BellSouth Interconnection Services
675 West Peachtree Street
Atlanta, Georgia 30375

Carrier Notification
SN91084103

Date: May 24, 2004

To: Competitive Local Exchange Carriers (CLEC)

Subject: CLECs – (OSS) – Upgrade to Telecommunications Access Gateway (TAG) Extensible Markup Language (XML) Digital Signature (DSIG) Product

This is to advise that with the implementation of ENCORE Release 16.0 on July 25, 2004, BellSouth will be upgrading its TAG XMLDSIG software to Betrusted KeyTools Version 5.2. This enhancement is to bring the interface into compliance with the World Wide Web Consortium (W3C) XMLDSIG guidelines. Please refer to the attachment for additional details regarding the enhancement.

This change affects only TAG XML customers who are using HTTPS in conjunction with non-compliant signing software. It affects both LAN-to-LAN and Internet HTTPS customers. It does not affect TAG XML customers who use Common Object Request Broker Architecture (CORBA) over Secure Socket Layer (SSL). It does not affect Electronic Data Interchange (EDI), EDI Interaction Agent (IA) or Local Exchange Navigation System (LENS) customers.

Please contact your BellSouth e-commerce account manager with any questions.

Sincerely,

ORIGINAL SIGNED BY PAT FINLEN FOR JERRY HENDRIX

Jerry Hendrix – Assistant Vice President
BellSouth Interconnection Services

Attachment

Attachment
SN91084103

Wholesale Customer Notification for SGG Digital Signature Software Change

Consistent with discussions at the monthly Telecommunications Access Gateway (TAG) CLEC Forum, BellSouth continues to incorporate changes into TAG when necessary to maintain compliance with the World Wide Web Consortium (W3C) Extensible Markup Language (XML) guidelines. BellSouth has incorporated such a change in the ENCORE Release 16.0. This change was made to rectify a non-compliance issue with the third party software previously used by TAG to digitally sign its HTTPS XML response instances and to verify signed input XML instances it receives. BellSouth has replaced the non-compliant software with the Betrusted KeyTools Version 5.2. This has corrected the W3C XML Digital Signature (DSIG) non-compliance issues.

This change affects only TAG XML customers who are using HTTPS in conjunction with non-compliant signing software. It affects both LAN-to-LAN and Internet HTTPS customers. It does not affect TAG XML customers who use Common Object Request Broker Architecture (CORBA) over Secure Socket Layer (SSL). It does not affect Electronic Data Interchange (EDI), EDI Interaction Agent (IA) or Local Exchange Navigation System (LENS) customers.

Affected customers should verify that their signing software is W3C XMLDSIG compliant prior to ENCORE Release 16.0 production date. This will ensure the customers can submit signed XML inputs into and can verify signed XML responses from the ENCORE Release 16.0. BellSouth has coded and tested TAG XML clients with four commercially or publicly available signing software products to determine whether they could be used with the new ENCORE Release 16.0 Betrusted KeyTools implementation. Specifically, BellSouth tested the following four products:

- Verisign's Trust Services Integration Kit (TSIK) Version 1.5
- Capicom.com Version 2.0.0.1
- Apache XML Security 1.1
- Betrusted KeyTools XML 5.2

Verisign's Trust Services Integration Kit (TSIK) Version 1.5 has been used for illustration purposes in previous TAG XML training classes. Testing demonstrated that messages signed using this TSIK product cannot be verified by the new W3C XMLDSIG compliant digital signature implementation. Therefore, beginning with ENCORE Release 16.0, any customer using this TSIK signing software in its TAG XML HTTPS client must migrate to a product that is XMLDSIG compliant to ensure the compatibility with ENCORE Release 16.0.

Testing demonstrated that any of the other three products could be verified by the new W3C XMLDSIG compliant digital signature implementation. These were Capicom.com Version 2.0.0.1, Betrusted KeyTools XML 5.2 and Apache XML Security 1.1. All three

have been successfully tested with ENCORE 16.0. Capicom.com is freeware available for download from URL:

https://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=860EE43A-A843-462F-ABB5-FF88EA5896F6

Capicom.com supports Microsoft Visual Studio.NET client applications.

Apache XML Security 1.1 is freeware available for download from URL:

http://xml.apache.org

It supports JAVA client implementations and is therefore equally usable on HP, Sun and NT clients.

Betrusted KeyTools XML 5.2 can be purchased and downloaded from URL:

http://www.betrusted.com/

There may be other equally suitable alternatives, but these three have been tested successfully for use with ENCORE Release 16.0.

