Passing credentials using WS-Security headers

When invoking a LiveCycle ES service using web services, you can use WS-Security headers to pass client authentication information that is required by LiveCycle ES. WS-Security defines SOAP extensions to implement client authentication, message confidentiality, and message integrity. As a result, you can invoke LiveCycle ES services when LiveCycle ES is deployed as stand-alone server or within a clustered environment.

How you pass WS-Security headers to LiveCycle ES depends on whether you are using Axis-generated Java classes or a .NET client assembly that consumes a service's native SOAP stack.

Note: As an example of invoking a service using WS-Security headers, this topic encrypts a PDF document with a password by invoking the Encryption service.

Passing client authentication using Axis-generated Java classes

To pass client authentication information when using Axis-generated Java classes, you can use WS-Security Axis handlers that require the following WSS4J (Web Service Security for Java) libraries:

- WSS4J
- XML Security

You can download these libraries at http://ws.apache.org/wss4j/package.html.

After you down the libraries, include the following JAR files in your class path:

- wss4j-1.5.1.jar
- commons-logging.jar
- xmlsec-1.3.0.jar

Deployment descriptor file

When using a WS-Security header to pass client authentication to LiveCycle ES, you must create a deployment descriptor file (client_deploy.wsdd). This file contains information such as the user name and the name of a security callback class that is used to send a password to LiveCycle ES.

The following code shows the format of the client_deploy.wsdd file:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http"
pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
   <requestFlow >
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
     <parameter name="action" value="UsernameToken"/>
     <parameter name="passwordType" value="PasswordText"/>
     <parameter name="user" value="administrator"/>
     <parameter name="passwordCallbackClass" value="PWCallBack"/>
    </handler>
```

```
   </requestFlow >
  </globalConfiguration >
</deployment>
```

Callback class

You must also create a callback class whose name is specified in the client_deploy.wsdd file. Notice that the name of the callback class in the client_deploy.wsdd file is `PWCallBack` and the user name is `administator`. This callback class must implement the `javax.security.auth.callback.CallbackHandler` interface.

The following Java code shows the syntax of the `PWCallBack` class:

```
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.ws.security.WSPasswordCallback;

public class PWCallBack implements CallbackHandler {

   public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
     for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof WSPasswordCallback) {
          WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];

          // set the password given a username
          if ("administrator".equals(pc.getIdentifer())) {
            pc.setPassword("password");
          }
        } else {
          throw new UnsupportedCallbackException(callbacks[i], "Unrecognized
Callback");
        }
     }
   }
}
```

Generating Axis library files required to invoke the Encryption service

You can generate Axis Java library files that consume the Encryption service WSDL by performing the following steps:

1. Install Apache Ant on the client computer. It is available at http://ant.apache.org/bindownload.cgi.
   - Add the bin directory to your class path.
   - Set the `ANT_HOME` environment variable to the directory where you installed Ant.
2. Install Apache Axis 1.4 on the client computer. It is available at http://ws.apache.org/axis/.
3. Set up the class path to use the Axis JAR files in your web service client, as described in the Axis installation instructions at http://ws.apache.org/axis/java/install.html.

4. Use the Apache WSDL2Java tool in Axis to generate Java proxy classes. You must create an Ant build script to accomplish this task. The following script is a sample Ant build script named *build.xml*:

```
<?xml version="1.0"?>
<project name="axis-wsdl2java">

<path id="axis.classpath">
<fileset dir="C:\axis-1_4\lib" >
  <include name="**/*.jar" />
</fileset>
</path>

<taskdef resource="axis-tasks.properties" classpathref="axis.classpath" />

<target name="repository-wsdl2java-client" description="task">
<axis-wsdl2java
  output="C:\JavaFiles"
  testcase="false"
  serverside="false"
  verbose="true"
  username="administrator"
  password="password"
  url="http://localhost:8080/soap/services/EncryptionService?WSDL" >
</axis-wsdl2java>
</target>

</project>
```

   Within this Ant build script, notice that the `url` property is set to reference the Repository WSDL running on localhost. The `username` and `password` properties must be set to a valid LiveCycle ES user name and password.

5. Create a BAT file to execute the Ant build script. The following command can be located within a BAT file that is responsible for executing the Ant build script:

