Caché Web Services Cookbook

Version XXXX ***DRAFT***
15 December 2009

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Caché Web Services Cookbook
Caché Version XXXX ***DRAFT*** 15 December 2009

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.

InterSystems CACHÉ® and InterSystems ENSEMBLE®

Caché WEBLINK, Distributed Cache Protocol, M/SQL, N/NET, and M/PACT are registered trademarks of InterSystems Corporation.

INTERSYSTEMS InterSystems HEALTHSHARE™ InterSystems DEEPSEE™ and InterSystems TRAKCARE™

InterSystems Jalapeño Technology, Enterprise Cache Protocol, ECP, and InterSystems Zen are trademarks of InterSystems Corporation.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel: +1 617 621-0700
Fax: +1 617 374-9391
Email: support@InterSystems.com

Table of Contents

1

Introduction

This book is meant to help programmers consume Caché Web services, particularly with Web clients that do not use Caché.

1.1 Scenarios Included in This Book

This book includes the following scenarios:

- Generating clients in .NET 3.5 and Java Metro.

  The examples in the rest of the book use these clients as the starting point.
- Using password authentication.
- Configuring SSL encryption with one-way authentication.
- Configuring SSL encryption with mutual authentication.
- Applying XML encryption to the SOAP messages themselves.
- Adding and using custom SOAP headers.

Because many of these tasks require certificates, this book includes an appendix on creating sample certificates and another appendix on installing them.

Another appendix includes tips on installing and configuring IIS (Internet Information Services).

1.2 Setting Up the Sample Web Service

This book uses a sample Caché Web service. To set up this sample:

1. Optionally create a new namespace in the System Management Portal to use as a test area. (This book uses the namespace `COOKBOOK`.)

   For this namespace, disable password authentication and enable unauthenticated access. We will change this setting later.
2. In Studio, switch to the new namespace.

3. Use **Tools > Import Local** to load and compile the sample file ModelLibrary.xml.

   After you do this, you will have a set of classes in the ModelLibrary package.

4. Compile these classes.

5. In the Terminal, execute the **Setup()** class method of the class ModelLibrary.Setup:

```
USER>zn "cookbook"
COOKBOOK>d ##class(ModelLibrary.Setup).Setup()
```

   This method creates some data for use by the Web service.

1.3 Introduction to the Sample Web Service

The sample Web service is ModelLibrary.WSCookbook, whose endpoint is available at the following URL:

```
http://localhost:57772/csp/cookbook/ModelLibrary.WSCookbook.CLS
```

Where *57772* is the port number of the Caché superserver and *cookbook* is the namespace into which you installed the sample. (This URL will be different when we configure the Web service to support SSL; see the chapter "Setting Up SSL Encryption.")

The service name of the Web service is `WSCookbook`.

The Web service uses the XML namespace `http://www.intersystems.com/ws-cookbook`.

1.3.1 Available Web Methods

The sample Web service enables users to work with a database of book titles and their reviews. It includes the following methods:

AddBook()

```
method AddBook(isbn As %Integer, title As %String, authorFname As %String, authorLname As %String)
as %Boolean
```

Adds a book to the library and returns true if successful or false if not successful.

AddReview()

```
method AddReview(isbn As %Integer, authorFname As %String, authorLname As %String,
    email As %String, input As %String) as %Boolean
```

Adds a review to a book, given its ISBN. Returns true if successful or false if not successful.

DeleteBook()

```
method DeleteBook(isbn As %Integer) as %Boolean
```

Given an ISBN, deletes a book from the database. Returns true if successful or false if not successful.

GetBook()

```
method GetBook(isbn As %Integer) as ModelLibrary.Book
```

Given an ISBN, returns an instance of ModelLibrary.Book that contains the information for that book.

GetBookCount()

```
method GetBookCount() as %Integer
```

Returns the number of books in the database.

LogOut()

```
LogOut() as %Boolean
```

Enables the client to end the CSP session that is used when the SOAP sessions are enabled (each CSP session consumes one license slot). Note that SOAP sessions are not enabled by default in this Web service.