To assist any customers who may need to migrate from a JAVA client implementation using Verisign's Trust Services Integration Kit (TSIK) Version 1.5 product, which was one of the implementations previously used for demonstration purposes in past TAG XML training classes, Attachment A contains the sample source code from that class that has been modified to successfully use Betrusted KeyTools XML 5.2. Attachment B contains the same class sample source code that has been modified to successfully use Apache XML Security 1.1. Change Tracking is turned on in both attachments for easy viewing of what changes were required to complete each migration.

Customers who wish to arrange for additional technical support and/or testing services should contact their BellSouth eCommerce Account Manager or Software Vendor Process Manager.

Attachment A
Attachment B

Attachment A

Sample Source Code

Migration from

Verisign's Trust Services Integration Kit (TSIK) Version 1.5

To

Betrusted KeyTools XML 5.2

```
/*
 * HTTPSTestClient.java
 *
 */

import java.net.*;/*
 * HTTPSTestClient.java
 *
 */

import java.net.*;

import java.io.*;

import java.io.*;
import java.net.*;

//package doc.examples.dsig;


import java.util.*;
import java.security.*;
import java.security.cert.*;
import java.security.cert.Certificate;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;


import java.security.spec.*;
import java.security.interfaces.*;

import com.sun.net.ssl.*;
import com.sun.net.ssl.internal.www.protocol.https.*;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.verisign.resource.ResourceFactory;
import com.verisign.resource.XMLResource;
import com.verisign.xmlsig.Signer;
import com.verisign.xmlsig.Verifier;
import com.verisign.xmlsig.SigningKey;
import com.verisign.xmlsig.VerifyingKey;
```

```
import com.verisign.xpath.XPath;
import com.verisign.domutil.DOMCursor;
import com.verisign.domutil.DOMWriteCursor;
com.baltimore.jcrypto.provider.*;
import com.baltimore.helper.xml.dsig.*;
import com.baltimore.xml.dsig.*;
import com.baltimore.util.xml.*;
import com.baltimore.util.collection.*;
import com.baltimore.util.xml.*;
import com.baltimore.util.xml.spec.*;
import com.baltimore.xml.c14n.*;
import com.baltimore.xml.c14n.spec.*;
import com.baltimore.xml.dsig.*;
import com.baltimore.xml.dsig.canonicalization.*;
import com.baltimore.xml.dsig.context.*;
import com.baltimore.xml.dsig.keyinfo.*;
import com.baltimore.xml.dsig.keyinfo.x509data.*;
import com.baltimore.xml.dsig.misc.*;
import com.baltimore.xml.dsig.object.*;
import com.baltimore.xml.dsig.transform.*;
import com.baltimore.xml.misc.*;


public class HTTPSTestClient
{
        public static String theURL          = null;
        public static String inputXML = null;
        public static String keyAlias = null;
        public static String keyPass  = null;
        public static String cPass           = null;
        public static String clientStoreName = null;
        public static String tPass           = null;
        public static String trustStoreName  = null;
        public static String log             = null;               // optional - uses default value
        public static String signXML  = null;              // optional - uses default value


        public static String logData         = "";
        public String tmplogData                    = null;
        public static String rspFilename     = null;
        public static String errorMsg        = null;
        public static boolean validFilename;

        public KeyStore                      clientStore;
        public KeyStore                      trustStore;
        public X509Certificate cert;
```

```
    public static HTTPSTestClient _instance;
    private URL                          _url;

    private void HTTPSTestClient()
    {
            _url = null;
            // Next 2 lines required to get past BST firewall, if needed
            //System.setProperty("https.proxyHost", "proxy.bst.bls.com");
            //System.setProperty("https.proxyPort", "8080");
    }

    public static HTTPSTestClient createInstance()
    {
            HTTPSTestClient ret = new HTTPSTestClient();
            return ret;
    }


/* Usage is as follows:
    ** -url        (URL)
    ** -input      (Input XML name)
    ** -alias      (key alias)
    ** -pass       (key password)
    ** -keystore           (keystore name)
    ** -keystorepass       (keystore password)
    ** -truststore         (truststore name)
    ** -truststorepass  (truststore password)
    ** -sign       (sign XML)                          // optional - default = yes
    ** -log        (log program progress) // optional - default = no
    */
    public static void usage()
    {
            System.out.println("Input parameter list incorrect...");
            System.out.println("Required USAGE is as follows:");
            System.out.println("   [-url]         your HTTPS URL");
            System.out.println("   [-input]       input XML (path)file");
            System.out.println("   [-alias]       key alias");
            System.out.println("   [-pass]        key password");
            System.out.println("   [-keystore]           keystore name");