```
ant -buildfile "build.xml" encryption-wsdl2java-client
```

This Ant build script generates JAVA files that can invoke the Encryption service. The JAVA files are written to the C:\JavaFiles folder as specified by the `output` property. To successfully invoke the Encryption service, you must import all of these JAVA files into your class path. By default, these files belong to a Java package named `com.adobe.idp.services`. It is recommended that you place all of these CLASS files into a JAR file and then import the JAR file into your client application's class path.

Amend the URL in the `EncryptionServiceServiceLocator` class to include `?blob=base64` to ensure that the `BLOB` object returns binary data. That is, in the `EncryptionServiceServiceLocator` class, locate the following line of code:

```
"http://localhost:8080/soap/services/EncryptionService";
```

and change it to this line:

```
"http://localhost:8080/soap/services/EncryptionService?blob=base64";
```

You must also add the following Axis JAR files to your Java project's class path:

- activation.jar
- axis.jar
- commons-codec-1.3.jar

- commons-collections-3.1.jar
- commons-discovery.jar
- commons-logging.jar
- dom3-xml-apis-2.5.0.jar
- jai_imageio.jar
- jaxen-1.1-beta-9.jar
- jaxrpc.jar
- log4j.jar
- mail.jar
- saaj.jar
- wsdl4j.jar
- xalan.jar
- xbean.jar
- xercesImpl.jar

These JAR files are in the [install directory]/Adobe/LiveCycle8/sdk/lib/thirdparty directory.

Caution: Make sure that you added `?blob=base64` to the URL in the `EncryptionServiceServiceLocator` class. Otherwise, you cannot retrieve binary data from the `BLOB` object.

Invoking the Encryption service using a WS-Security header

To invoke the Encryption service using Axis-generated library files and passing a WS-Security header, perform the following steps:

1. Create Java proxy classes that consume the Encryption service WSDL. (See Generating Axis library files required to invoke the Encryption service.)
2. Include the Java proxy classes into your class path.
3. Add the WSS4J (Web Service Security for Java) library files to your class path. (See Passing client authentication using Axis-generated Java classes.)
4. Create a deployment descriptor file (client_deploy.wsdd). (See Deployment descriptor file.)
5. Create a callback class that implements the `javax.security.auth.callback.CallbackHandler` interface. (See Callback class.)
6. Create a `java.io.FileInputStream` object by using its constructor and passing the location of the deployment descriptor file.
7. Create an `org.apache.axis.EngineConfiguration` object by using the `org.apache.axis.configuration.FileProvider` constructor and passing the `java.io.FileInputStream` object.
8. Create an `EncryptionServiceServiceLocator` object by using its constructor and passing the `org.apache.axis.EngineConfiguration` object.

9. Create a `EncryptionService` object by invoking the `EncryptionServiceServiceLocator` object's `getEncryptionService` method.

10. Create a `BLOB` object by using its constructor. The `BLOB` object is used to store a PDF document that is encrypted with a password.

11. Create a `java.io.File` object by invoking its constructor and passing a string value that represents the file location of the PDF document to encrypt.

12. Create a `java.io.FileInputStream` object by using its constructor and passing the `java.io.File` object that references the PDF file.

13. Create a byte array that stores the content of the `java.io.FileInputStream` object. You can determine the size of the byte array by invoking the `java.io.FileInputStream` object's `available` method.

14. Populate the byte array with stream data by invoking the `java.io.FileInputStream` object's `read` method and passing the byte array.

15. Populate the `BLOB` object by invoking its `setBinaryData` method and passing the populated byte array.

16. Create a `PasswordEncryptionOptionSpec` object by using its constructor.

17. Specify the password value that lets a user open the encrypted PDF document by invoking the `PasswordEncryptionOptionSpec` object's `setDocumentOpenPassword` method and passing a string value that represents the open password.