Also notice that CSP sessions normally time out after a brief interval, so that a client is not required to log out.

This method is used in the chapter "Setting Up Password Authentication."

1.3.2 Accessing the WSDL

The WSDL for this Web service is available at the following URL:

```
http://localhost:57772/csp/cookbook/ModelLibrary.WSCookbook.CLS?WSDL=1
```

Where *57772* is the port number of the Caché superserver and *cookbook* is the namespace into which you installed the sample.

The first examples in this book assume that the Web service supports unauthenticated access. In a later part of the book, you turn on password authentication for the CSP application for this namespace. When you do so, if you need to access the WSDL again, you must include a username and password in the URL:

```
http://localhost:57772/csp/user/ModelLibrary.WebService.CLS?WSDL=1&CacheUsername=username&CachePassword=password
```

Where *username* and *password* are a valid Caché username and corresponding password, respectively.

1.4 Useful Software

To perform the tasks in this tutorial, you need the following tools:

1.4.1 For .NET (C#)

- Visual Studio 2008: http://msdn.microsoft.com/en-us/vstudio/default.aspx

1.4.2 For Java Metro

- Metro JAX-WS Dev Kit (version 2.0 or higher): https://metro.dev.java.net/

  The installer is a self-extracting JAR file. To set up:

  1. Place the JAR file wherever you wish and run it.

2. Define the *METRO_HOME* environment variable. For example: `C:\Program Files\Sun\metro`
3. Add the Metro bin directory to your *PATH* environment variable. For example: `C:\Program Files\Sun\metro\bin`

This utility provides a WSDL reader that generates proxy class files from an input WSDL.

- NetBeans (integrated development environment): http://www.netbeans.org/
- Eclipse (integrated development environment): http://www.eclipse.org/
- GlassFish >> Metro: https://metro.dev.java.net/
- Portecle: http://portecle.sourceforge.net/
- Java SE Development Kit (JDK): http://java.sun.com/javase/downloads/index.jsp
- Java Runtime Environment: http://www.java.com/en/download/manual.jsp

1.4.3 For Security and Packet Tracing

- TcpTrace (free packet trace utility): http://www.pocketsoap.com/tcptrace/
- WireShark (free detailed packet trace utility): http://www.wireshark.org/
- OpenSSL: http://www.openssl.org/ (source files) or http://gnuwin32.sourceforge.net/packages/openssl.htm (installer)

  Be sure to add the OpenSLL bin directory to your *PATH* environment variable. For example: `C:\Program Files\GnuWin32\bin`

  You'll also need an openssl.cnf file, an OpenSSL configuration file that is not included in the install. Download one from http://tud.at/programm/openssl.cnf or other locations and save it to your disk.

2

Creating a Basic Client Using .NET 3.5

To generate a Web Service client in .NET, you use the following procedure:

1. Starting with the WSDL for the Web service, generate a set of proxy classes that communicate with that Web service.
2. Compile the proxy classes as a DLL.
3. Create a small client in C# that uses this DLL.

2.1 Prerequisites

- Determine the URL for the Web service, as appropriate for your environment. For the sample, see "Accessing the WSDL," earlier in this book.
- Ensure that you have Visual Studio installed. See "Useful Software," earlier in this book.
- Ensure that the CSP application for the relevant Caché namespace uses only unauthenticated access.

2.2 Generating and Compiling the Client Proxy Classes

1. Open the Visual Studio command prompt.

   For example, click **Start > All Programs > Visual Studio > Tools > Visual Studio Command Prompt**.
2. Navigate to the directory in which you would like to work. For example: `C:\WS-cookbook\`
3. Type in the following code to parse the WSDL document:

   ```
   wsdl /l:CS /protocol:SOAP http://localhost:57772/csp/cookbook/ModelLibrary.WSCookbook.cls?WSDL
   ```

   Where *57772* is the port number of the Caché superserver and *cookbook* is the namespace into which you installed the sample.

   The flags are as follows:

   - `/l:CS` — Tells the WSDL parser to generate the proxy source code.

- `/protocol:SOAP` — Indicates that we are using SOAP to access the WSDL.
- `http://...` — Provides the address of the WSDL document. This can be a Web address or local file.