            System.out.println("   [-keystorepass] keystore password");
            System.out.println("   [-truststore]  truststore name");
            System.out.println("   [-truststorepass]     truststore password");
    }

    /****/
    // Called if input param "-log" set to "yes".
    // Stores (as String) program progress / errors,
```

```
// used for screen putput and written to file at program termination.
public void doLog(String temp)
{
        System.out.println(temp);
        temp += "\n";
        logData += temp;
        temp = null;
}

/****/
// Loads XML input file
public static String loadFile(String fileName)
{
        File inFile = new File(fileName);
        inFile.canRead();

        String inXML = "";
        try
        {
                FileInputStream fileStream = new FileInputStream(inFile);

                byte[] buffer = new byte[(int)inFile.length()];
                int nBytes = fileStream.read(buffer);

                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Reading input file.");
                }
                inXML = new String(buffer);
        }
        catch (FileNotFoundException e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: FileInputStream: " + errorMsg);
                }
                else
                {
                        System.out.println("ERROR: FileInputStream: " + errorMsg);
                }
                return null;
        }
        catch (java.io.IOException e)
        {
                errorMsg = null;
```

```
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: FileInputStream: " + errorMsg);
                }
                else
                {
                        System.out.println("ERROR: FileInputStream: " + errorMsg);
                }
                return null;
        }
        inXML.trim();
        return inXML;
}

/****/
// Creates new file name for signed and response files.
// Adds "Signed", "Rsp" respectively, to input file name.
public String getNewFilename(String type, String aFilename)
{
        String newName = null;
        int index = aFilename.lastIndexOf(".");
        newName = aFilename.substring(0, index);
        if (type.compareTo("rsp") == 0)                     // new response xml file name
        {
                newName = newName + "Rsp.xml";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Created new response file name: "+ newName);
                }
        }
        else                                                                    // new signed xml file name
        {
                newName = newName + "Signed.xml";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Created new signed file name: "+ newName);
                }
        }
        return newName;
}

/***/
// Actual server communication here. Reads input file, establishes connection,
// sets connection params, writes response file
public void doTransaction (String urlName, String fileName) throws Exception
{
```

```
String inXML = loadFile(fileName);
java.net.HttpURLConnection _connection;
if (log.compareTo("yes") == 0)
{
	_instance.doLog("In doTransaction.");
}
try
{
	_url = new URL(urlName);
}
catch(MalformedURLException ue)
{
	errorMsg = ue.getMessage();
	System.out.println(errorMsg);
}
catch(Exception e)
{
	errorMsg = null;
	errorMsg = e.getMessage();
	if (log.compareTo("yes") == 0)
	{
		_instance.doLog("ERROR: URL creation: " + errorMsg);
	}
	System.out.println(errorMsg);
	e.printStackTrace();
	throw e;
}

try
{
	_connection = (java.net.HttpURLConnection)_url.openConnection();
}
catch(Exception e)
{
	errorMsg = null;
	errorMsg = e.getMessage();
	if (log.compareTo("yes") == 0)
	{
		_instance.doLog("ERROR: URL openConnection: " + errorMsg);
	}
	System.out.println(errorMsg);
	e.printStackTrace();
	throw e;
}

_connection.setRequestMethod("POST");
_connection.setDoOutput(true);
```

```
System.out.println("Sending input XML...");
try
{
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("Sending input through PrintWriter.");
        }
        PrintWriter outWriter = new PrintWriter(_connection.getOutputStream());
        outWriter.print(inXML);
        outWriter.flush();
        outWriter.close();
}
catch (Exception e)
{
        errorMsg = null;
        errorMsg = e.getMessage();
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("ERROR: PrintWriter: " + errorMsg);
        }
        System.out.println(errorMsg);
        e.printStackTrace();
        throw e;
}

System.out.println("Waiting for Response...");
int bufsize = 204800;
char[] buffer = new char[bufsize];
int rBytes;
rspFilename = getNewFilename("rsp", fileName);
FileWriter rspFile = new FileWriter(rspFilename);
try
{
        String tmpRead ="";
        BufferedReader inReader = new BufferedReader(new InputStreamReader(_connection.getInputStream()));
        String Response;
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("Reading server response.");
        }
        while ((rBytes = inReader.read(buffer, 0, bufsize)) >=0)
        {
                tmpRead = new String(buffer);
                rspFile.write(buffer, 0, rBytes);
        }
        rspFile.close();