18. Specify the Acrobat compatibility option by invoking the `PasswordEncryptionOptionSpec` object's `setCompatability` method and passing a `PasswordEncryptionCompatability` enum value. For example, pass `PasswordEncryptionCompatability.ACRO_7` to encrypt the document by using the Advanced Encryption Standard.

19. Specify the password value that lets a user remove encryption from the PDF document by invoking the `PasswordEncryptionOptionSpec` object's `setPermissionPassword` method and passing a string value that represents the permission password.

20. Specify the PDF document resources to encrypt by invoking the `PasswordEncryptionOptionSpec` object's `setEncryptOption` method and passing a `PasswordEncryptionOption` enum value. To encrypt the entire PDF, include its metadata and attachments, and pass the value `PasswordEncryptionOption.ALL`.

21. Encrypt the PDF document by invoking the `EncryptionServiceService` object's `encryptPDFUsingPassword` method and passing the following values:

    - The `BLOB` object that contains the PDF document to encrypt
    - The `PasswordEncryptionOptionSpec` object that contains encryption run-time options

    The `encryptPDFUsingPassword` method returns a `BLOB` object that contains a password-encrypted PDF document.

22. Create a `java.io.File` object by invoking its constructor and passing a string value that represents the file location of the secured PDF document.

23. Create a byte array that stores the data content of the `BLOB` object that was returned by the `encryptPDFUsingPassword` method. Populate the byte array by invoking the `BLOB` object's `getBinaryData` method.

24. Create a `java.io.FileOutputStream` object by invoking its constructor and passing the `java.io.File` object that represents the secured PDF document.

25. Write the contents of the byte array to a PDF file by invoking the `java.io.FileOutputStream` object's `write` method and passing the byte array.

Example: *Invoking the Encryption service using Axis-generated files and a WS-Security header*

The following Java code example uses Axis-generated Java files and a WS-Security header to encrypt a PDF document with a password:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import org.apache.axis.EngineConfiguration;
import org.apache.axis.configuration.FileProvider;
import com.adobe.idp.services.*;

public class EncryptDoc {

   public static void main(String[] args) {
   try
      {

      //Authenticate the user using a WS-Security header
      FileInputStream wsSecurity = new
FileInputStream("C:\\Adobe\\client_deploy.wsdd");
      EngineConfiguration config = new FileProvider(wsSecurity);

      //Create an EncryptionService object
      EncryptionServiceServiceLocator sl = new
EncryptionServiceServiceLocator(config);
      EncryptionService encryptionOb = sl.getEncryptionService();

      //Create a BLOB object
      BLOB fileBlob = new BLOB();

      //Create a File object
      File myFile = new File("C:\\Map.pdf");

      //Create an InputStream object
      FileInputStream fileInput = new FileInputStream(myFile);

      //Create a byte array and populate it with stream data
      int size = fileInput.available();
      byte []myByte = new byte[size];
      fileInput.read(myByte);

       //Populate the BLOB object
       fileBlob.setBinaryData(myByte);
```

```
     //Create a PasswordEncryptionOptionSpec
     PasswordEncryptionOptionSpec passSpec = new
PasswordEncryptionOptionSpec();
     passSpec.setDocumentOpenPassword("AdobeOpen");
     passSpec.setCompatability(PasswordEncryptionCompatability.ACRO_7);
     passSpec.setPermissionPassword("AdobeMaster");
     passSpec.setEncryptOption(PasswordEncryptionOption.ALL);

     //Secure a PDF with a password
     BLOB noPassDoc =
encryptionOb.encryptPDFUsingPassword(fileBlob,passSpec);

     //Write the encrypted PDF data stream to a local file
     File outFile = new File("C:\\MapPasswordEncrypt.pdf");
     byte [] securedDocument = noPassDoc.getBinaryData();

     //Create a Java FileOutputStream object
     FileOutputStream myOutput = new FileOutputStream(outFile) ;

     //Write the byte array contents to the PDF file
     myOutput.write(securedDocument);
     myOutput.close();
      }

  catch (Exception e) {
     e.printStackTrace();
     }
  }
}
```

