Method call to get encrypted and base 64 encoded data.
                String base64cipherData = this.getEncryptedData(plainData);
                // Get XML Structure for <CipherData> element.
                Document cipherDataDoc = this.getCipherDataDoc(base64cipherData);
                // Join these XML Structures.
                encDataObj.addChild(encMethodDoc);
                encDataObj.addChild(encKeyInfoDoc);
                encDataObj.addChild(cipherDataDoc);//
                // Return the encrypted content as a Document type object.
                Document encElementDoc = encDataObj.getEncData();
                //Replace the specified element with its encrypted form
                boolean repstatus = false;
                repstatus = replaceContent(encElementDoc, elementName);
                return getString((XmlDocument)clearDoc);
        }// End encryptElementOfXmlFile()


        // This is where the actual JCA/JCE data encryption takes place.
        public String getEncryptedData(String data) {
                String clearData = new String(data);
                String cipherDataBase64 = null;
                try {
                        //padding the last byte with " "
                        int pad = clearData.length()%8;
                        for(int i=0;i<(8-pad);i++)
                                clearData += " ";
                        // Creating a Cipher Object.
                        Cipher cipherObj =
Cipher.getInstance(this.algoName+"/CBC/NoPadding");
                        // Initializing the Cipher Object in encryption mode.
                        cipherObj.init(Cipher.ENCRYPT_MODE,this.encKey);
                        // Retreiving the IV
                        byte [] iv = cipherObj.getIV();
                        //Ciphering the clear text.
                        byte[] cipherTemp = cipherObj.doFinal(clearData.getBytes());
                        //Prefixing the IV to the cipher string to produce cipherData
```

```
                        int ivlen = iv.length;
                        int cTlen = cipherTemp.length;
                        int cDlen = ivlen+cTlen ;
                        byte [] cipherData = new byte[cDlen];
                        // First 8 bytes are IV bytes
                        for (int i=0;i<ivlen;i++)
                                cipherData[i] = (byte)iv[i];
                        // Rest of the bytes are cipher bytes
                        for (int i=ivlen;i<cDlen;i++)
                                cipherData[i] = (byte)cipherTemp[i-8];
                        //Base 64 encoding of cipherData
                        cipherDataBase64 = getBase64Encoded(cipherData);
                }
                catch (Exception e) {
                        System.out.println("Problem in getEncryptedData()");
                        e.printStackTrace();
                }
                return cipherDataBase64;
        }// End getEncryptedData()

        //Decrypting the XML Encrypted file
        public String getDecryptedData(String encString) {
                String decString = "UNDER DEVELOPMENT";
                try     {
                        //get the encrypted XML file string parsed into a Document
object
                        ByteArrayInputStream bais = new ByteArrayInputStream
                                (encString.getBytes());
                        Document encDoc = docBuilder.parse(bais);
                        //Get a list of all the EncryptedData tags
                        NodeList nl = encDoc.getElementsByTagName("EncryptedData");
                        //Load, decrypt and replace
                        //each EncryptedData tag in the Document object
                        for(int i=0;i<nl.getLength();i++) {
                                //Loading an element
                                Node edata = nl.item(i);
                                //Extracting the values of  Algorithm, KeyName,
                                //Type(of encryption) and CipherValue
                                String algo = null;
                                String keyname = null;
                                String encType = null;
                                String ciphervalue = null;
                                //Setting the values
                                edata.normalize();
                                //Setting the value of encType
                                encType = edata.getAttributes().getNamedItem("Type")
                                        .getNodeValue();
                                //Setting the values of the remaining parameters
                                NodeList algoNL = edata.getChildNodes();
                                for(int j=0;j<algoNL.getLength();j++) {
                                        //Setting the value of algo
                                        if(algoNL.item(j).getNodeName().equals
                                                ("EncryptionMethod"))
                                                algo =
algoNL.item(j).getAttributes().

getNamedItem("Algorithm").getNodeValue();
                                        //Setting the value of keyname

if(algoNL.item(j).getNodeName().equals("ds:KeyInfo")) {
                                                NodeList knNL =
algoNL.item(j).getChildNodes();
                                                for(int k=0;k<knNL.getLength();k++)
```