```

```
                inReader.close();

                File inFile = new File(rspFilename);

                FileInputStream fileStream = new FileInputStream(inFile);
                byte[] readbuffer = new byte[(int)inFile.length()];
                int nBytes = fileStream.read(readbuffer);

                String tmp = new String(readbuffer);
                System.out.println(tmp);

                String result = "Server response received.";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog(result);
                }
                else
                {
                        System.out.println(result);
                }
        }
        catch (Exception e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: BufferedReader: " + errorMsg);
                }
                System.out.println(errorMsg);
                e.printStackTrace();
                throw e;
        }

        _connection.disconnect();
}

/****/
// Creates keystores and related objects.
public void setSecurity() throws Exception
{
        try
        {
                char clientPass[] = cPass.toCharArray();
                char trustPass[] = tPass.toCharArray();

                if (log.compareTo("yes") == 0)
```

```
                        {
                                _instance.doLog("Accessing keystores.");
                        }
                        clientStore = KeyStore.getInstance("JKS");
                        clientStore.load(new FileInputStream(clientStoreName), clientPass);

                        trustStore = KeyStore.getInstance("JKS");
                        trustStore.load(new FileInputStream(trustStoreName), trustPass);

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating key factory manager.");
                        }
                        KeyManagerFactory keyMgrFac = KeyManagerFactory.getInstance("SunX509", new
com.sun.net.ssl.internal.ssl.Provider());
                        keyMgrFac.init(clientStore, clientPass);
                        KeyManager[] keyMgr = keyMgrFac.getKeyManagers();

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating trust factory manager.");
                        }
                        TrustManagerFactory trustMgrFac = TrustManagerFactory.getInstance("SunX509",
                                                                                          new
com.sun.net.ssl.internal.ssl.Provider());
                        trustMgrFac.init(trustStore);
                        TrustManager trustMgr[] = trustMgrFac.getTrustManagers();

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating SSLContext instance.");
                        }
                        SSLContext ctx = SSLContext.getInstance("SSLv3", new com.sun.net.ssl.internal.ssl.Provider());

                        ctx.init(keyMgr, trustMgr, null);

                        com.sun.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(ctx.getSocketFactory());
                }
                catch (Exception e)
                {
                        errorMsg = null;
                        errorMsg = e.getMessage();
                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("ERROR: Set Security: " + errorMsg);
                        }
                        System.out.println(errorMsg);
```

```
                e.printStackTrace();
                throw e;
        }
}

/****/
// Signs input XML file.
// Can be turned off if input param "-sign" set to "no".
public String doSignXML(String XMLstring) throws Exception
{
        String signedName;
        try
        {
                Security.insertProviderAt (new JCRYPTO (), 1);
                XMLDSIGInit.init ();


                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Creating XMLResource.");
                }
                XMLResource xmlres = ResourceFactory.getXMLResource();
                Document doc = xmlres.parseXML(new FileInputStream(XMLstring),  false);

                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Getting keystore certificate.");
                }
                cert = (X509Certificate) clientStore.getCertificate(keyAlias);
                PrivateKey privKey =
                        (PrivateKey) clientStore.getKey(keyAlias, keyPass.toCharArray());

                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Signing XML.");
                }
                Signer signer = new Signer(doc, privKey, cert);

                doc = signer.sign( );
                PrivateKey prv = (PrivateKey) clientStore.getKey (keyAlias, keyPass.toCharArray ());
                Certificate[] chain = clientStore.getCertificateChain (keyAlias);

/* Create a new document for the signature. */
```

```
                        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance ();
                        builderFactory.setNamespaceAware (true);
                        DocumentBuilder builder = builderFactory.newDocumentBuilder ();
                        Document document = builder.parse (XMLstring);
                        Element temp = document.createElement ("Temp");
                        temp.appendChild (document.getDocumentElement());