Passing client authentication using a .NET client assembly

To pass client authentication information when using a .NET client assembly, you can use Web Services Enhancements (WSE) 3.0 for Microsoft .NET. To do so, you must install WSE on your development computer and integrate it with Microsoft Visual Studio .NET. You can download WSE from www.microsoft.com/downloads/search.aspx.

From this web page, search for Web Services Enhancements 3.0 and download it onto your development computer. This places a file named *Microsoft WSE SPI.msi* on your computer. Run the installation program and follow the online instructions.

Creating a proxy class

Create a proxy class that is used to create the .NET client assembly by using a tool that accompanies Microsoft Visual Studio. The name of the tool is *wsdl.exe* and is located in the Microsoft Visual Studio installation folder. To create a proxy class, open the command prompt and navigate to the folder that contains the wsdl.exe file. Enter the following command at the command prompt:

```
wsdl http://localhost:8080/soap/services/EncryptionService?WSDL
```

By default, this tool creates a CS file in the same folder that is based on the name of the WSDL. In this situation, it creates a CS file named *EncryptionServiceService.cs*. You use this CS file to create a proxy object that lets you invoke the service that was specified in the WSDL definition.

Amend the WSDL definition in the proxy class to include `?blob=base64` to ensure that the `BLOB` object returns binary data; that is, in the proxy class, locate the following line of code:

```
"http://localhost:8080/soap/services/EncryptionService";
```

and change it to this line:

```
"http://localhost:8080/soap/services/EncryptionService?blob=base64";
```

Note: For more information about the wsdl.exe tool, see MSDN Help.

Caution: Make sure that you added `?blob=base64` to the URL in the proxy class that is used to create the .NET client assembly. Otherwise, you cannot retrieve binary data from the `BLOB` object.

Creating the .NET client assembly

Create a new Visual Studio Class Library project that produces a .NET client assembly. You can import the CS file that you created using the wsdl.exe tool. You must also enable this project to use Web Services Enhancements. This project will produce a DLL file that you can use in other Visual Studio .NET projects to invoke the Encryption service.

➤ **To create the .NET client assembly:**

1. Start Microsoft Visual Studio .NET.
2. Create a new Class Library project and name it **EncryptionService**.
3. Import the CS file that you created using Webservice Studio.
4. Enable the project for WSE by right-clicking in the Solution Explorer panel and selecting **WSE Settings 3.0**.
5. Select **Enable This Project for Web Services Enhancements**, and click **OK**.

Tip: Instead of importing the CS file into your Class Library project, you can copy the contents of the file and paste it into your project.

➤ **To reference the WSE library:**

1. In the **Project** menu, select **Add Reference**.
2. In the Add Reference dialog box, select **Microsoft.Web.Services3.dll**.
3. Select **System.Web.Services.dll**.
4. Click **Select** and then click **OK**.

➤ **To compile the .NET client assembly:**

1. In the EncryptionServiceService.cs file, change the class definition from the following text:

```
public class EncryptionService :
System.Web.Services.Protocols.SoapHttpClientProtocol
```

to this text:

```
public class EncryptionService :
Microsoft.Web.Services3.WebServicesClientProtocol
```

2. Compile and link the project.

Note: This procedure creates a .NET client assembly named *EncryptionService.dll*.

Referencing the .NET client assembly

Place your newly-created .NET client assembly on the computer where you are developing your client application. After you place the .NET client assembly in a directory, you can reference it from a project. You must also reference the `System.Web.Services` and `Microsoft.Web.Services3` libraries from your project. If you do not reference these libraries, you cannot use the .NET client assembly to invoke a service.

➤ **To reference the .NET client assembly:**

1. In the **Project** menu, select **Add Reference**.
2. Click the **.NET** tab.
3. Click **Browse** and locate the EncryptionService.dll file.
4. Click **Select** and then click **OK**.

Invoking the Encryption service using a WS-Security header

To encrypt a PDF document with a password by using a .NET client assembly and passing a WS-Security header, perform the following steps:

1. Using the Microsoft .NET client assembly, create an `EncryptionServiceService` object by invoking its default constructor.
2. Create a `UsernameToken` object that represents security credentials by using its constructor. Within the constructor, specify the LiveCycle ES user name and the corresponding password, as well as a `PasswordOption` enumeration, which specifies how the password is sent. For information about a `PasswordOption` enumeration, see MSDN Help.
3. Use the `EncryptionServiceService` object's `RequestSoapContext` property to set the following SOAP request properties:
   - `SoapContext.Security.TimeStamp`: The timestamp located in a SOAP message header
   - `SoapContext.Security.Tokens`: A collection of security tokens
   - `SoapContext.Security.MustUnderstand`: Specifies whether the SoapHeader must be understood
4. Create a `BLOB` object by using its constructor. The `BLOB` object is used to store a PDF document that is encrypted with a password.
5. Create a `System.IO.FileStream` object by invoking its constructor and passing a string value that represents the file location of the PDF document to encrypt and the mode in which to open the file.
6. Create a byte array that stores the content of the `System.IO.FileStream` object. You can determine the size of the byte array by getting the `System.IO.FileStream` object's `Length` property.
7. Populate the byte array with stream data by invoking the `System.IO.FileStream` object's `Read` method and passing the byte array, the starting position, and the stream length to read.

8. Populate the `BLOB` object by assigning its `binaryData` property with the contents of the byte array.

9. Create a `PasswordEncryptionOptionSpec` object by using its constructor.

10. Specify the PDF document resources to encrypt by assigning a `PasswordEncryptionOption` enum value to the `PasswordEncryptionOptionSpec` object's `encryptOption` data member. To encrypt the entire PDF, including its metadata and its attachments, assign `PasswordEncryptionOption.ALL` to this data member.