This step generates a proxy class file called `WSCookbook.cs`, which is based on the Web service name. This file includes a partial class called `WSCookbook`, which is the proxy client. This class includes one method for each Web method of the original Web service. For example, it includes the `GetBook()` method, which invokes the Web method of the same name.

This file also includes partial classes for every type. So, for example, it includes the `Book` class and a `Review` class.

Note: After you generate the proxy classes, you do not usually edit them. Instead you create other classes that act as a wrapper and that provide client-side error handling. However, it is worthwhile to review the types given in the WSDL, examine the method signatures in the generated classes, and adjust any types as appropriate for .NET. In this example, no changes are needed.

4. Compile the proxy class file into a DLL as follows:

```
csc /t:library /r:System.Web.Services.dll /r:System.Xml.dll WSCookbook.cs
```

The `/r:...` flags specify two system reference libraries that we need: Web.Services.dll and Xml.dll

Now you should have two files in this directory: WSCookbook.cs and WSCookbook.dll.

2.3 Creating and Testing a Web Client

For the purpose of demonstration, the Web client will be hardcoded to invoke the **GetBook()** method. An actual client would of course be more generic and would access more of the Web methods.

1. Start Visual Studio.
2. Create a new project.

   For simplicity, use the **Console Application** template.
3. For this project, specify a working directory. In this example, we use `c:/WS-cookbook`.
4. Add the following references to the project:
   - WSCookbook.dll, the DLL that we just created.
   - System.Web.Services.dll, which is in the **.NET** group of DLLs.

   To add references, right-click **References** in the right area and then click **Add Reference**. Or click **Project > Add Reference**.
5. Edit the main program file, `Program.cs` as follows:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WS_cookbook
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Retrieving title of book 123401:");

            WSCookbook service = new WSCookbook();
            Book test = new Book();

            test = service.GetBook(123401, true);
            Console.WriteLine(test.Title);

        }
    }
}
```

6. Build the file. Visual Studio then generates a file named *projectname*.exe, where *projectname* is the name you gave to the project. It places this file as well as a copy of the WSCookbook.dll into the following subdirectory:

   ```
   working-dir/projectname/projectname/bin/Debug
   ```

   Where *working-dir* is the working directory you specified earlier.

7. In a DOS shell, navigate to this directory and run the file *projectname*.exe:

   ```
   c:\WS-cookbook\DotNetClient\DotNetClient\bin\Debug>DotNetClient
   Retrieving title of book 123401:
   Book 2
   ```

   Or run the project from the **Debug** menu.

3

Creating a Basic Client Using Java Metro

To generate a Web Service client in Java Metro, you use the following procedure:

1. Starting with the WSDL for the Web service, generate a set of client proxy classes that communicate with that Web service.
2. Create a small client in Java Metro that uses these proxy classes.

3.1 Prerequisites

- Determine the URL for the Web service, as appropriate for your environment. For the sample, see "Accessing the WSDL," earlier in this book.
- Ensure that you have the software needed to create classes in Java Metro (especially the Metro JAX-WS Dev Kit). See "Useful Software," earlier in this book.
- Ensure that the CSP application for the relevant Caché namespace uses only unauthenticated access.

Tip: Every time you execute a Web client based on Java Metro, that client must have access to the WSDL. This means that even if you save the WSDL to a file and generate the client proxy from that file, the WSDL must still be accessible at all times to the client.

3.2 Generating the Client Proxy Classes

1. Open a command prompt.
2. Type the following command:

   ```
   wsimport http://localhost:57772/csp/cookbook/ModelLibrary.WSCookbook.cls?WSDL -keep
   ```

   Where *57772* is the port number of the Caché superserver and *cookbook* is the namespace into which you installed the sample.

   If the Web service is in a CSP application where password authentication is required, append a string like the following at the end of the URL (using the username `_SYSTEM` and password `SYS` as an example):