                        Element toBeSigned = (Element) temp.getFirstChild();

/* Create an XML signer. */
                        XMLDSIGSigner signer = new XMLDSIGSigner (prv, chain);
                        signer.setIndentation (true);

/* Sign the element. */
                        signer.signElementEnveloping ("pp", toBeSigned, document, null);

                        signedName = getNewFilename("sign", inputXML);

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating new signed XML file.");
                        }
                        xmlres.publish(doc, new FileOutputStream(new File(signedName)));String file = "sign-enveloping-
hl.xml";
                        OutputStream out = new FileOutputStream (signedName);
                        DOMSerializer.serialize (document, null, out);
                        out.close ();
                }
                catch (Exception e)
                {
                        errorMsg = null;
                        errorMsg = e.getMessage();
                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("ERROR: Sign XML: " + errorMsg);
                        }
                        System.out.println(errorMsg);
                        e.printStackTrace();
                        throw e;
                }
                return signedName;
        }

        public boolean IsValidFile(String theFile)
        {
                File inFile = new File(theFile);
                validFilename = inFile.exists();
                return validFilename;
```

```
}

/****/
public static void main(String[] args) throws Exception
{
        theURL         = findValue("-url", args);
        inputXML       = findValue("-input", args);
        keyAlias       = findValue("-alias", args);
        keyPass        = findValue("-pass", args);
        cPass          = findValue("-keystorepass", args);                 // "sseddog"
        clientStoreName        = findValue("-keystore", args);                     // "httpskeystore"
        tPass          = findValue("-truststorepass", args);        // "sseddog"
        trustStoreName = findValue("-truststore", args);            // "mytruststore"
        signXML        = findValue("-sign", args);                         // default value = "yes"
        log                    = findValue("-log", args);                              // default value = "no"

        String signedInputXML = null;

        // The required params
        if (theURL == null || inputXML == null || keyAlias == null || keyPass == null
                               || clientStoreName == null || cPass == null || trustStoreName == null || tPass == null)
        {
                usage();
                return;
        }
        System.setProperty("javax.net.ssl.trustStore", trustStoreName);
        System.setProperty("javax.net.ssl.trustStorePassword", tPass);

        _instance = HTTPSTestClient.createInstance();

        validFilename = _instance.IsValidFile(inputXML);
        if (validFilename == false)
        {
                String noFile = "ERROR: Input file not found";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog(noFile);
                        return;
                }
                else
                {
                        System.out.println(noFile);
                        return;
                }
        }

        _instance.setSecurity();
```

```
            if (signXML.toUpperCase().compareTo("YES") == 0)
            {
                    signedInputXML = _instance.doSignXML(inputXML);
                    _instance.doTransaction(theURL, signedInputXML);
            }
            else
            {
                    _instance.doTransaction(theURL, inputXML);
            }

            if (log.compareTo("yes") == 0)
            {
                    FileWriter logFile = new FileWriter("logFile.txt");
                    logFile.write(logData);
                    logFile.close();
            }

    }

       /****/
       // Gets command line param value
       static String findValue(String toFind, String argv[])
    {
            String val;
            if (toFind.compareTo("-log") == 0)
            {
                    val = findValue(toFind, argv, "no");
            }
            else if (toFind.compareTo("-sign") == 0)
            {
                    val = findValue(toFind, argv, "yes");
            }
            else
            {
                    val = findValue(toFind, argv, null);
            }
        return val;
    }

       static String findValue(String toFind, String argv[], String defaultVal)
    {
            for (int i = 0; i < argv.length; ++i)
            {
                    if (argv[i].equals(toFind))
                    {
             i += 1;
```

```
                if (i < argv.length && (argv[i] != null) && (!argv[i].startsWith("-")))
                                {
                    return argv[i];
                }
                                else
                                {
                    break;
                }
                       }
                }
        return defaultVal;
    }
}
}
```