11. Specify the Acrobat compatibility option by assigning a `PasswordEncryptionCompatability` enum value to the `PasswordEncryptionOptionSpec` object's `compatability` data member. For example, assign `PasswordEncryptionCompatability.ACRO_7` to this data member.

12. Specify the password value that lets a user open the encrypted PDF document by assigning a string value that represents the open password to the `PasswordEncryptionOptionSpec` object's `documentOpenPassword` data member.

13. Specify the password value that lets a user remove encryption from the PDF document by assigning a string value that represents the master password to the `PasswordEncryptionOptionSpec` object's `permissionPassword` data member.

    Encrypt the PDF document by invoking the `EncryptionServiceService` object's `encryptPDFUsingPassword` and passing the following values:

    - The `BLOB` object that contains the PDF document to encrypt with the password
    - The `PasswordEncryptionOptionSpec` object that contains encryption run-time options

    The `encryptPDFUsingPassword` method returns a `BLOB` object that contains a password-encrypted PDF document.

14. Create a `System.IO.FileStream` object by invoking its constructor and passing a string value that represents the file location of the secured PDF document.

15. Create a byte array that stores the data content of the `BLOB` object that was returned by the `encryptPDFUsingPassword` method. Populate the byte array by getting the value of the `BLOB` object's `binaryData` data member.

16. Create a `System.IO.BinaryWriter` object by invoking its constructor and passing the `System.IO.FileStream` object.

17. Write the contents of the byte array to a PDF file by invoking the `System.IO.BinaryWriter` object's `Write` method and passing the byte array.

Example: *Invoking the Encryption service using a .NET client assembly and a WS-Security header*

The following C# code example uses .NET client assembly and a WS-Security header to encrypt a PDF document with a password:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.IO ;
using Microsoft.Web.Service3;
using Microsoft.Web.Services3.Security.Tokens;

namespace PasswordEncryptPDF
```

```
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         try
         {
            //Create an EncryptionServiceService client object
            EncryptionServiceService  encryptionClient = new
EncryptionServiceService();

            //Specify the LiveCycle ES user name and password
            String auth_uid = "administrator";
            String auth_pwd = "password";

            UsernameToken userToken = new UsernameToken(auth_uid,
auth_pwd,PasswordOption.SendPlainText);
            encryptionClient.RequestSoapContext.Security.Timestamp.TtlInSeconds
= 60;
            encryptionClient.RequestSoapContext.Security.Tokens.Add(userToken);
            encryptionClient.RequestSoapContext.Security.MustUnderstand =
false;


            //Create a BLOB object to store the PDF document
            BLOB inDoc = new BLOB();

            //Specify the PDF document to encrypt with a password
            string path = "C:\\Adobe\\Loan.pdf";
            FileStream fs = new FileStream(path, FileMode.Open);

            //Get the length of the file stream
            int len = (int)fs.Length;
            byte[] ByteArray=new byte[len];

            //Populate the byte array with the contents of the FileStream object
            fs.Read(ByteArray, 0, len);
            inDoc.binaryData = ByteArray;

            //Create a PasswordEncryptionOptionSpec
            //object that stores encryption run-time values
            PasswordEncryptionOptionSpec passSpec = new
PasswordEncryptionOptionSpec();

            //Specify the PDF document resource to encrypt
            passSpec.encryptOption=PasswordEncryptionOption.ALL;

            //Specify the Acrobat version
            passSpec.compatability =  PasswordEncryptionCompatability.ACRO_7;

            //Specify the password values
            passSpec.documentOpenPassword = "OpenPassword";
            passSpec.permissionPassword = "PermissionPassword";
```

```
            //Encrypt the PDF document with a password
            BLOB outDoc =
encryptionClient.encryptPDFUsingPassword(inDoc,passSpec);

            //Populate a byte array with a BLOB data
            byte[] outByteArray=outDoc.binaryData;

            //Create a new file that represents the encrypted PDF document
            string FILE_NAME = "C:\\Adobe\\EncryptLoan.pdf" ;
            FileStream fs2 = new FileStream(FILE_NAME, FileMode.OpenOrCreate);

            //Create a BinaryWriter object
            BinaryWriter w = new BinaryWriter(fs2);
            w.Write(outByteArray);
            w.Close();
            fs2.Close();
          }
          catch (Exception ee)
          {
            Console.WriteLine(ee.Message);
          }
        }
    }
}
```