```
{
if(knNL.item(k).getNodeName().equals
                                                        ("KeyName"))
                                                        keyname =
(knNL.item(k).getFirstChild().
getNodeValue());
                                                }
                                        }
                                        //Setting the value of ciphervalue
if(algoNL.item(j).getNodeName().equals("CipherData")) {
                                                NodeList cvNL =
algoNL.item(j).getChildNodes();
                                                for(int v=0;v<cvNL.getLength();v++)
{
if(cvNL.item(v).getNodeName().equals
                                                        ("CipherValue"))
                                                        ciphervalue =
(cvNL.item(v)
.getFirstChild().getNodeValue());
                                                }
                                        }
                                }
                                if (algo.equals
                                        ("http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"))
                                        algo = "DESede";
                                //Reading the key file and generating/setting decKey
                                this.generateDecKey(keyname, algo);
                                //Decrypt the cipher
                                String decbit =
Decrypt(ciphervalue,this.decKey,algo).trim();
                                //Replacement Logic
                                //For replacing an entire XML file
                                if(encType.equals(
                        "http://www.isi.edu/in-notes/iana/assignments/media-
types/text/xml"))
                                        decString = decbit;
                                //Decrypted element replacement
                                else
if(encType.equals("http://www.w3.org/2001/04/xmlenc#Element")) {
                                        try     {
                                                byte [] a = decbit.getBytes();
                                                ByteArrayInputStream bais2 =
                                                        new ByteArrayInputStream(a);
                                                Document decdoc =
docBuilder.parse(bais2);
                                                Node decNode =
                                                        encDoc.importNode
(decdoc.getFirstChild(), true);
edata.getParentNode().replaceChild(decNode,edata);
                                        }
                                        catch(org.xml.sax.SAXParseException spe) {
                                                Text decText =
encDoc.createTextNode(decbit);
edata.getParentNode().replaceChild(decText,edata);
```

```
                                        }
                                        decString = (getString((XmlDocument)encDoc));
                                }
                                //Decrypted content replacement
                                else if(encType.equals
                                        ("http://www.w3.org/2001/04/xmlenc#Content"))
{
                                        try {
                                                // Works if the content is a single
child element.
                                                byte [] a = decbit.getBytes();
                                                ByteArrayInputStream bais2 =
                                                        new ByteArrayInputStream(a);
                                                Document decdoc =
docBuilder.parse(bais2);
                                                Node decNode =
                                                        encDoc.importNode
                                                        (decdoc.getFirstChild(),
true);

edata.getParentNode().replaceChild(decNode,edata);
                                        }
                                        catch(org.xml.sax.SAXParseException spe) {
                                                //In case the content is plain text
                                                //or a group of child elements
                                                Text decText =
encDoc.createTextNode(decbit);

edata.getParentNode().replaceChild(decText,edata);
                                        }
                                        decString = (getString((XmlDocument)encDoc));
                                }
                        }
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return decString;
        }// End getDecryptedData()


        // This is where the actual JCA/JCE data decryption takes place.
        private String Decrypt(String encString, Key decKey, String algo) {
                // Decoding the Base 64 Encoded IV+cipher String into a byte array
                byte[] g = getBase64Decoded(encString);
                int glen = g.length;
                // Separating the IV from the byte array
                byte [] iv = new byte[8];
                for(int t=0;t<8;t++)
                        iv[t] = g[t];
                // Separating the cipher from the byte array
                byte [] Enc = new byte[glen-8];
                for(int p=8;p<glen;p++)
                        Enc[p-8] = g[p];
                // This will hold the decrypted String
                String decString = null;
                // Decrypting the cipher and setting decString:
                try {
                        IvParameterSpec ivps = new IvParameterSpec(iv);
                        AlgorithmParameters aparam =
AlgorithmParameters.getInstance(algo);
                        aparam.init(ivps);
                    Cipher cipherObj = Cipher.getInstance(algo+"/CBC/NoPadding");
```

```
                        cipherObj.init(Cipher.DECRYPT_MODE, decKey, aparam);
                        decString = new String(cipherObj.update(Enc));
                }
                catch(Exception e) {
                        System.out.println("Problem in Decrypt()");
                        e.printStackTrace();
                }
                return decString;
        }//End Decrypt()


        //Base 64 Encoding a byte array and returning it as a String
        private String getBase64Encoded(byte[] plainText) {
                String encoded = null;
                try {
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        Base64OutputStream out= new Base64OutputStream(baos,true);
                        out.write(plainText);
                        out.close();
                        String encText = new String(baos.toByteArray());
                        baos.close();
                        encoded = encText;
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return encoded;
        }//End getBase64Encoded()

        //Base 64 Decoding a String and returning it as a byte array
        private byte[] getBase64Decoded(String codedText) {
                String ct = codedText;
                int lenEnc = ct.length();
                int lenDec = 0;
                // To calculate the max size of the decoded byte array
                if (ct.charAt(lenEnc-1) == '=')
                                if (ct.charAt(lenEnc-2) == '=') {
                                        lenDec = ((lenEnc-4)* 3/4);
                                        lenDec = ((lenDec - (lenDec/77))+1);
                                }
                                else {
                                        lenDec = ((lenEnc-4)* 3/4);
                                        lenDec = ((lenDec - (lenDec/77))+2);
                                }
                        else {
                                lenDec = (lenEnc * 3/4);
                                lenDec = (lenDec - (lenDec/77));
                        }
                byte [] decoded = new byte[lenDec];
                try {

                        ByteArrayInputStream bais = new
ByteArrayInputStream(ct.getBytes());
                        Base64InputStream in = new Base64InputStream(bais);
                        in.read(decoded);
                        in.close();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return decoded;
        }// End getBase64Decoded()