Attachment B

Sample Source Code

Migration from

Verisign's Trust Services Integration Kit (TSIK) Version 1.5

to

Apache XML Security 1.1

```
/*
 * HTTPSTestClient.java
 *
 */

import java.net.*;
import java.io.*;

import java.io.*;
import java.net.*;

//package doc.examples.dsig;


import java.util.*;
import java.security.*;
import java.security.cert.*;
import java.security.cert.Certificate;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;


import java.security.spec.*;
import java.security.interfaces.*;

import com.sun.net.ssl.*;
import com.sun.net.ssl.internal.www.protocol.https.*;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import com.verisign.resource.ResourceFactory;
import com.verisign.resource.XMLResource;
import com.verisign.xmlsig.Signer;
import com.verisign.xmlsig.Verifier;
import com.verisign.xmlsig.SigningKey;
import com.verisign.xmlsig.VerifyingKey;
import com.verisign.xpath.XPath;
import com.verisign.domutil.DOMCursor;
import com.verisign.domutil.DOMWriteCursor;
import org.w3c.dom.Node;

import org.apache.xml.security.signature.ObjectContainer;
import org.apache.xml.security.signature.XMLSignature;
```

```
import org.apache.xml.security.transforms.Transforms;
import org.apache.xml.security.utils.Constants;
import org.apache.xml.security.utils.XMLUtils;


public class HTTPSTestClient
{
        public static String theURL          = null;
        public static String inputXML = null;
        public static String keyAlias = null;
        public static String keyPass  = null;
        public static String cPass           = null;
        public static String clientStoreName = null;
        public static String tPass           = null;
        public static String trustStoreName  = null;
        public static String log             = null;              // optional - uses default value
        public static String signXML  = null;              // optional - uses default value


        public static String logData         = "";
        public String tmplogData                    = null;
        public static String rspFilename     = null;
        public static String errorMsg        = null;
        public static boolean validFilename;

        public KeyStore                      clientStore;
        public KeyStore                      trustStore;
        public X509Certificate cert;
        public static HTTPSTestClient _instance;
        private URL                          _url;

        private void HTTPSTestClient()
        {
                _url = null;
                // Next 2 lines required to get past BST firewall, if needed
                //System.setProperty("https.proxyHost", "proxy.bst.bls.com");
                //System.setProperty("https.proxyPort", "8080");
        }

        public static HTTPSTestClient createInstance()
        {
                HTTPSTestClient ret = new HTTPSTestClient();
                return ret;
        }

    /* Usage is as follows:
       ** -url        (URL)
```

```
** -input      (Input XML name)
** -alias      (key alias)
** -pass       (key password)
** -keystore           (keystore name)
** -keystorepass       (keystore password)
** -truststore         (truststore name)
** -truststorepass  (truststore password)
** -sign       (sign XML)                          // optional - default = yes
** -log        (log program progress) // optional - default = no
*/
public static void usage()
{
        System.out.println("Input parameter list incorrect...");
        System.out.println("Required USAGE is as follows:");
        System.out.println("   [-url]         your HTTPS URL");
        System.out.println("   [-input]       input XML (path)file");
        System.out.println("   [-alias]       key alias");
        System.out.println("   [-pass]        key password");
        System.out.println("   [-keystore]           keystore name");
        System.out.println("   [-keystorepass] keystore password");
        System.out.println("   [-truststore]  truststore name");
        System.out.println("   [-truststorepass]     truststore password");
}

/****/
// Called if input param "-log" set to "yes".
// Stores (as String) program progress / errors,
// used for screen putput and written to file at program termination.
public void doLog(String temp)
{
        System.out.println(temp);
        temp += "\n";
        logData += temp;
        temp = null;
}

/****/
// Loads XML input file
public static String loadFile(String fileName)
{
        File inFile = new File(fileName);
        inFile.canRead();

        String inXML = "";
        try
        {
                FileInputStream fileStream = new FileInputStream(inFile);
```

```
                byte[] buffer = new byte[(int)inFile.length()];
                int nBytes = fileStream.read(buffer);

                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Reading input file.");
                }
                inXML = new String(buffer);
        }
        catch (FileNotFoundException e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: FileInputStream: " + errorMsg);
                }
                else
                {
                        System.out.println("ERROR: FileInputStream: " + errorMsg);
                }
                return null;
        }
        catch (java.io.IOException e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: FileInputStream: " + errorMsg);
                }
                else
                {
                        System.out.println("ERROR: FileInputStream: " + errorMsg);
                }
                return null;
        }
        inXML.trim();
        return inXML;
}

/****/
// Creates new file name for signed and response files.
// Adds "Signed", "Rsp" respectively, to input file name.
public String getNewFilename(String type, String aFilename)
{
```

```
        String newName = null;
        int index = aFilename.lastIndexOf(".");
        newName = aFilename.substring(0, index);
        if (type.compareTo("rsp") == 0)                      // new response xml file name
        {
                newName = newName + "Rsp.xml";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Created new response file name: "+ newName);
                }
        }
        else                                                                  // new signed xml file name
        {
                newName = newName + "Signed.xml";
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Created new signed file name: "+ newName);
                }
        }
        return newName;
}

/***/
// Actual server communication here. Reads input file, establishes connection,
// sets connection params, writes response file
public void doTransaction (String urlName, String fileName) throws Exception