```
&CacheUserName=_SYSTEM&CachePassword=SYS
```

This creates the directory com\intersystems\ws_cookbook\, which is based on the XML namespace of the Web service. This directory contains the following Java class files:

- Two classes for each Web method, for example, `AddBook` and `AddBookResponse`
- One class for each type in the WSDL, for example, `Author` and `Book`
- Two proxy classes that use the service name of the Web service: `WSCookbook` and `WSCookbookSoap`
- One information class: `package-info`

The package for all of the classes is *com.intersystems.ws_cookbook*.

3. Edit the file WSCookbook.java to remove or comment out the following constructors, which we do not need:

```
public WSCookbook(WebServiceFeature... features) {
        super(WSCOOKBOOK_WSDL_LOCATION, WSCOOKBOOK_QNAME, features);
    }

public WSCookbook(URL wsdlLocation, WebServiceFeature... features) {
        super(wsdlLocation, WSCOOKBOOK_QNAME, features);
    }

public WSCookbook(URL wsdlLocation, QName serviceName, WebServiceFeature... features) {
        super(wsdlLocation, serviceName, features);
    }
```

4. Delete the .class files, which we do not need.

Note: After you generate the proxy classes, you do not usually edit them. Instead you create other classes that act as a wrapper and that provide client-side error handling. However, it is worthwhile to review the types given in the WSDL, examine the method signatures in the generated classes, and adjust any types as appropriate for Java. In this example, no changes are needed.

3.3 Creating and Testing a Web Client

For the purpose of demonstration, the Web client will be hardcoded to invoke the **GetBook()** method. An actual client would of course be more generic and would access more of the Web methods.

To create a Web client that uses the generated proxy classes, do the following:

1. Open NetBeans or Eclipse, whichever you prefer to use.
2. Create a new project. For the project, specify a working directory, for example: `C:\WS-cookbook\JavaClient`.
3. Import the .java files into the project, so that the com directory and all its contents are within the src directory for the project.
4. In NetBeans or Eclipse, add all the JAR library files in the Metro/lib directory to your build path. These JAR files need to be available at compile time.
5. Add the .java files that you generated in the previous section as source files.
6. Create a new class (for example, `Client`). For the package, use the same package to which the proxy classes belong (see the previous section).

7. Edit this class as follows:

```
package com.intersystems.ws_cookbook;

public class Client {

    public static void main(String[] args) {

        WSCookbook service = new WSCookbook();
        WSCookbookSoap serviceSoap = service.getWSCookbookSoap();
        Book test = new Book();

        test = serviceSoap.getBook((long)123401);

        System.out.println("Retrieving title of book 123401:");
        System.out.println(test.getTitle());

    }

}
```

8. Test your code. In Eclipse, you can execute it from the Console.

4

Setting Up Password Authentication

This chapter describes how to set up password authentication for a Caché Web service and how to modify Web clients so that they can still work with this Web service. It discusses the following topics:

- Introduction
- How to configure the Caché Web service
- Using HTTP authentication with Java Metro

4.1 Introduction

You can configure a Caché Web service to require password authentication. Caché Web services follow WS-Security, which allows for user authentication by means of the <UsernameToken> element.

Before enabling authentication, consider the following points:

- Anonymous access of your Web service will not be allowed.
- Each login will use one Caché user license.
- The clients send the username and password in clear text, so it is necessary to also use SSL, which is discussed in a later chapter.

4.1.1 About the Client Examples in This Chapter

The examples in this chapter use the following approach:

1. Configure the Caché Web service to require password authentication.
2. Also configure the Caché Web service to use Caché SOAP session support.

   When SOAP sessions are enabled for a Web service, Caché maintains a CSP session when the Web service is used. The Web service stores the SessionID in a cookie and passes that to the client in the CSPCHD SOAP header. In order to maintain the session, the Web client must then always pass the CPSCHD SOAP header back to the server with every request.
3. Within the client, do the following:

a. Send a custom HTTP request that contains the <UsernameToken> element.

b. Use this request to retrieve the CSPCHD value.

c. Save that value in the client for use in outbound SOAP calls.

d. Call the proxy Web method itself.

e. Optionally log out (ending the CSP session).

   Or simply let the CSP session time out as it normally does.

   A manual logout is useful in cases where the Web method will be invoked frequently in a short span of time, and you do not want to hold licenses longer than necessary.

4.1.2 A Look at a Request Message

To understand how to send login data and interpret incoming SOAP messages, we first look at a valid SOAP request that includes the <UsernameToken> element:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <Security
      xmlns="http://docs.oasis-open.org...wssecurity-secext-1.0.xsd">
      <UsernameToken>
        <Username>test</Username>
        <Password Type="http://docs.oasis...PasswordText">test</Password>
      </UsernameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <GetBook xmlns="http://www.intersystems.com/ws-cookbook">
      <isbn>00123400</isbn>
    </GetBook>
  </soap:Body>
</soap:Envelope>
```