        // To generate the named Key Object and to set the decryption key
```

```
        private void generateDecKey(String keyname, String algo) {
                try {
                        FileInputStream dkFIS = new FileInputStream(keyname);
                        byte [] dk = new byte[dkFIS.available()];
                        dkFIS.read(dk);
                        dkFIS.close();
                        // Generating/setting the decryption key.
                        try {
                                SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance(algo);
                                DESedeKeySpec dkSpec = new DESedeKeySpec(dk);
                                SecretKey sKey = keyFactory.generateSecret(dkSpec);
                                this.setDecKey(sKey);
                        }
                        catch(Exception e) {
                                System.out.println("Unable to generate/set the
decryption key.");
                                e.printStackTrace();
                        }
                }
                catch(Exception e) {
                        System.out.println("Unable to set the decryption key.");
                        e.printStackTrace();
                }
        }// End generateDecKey()


        //Replace the specified element with the encrypted data element
        private boolean replaceElement(Document edDoc, String elementName) {
                boolean isReplaced = false;
                try {
                        Node clearNode = this.getClearNode(elementName);
                        Node parentNode = clearNode.getParentNode();
                        Node edNode = clearDoc.importNode(edDoc.getFirstChild(),
true);
                        parentNode.replaceChild(edNode,clearNode);
                        isReplaced = true;
                }
                catch(Exception e) {
                        isReplaced = false;
                        System.out.println("\nReplacement Failed.");
                        e.printStackTrace();
                }
                return isReplaced;
        }//End replaceElement()


        //Replace the specified element's content with the encrypted data element
        private boolean replaceContent(Document edDoc, String elementName) {
                boolean isReplaced = false;
                try {
                        if (removeAllChildNodes(elementName)) {
                                Node parentNode = this.getClearNode(elementName);
                                //Importing the encryptedData Node
                                Node edNode =
clearDoc.importNode(edDoc.getFirstChild(), true);
                                //Appending to an already empty parentNode
                                parentNode.appendChild(edNode);
                                isReplaced = true;
                        }
                        else
                                System.out.println(
                                        "Child nodes not removed; replacement did not
```

```
initiate.");
                }
                catch(Exception e) {
                        isReplaced = false;
                        System.out.println("\nContent replacement failed.");
                        e.printStackTrace();
                }
                return isReplaced;
        }//End replaceContent()


        //Search the document tree and return the specified element as a Node Object.
        private Node getElement(Document doc, String elementName) {
                NodeList nList = doc.getElementsByTagName(elementName);
                Node theNode = nList.item(0);
                return theNode;
        }// End getElement();


        //Get a clear node by its element name
        private Node getClearNode(String elementName) {
                return getElement(this.clearDoc, elementName);
        }// End getClearNode()


        //Get a clear node's content by its element name
        public String getElementContent(String elementName) {
                String elementContent = new String();
                NodeList nl = this.getClearNode(elementName).getChildNodes();
                for(int i=0;i<nl.getLength();i++)
                        elementContent += (nl.item(i)).toString().trim();
                return elementContent;
        }// End setElementContent()


        //Remove all the child nodes of clearNode
        private boolean removeAllChildNodes(String elementName) {
                boolean removed = false;
                Node clearNode = this.getClearNode(elementName);
                try     {
                        while(clearNode.hasChildNodes()) {
                                NodeList t = clearNode.getChildNodes();
                                // Removing a child Node
                                clearNode.removeChild(t.item(0));
                        }//All children removed
                        removed = true;
                }
                catch(Exception e) {
                        removed = false;
                        e.printStackTrace();
                }
                return removed;
        }//End removeAllChildNodes()


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


        // 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);
                if (encType.equals("ELEMENT"))
                        ed.setType(AlgoNames.ELEMENT);
                if (encType.equals("CONTENT"))
                        ed.setType(AlgoNames.CONTENT);
                return ed;
        }// End getEncryptedDataDoc()


        // Returns the <EncryptionMehtod> structure.
        public Document getEncryptionMethodDoc (String algoName) {
                EncryptionMethod em = new EncryptionMethod(this.getNewDocument());
                if (algoName.equals("DESede"))
                        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()


        //Convert an XmlDocument type object into a String
        public String getString(XmlDocument xmldoc) {
                String DocumentAsString = null;
                try     {
                        ByteArrayOutputStream bos = new ByteArrayOutputStream();
                        xmldoc.write(bos);
                        DocumentAsString = bos.toString();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return DocumentAsString;
        }//End getString()


        //Convert a Document type object into a String
        //The output string does not contain initial encoding tags
        public String getString(Document doc) {
                String DocumentAsString = null;
                try     {
                        if (doc.getFirstChild().getNodeName().equals("#comment")) {
                                DocumentAsString = "<!--
"+doc.getFirstChild().getNodeValue()+
                                        "-->\n\n";
```

```
                                DocumentAsString +=
doc.getDocumentElement().toString();
                        }
                        else
                                DocumentAsString =
doc.getDocumentElement().toString();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return DocumentAsString;
        }//End getString()


}// End class demoXmlEncApp
```