{
        String inXML = loadFile(fileName);
        java.net.HttpURLConnection _connection;
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("In doTransaction.");
        }
        try
        {
                _url = new URL(urlName);
        }
        catch(MalformedURLException ue)
        {
                errorMsg = ue.getMessage();
                System.out.println(errorMsg);
        }
        catch(Exception e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
```

```
        {
                _instance.doLog("ERROR: URL creation: " + errorMsg);
        }
        System.out.println(errorMsg);
        e.printStackTrace();
        throw e;
}

try
{
        _connection = (java.net.HttpURLConnection)_url.openConnection();
}
catch(Exception e)
{
        errorMsg = null;
        errorMsg = e.getMessage();
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("ERROR: URL openConnection: " + errorMsg);
        }
        System.out.println(errorMsg);
        e.printStackTrace();
        throw e;
}

_connection.setRequestMethod("POST");
_connection.setDoOutput(true);

System.out.println("Sending input XML...");
try
{
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("Sending input through PrintWriter.");
        }
        PrintWriter outWriter = new PrintWriter(_connection.getOutputStream());
        outWriter.print(inXML);
        outWriter.flush();
        outWriter.close();
}
catch (Exception e)
{
        errorMsg = null;
        errorMsg = e.getMessage();
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("ERROR: PrintWriter: " + errorMsg);
```

```
        }
        System.out.println(errorMsg);
        e.printStackTrace();
        throw e;
}

System.out.println("Waiting for Response...");
int bufsize = 204800;
char[] buffer = new char[bufsize];
int rBytes;
rspFilename = getNewFilename("rsp", fileName);
FileWriter rspFile = new FileWriter(rspFilename);
try
{
        String tmpRead ="";
        BufferedReader inReader = new BufferedReader(new InputStreamReader(_connection.getInputStream()));
        String Response;
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog("Reading server response.");
        }
        while ((rBytes = inReader.read(buffer, 0, bufsize)) >=0)
        {
                tmpRead = new String(buffer);
                rspFile.write(buffer, 0, rBytes);
        }
        rspFile.close();
        inReader.close();

        File inFile = new File(rspFilename);

        FileInputStream fileStream = new FileInputStream(inFile);
        byte[] readbuffer = new byte[(int)inFile.length()];
        int nBytes = fileStream.read(readbuffer);

        String tmp = new String(readbuffer);
        System.out.println(tmp);

        String result = "Server response received.";
        if (log.compareTo("yes") == 0)
        {
                _instance.doLog(result);
        }
        else
        {
                System.out.println(result);
        }
```

```
                }
                catch (Exception e)
                {
                        errorMsg = null;
                        errorMsg = e.getMessage();
                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("ERROR: BufferedReader: " + errorMsg);
                        }
                        System.out.println(errorMsg);
                        e.printStackTrace();
                        throw e;
                }

                _connection.disconnect();
        }

        /****/
        // Creates keystores and related objects.
        public void setSecurity() throws Exception
        {
                try
                {
                        char clientPass[] = cPass.toCharArray();
                        char trustPass[] = tPass.toCharArray();

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Accessing keystores.");
                        }
                        clientStore = KeyStore.getInstance("JKS");
                        clientStore.load(new FileInputStream(clientStoreName), clientPass);

                        trustStore = KeyStore.getInstance("JKS");
                        trustStore.load(new FileInputStream(trustStoreName), trustPass);

                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating key factory manager.");
                        }
                        KeyManagerFactory keyMgrFac = KeyManagerFactory.getInstance("SunX509", new
com.sun.net.ssl.internal.ssl.Provider());
                        keyMgrFac.init(clientStore, clientPass);
                        KeyManager[] keyMgr = keyMgrFac.getKeyManagers();

                        if (log.compareTo("yes") == 0)
                        {
```

```
                        _instance.doLog("Creating trust factory manager.");
                }
                TrustManagerFactory trustMgrFac = TrustManagerFactory.getInstance("SunX509",
                                                                                   new
com.sun.net.ssl.internal.ssl.Provider());
                trustMgrFac.init(trustStore);
                TrustManager trustMgr[] = trustMgrFac.getTrustManagers();

                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Creating SSLContext instance.");
                }
                SSLContext ctx = SSLContext.getInstance("SSLv3", new com.sun.net.ssl.internal.ssl.Provider());

                ctx.init(keyMgr, trustMgr, null);

                com.sun.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(ctx.getSocketFactory());
        }
        catch (Exception e)
        {
                errorMsg = null;
                errorMsg = e.getMessage();
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("ERROR: Set Security: " + errorMsg);
                }
                System.out.println(errorMsg);
                e.printStackTrace();
                throw e;
        }
}

/****/
// Signs input XML file.
// Can be turned off if input param "-sign" set to "no".
public String doSignXML(String XMLstring) throws Exception
{
        String signedName;
        try
        {
                if (log.compareTo("yes") == 0)
                {
                        _instance.doLog("Creating XMLResource.");
                }
                XMLResource xmlres = ResourceFactory.getXMLResource();
                Document doc = xmlres.parseXML(new FileInputStream(XMLstring),  false);
```

```
                    if (log.compareTo("yes") == 0)
                    {
                        _instance.doLog("Getting keystore certificate.");
                    }
                    PrivateKey prv = (PrivateKey) clientStore.getKey (keyAlias, keyPass.toCharArray ());
                    X509Certificate cert = (X509Certificate) clientStore.getCertificate(keyAlias);