Note: The namespace for the <Security> element as well as the type for the <Password> element have been shortened for readability purposes.

4.1.3 A Look at the HTTP Header

A SOAP request also has an HTTP header, like this:

```
POST /csp/cookbook/ModelLibrary.WSCookbook.cls HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 2.0.50727.3082)
Content-Type: text/xml; charset=utf-8
Soapaction: http://localhost:57772/csp/cookbook/ModelLibrary.WSCookbook.GetBook
Host: localhost:57772
Content-Length: 727
Connection: Keep-Alive
```

In our custom HTTP request, we will have to mimic the details shown here.

4.2 Configuring the Web Service

4.2.1 Requiring Password Authentication for the Web Service

In any given Caché namespace, the Web services are controlled by the CSP application that corresponds to that namespace. You enable or disable password authentication for the CSP application itself as follows:

1. In the System Management Portal, click **Security Management**.
2. Under **Application Definitions**, click **CSP Applications**.
3. Click **Edit** in the row for namespace, for example, `/csp/cookbook`.
4. Clear **Unauthenticated**.
5. Select **Password**.
6. Click **Save**.

Tip: Any client application you may have written up to this point may not work. Refer to the examples in this chapter for more information.

4.2.2 Enabling SOAP Sessions

To enable the Web service to use SOAP sessions, add the following parameter to the Web service class:

```
Parameter SOAPSESSION = 1;
```

Then recompile the Web service.

4.3 Performing Basic HTTP Authentication with the Java Client

Java Metro supports the use of basic HTTP authentication with its generated proxy classes. You can use a Java Authenticator to achieve this.

```
final String login ="...";
final String password ="...";

System.setProperty("javax.net.ssl.keyStore", "c:\\clicert.jks");
System.setProperty("javax.net.ssl.keyStorePassword", "client");

Authenticator.setDefault(new Authenticator() {
    protected PasswordAuthentication getPasswordAuthentication(){
        return new PasswordAuthentication(login,
              password.toCharArray());
    }
});
```

For information on configuring Caché to accept basic HTTP authentication, see the *CSP Gateway Configuration Guide*.

5

Encryption and Certificates

This chapter provides background information for the following chapters. It discusses the following topics:

- An overview of encryption options for Web services
- Encryption and keys
- Keys and certificates

5.1 Encryption Options for Web Services

For Web services, messages are typically encrypted in one or both of the following ways:

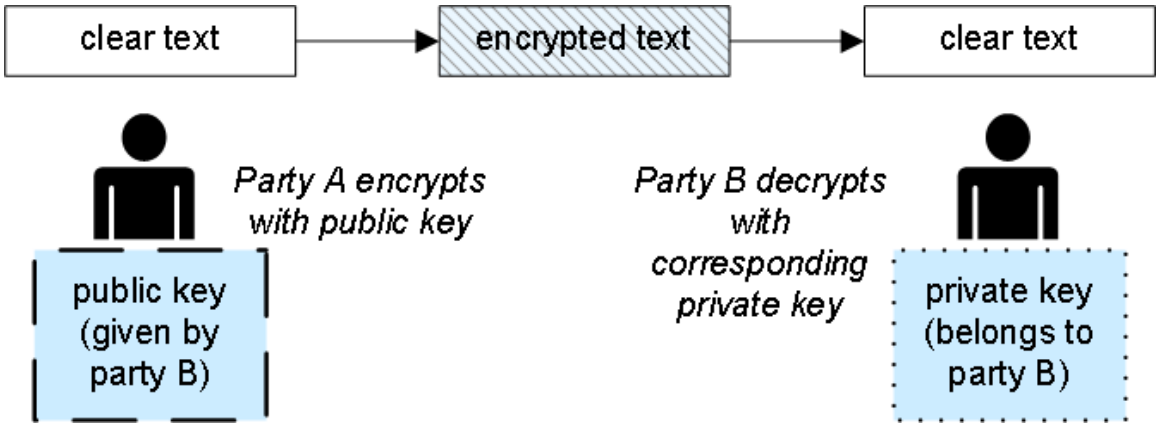
- In the *transport layer*. In this case, you encrypt the communications channel with SSL/TLS.