                    PrivateKey privKey =
                        (PrivateKey) clientStore.getKey(keyAlias, keyPass.toCharArray());

                    if (log.compareTo("yes") == 0)
                    {
                        _instance.doLog("Signing XML.");
                    }
                    Signer signer = new Signer(doc, privKey, cert);

                    doc = signer.sign( );
//                  Certificate[] chain = clientStore.getCertificateChain (keyAlias);
                    signedName = getNewFilename("sign", inputXML);
                    File signatureFile = new File(signedName);

/* Create a new document for the signature. */
                    DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance ();
                    builderFactory.setNamespaceAware (true);
                    DocumentBuilder builder = builderFactory.newDocumentBuilder ();
                    Document document = builder.newDocument();
                    String BaseURI = signatureFile.toURL().toString();
                    XMLSignature sig = new XMLSignature(document, BaseURI, XMLSignature.ALGO_ID_SIGNATURE_RSA);
                    document.appendChild(sig.getElement());

                    Document d = builder.parse(XMLstring);
                    Node node = document.importNode(d.getDocumentElement(), true);
                    DocumentFragment docfrag = document.createDocumentFragment();
                    docfrag.appendChild(document.importNode(d.getDocumentElement(), true));

                    ObjectContainer obj = new ObjectContainer(document);
                    obj.appendChild(docfrag);
                    String Id = "pp";
                    obj.setId(Id);
                    sig.appendObject(obj);

                    Transforms transforms = new Transforms(document);
                    transforms.addTransform(Transforms.TRANSFORM_C14N_OMIT_COMMENTS);
```

```
                        sig.addDocument("#" + Id, transforms, Constants.ALGO_ID_DIGEST_SHA1);

                        sig.addKeyInfo(cert);
                        sig.addKeyInfo(cert.getPublicKey());
                        sig.sign(prv);


                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("Creating new signed XML file.");
                        }
                        xmlres.publish(doc, new FileOutputStream(new File(signedName)));//              String file =
"sign-enveloping-hl.xml";
                        FileOutputStream out = new FileOutputStream (signedName);
                        XMLUtils.outputDOMc14nWithComments(document, out);
                        out.close();
                }
                catch (Exception e)
                {
                        errorMsg = null;
                        errorMsg = e.getMessage();
                        if (log.compareTo("yes") == 0)
                        {
                                _instance.doLog("ERROR: Sign XML: " + errorMsg);
                        }
                        System.out.println(errorMsg);
                        e.printStackTrace();
                        throw e;
                }
                return signedName;
        }

        public boolean IsValidFile(String theFile)
        {
                File inFile = new File(theFile);
                validFilename = inFile.exists();
                return validFilename;
        }

        /****/
        public static void main(String[] args) throws Exception
        {
                theURL          = findValue("-url", args);
                inputXML        = findValue("-input", args);
                keyAlias        = findValue("-alias", args);
                keyPass         = findValue("-pass", args);
                cPass           = findValue("-keystorepass", args);                 // "sseddog"
```

```
clientStoreName       = findValue("-keystore", args);                    // "httpskeystore"
tPass          = findValue("-truststorepass", args);        // "sseddog"
trustStoreName = findValue("-truststore", args);            // "mytruststore"
signXML        = findValue("-sign", args);                              // default value = "yes"
log                   = findValue("-log", args);                                 // default value = "no"

String signedInputXML = null;

// The required params
if (theURL == null || inputXML == null || keyAlias == null || keyPass == null
                      || clientStoreName == null || cPass == null || trustStoreName == null || tPass == null)
{
       usage();
       return;
}
System.setProperty("javax.net.ssl.trustStore", trustStoreName);
System.setProperty("javax.net.ssl.trustStorePassword", tPass);

_instance = HTTPSTestClient.createInstance();

validFilename = _instance.IsValidFile(inputXML);
if (validFilename == false)
{
       String noFile = "ERROR: Input file not found";
       if (log.compareTo("yes") == 0)
       {
              _instance.doLog(noFile);
              return;
       }
       else
       {
              System.out.println(noFile);
              return;
       }
}

_instance.setSecurity();

if (signXML.toUpperCase().compareTo("YES") == 0)
{
       signedInputXML = _instance.doSignXML(inputXML);
       _instance.doTransaction(theURL, signedInputXML);
}
else
{
       _instance.doTransaction(theURL, inputXML);
}
```

```
            if (log.compareTo("yes") == 0)
            {
                    FileWriter logFile = new FileWriter("logFile.txt");
                    logFile.write(logData);
                    logFile.close();
            }
}

    /****/
    // Gets command line param value
    static String findValue(String toFind, String argv[])
{
            String val;
            if (toFind.compareTo("-log") == 0)
            {
                    val = findValue(toFind, argv, "no");
            }
            else if (toFind.compareTo("-sign") == 0)
            {
                    val = findValue(toFind, argv, "yes");
            }
            else
            {
                    val = findValue(toFind, argv, null);
            }
     return val;
}

    static String findValue(String toFind, String argv[], String defaultVal)
{
            for (int i = 0; i < argv.length; ++i)
            {
                    if (argv[i].equals(toFind))
                    {
             i += 1;
             if (i < argv.length && (argv[i] != null) && (!argv[i].startsWith("-")))
                            {
                 return argv[i];
             }
                            else
                            {
                 break;
             }
                    }
            }
     return defaultVal;
```

```
     }

    static {
       org.apache.xml.security.Init.init();
    }

}
```