  With this type of encryption, you can have either one-way authentication or mutual authentication.
- In the *message layer*. In this case, you use XML Encryption, which is a WS-Security feature.

5.2 Encryption and Keys

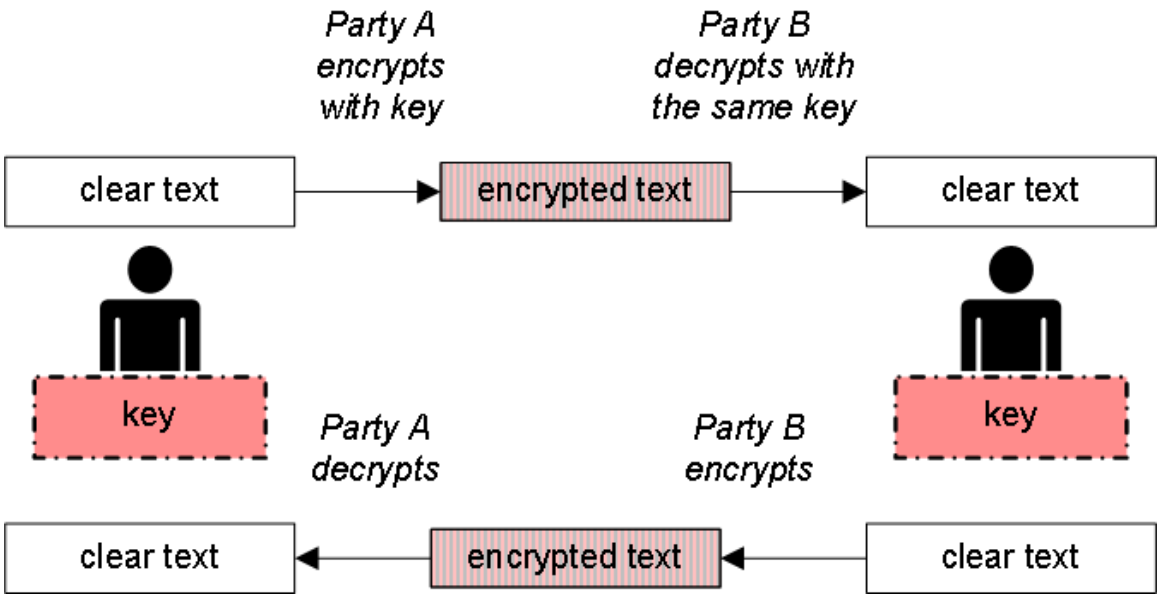
In all cases, encryption is implemented using keys.

In *asymmetric-key encryption*, two keys are used: the *private key* and the corresponding *public key*. When something is encrypted with the private key, only the public key can decrypt it. When something is encrypted with the public key, only the private key can decrypt it. To enable others to send encrypted text to you, you create a pair of keys, you keep the private key, and you distribute the public key. Then anyone who has a copy of the public key can send encrypted text to you, and nobody else can decrypt that text.



Notice that in this scheme, party B cannot communicate securely. This party can encrypt messages, but anyone can decrypt them, because the public key has been distributed publicly.

In *symmetric-key encryption*, one key is used, and it can both encrypt text and decrypt the same text.



In this scheme, both parties can send encrypted messages to each other. The difficulty is establishing a common key that is known only to the two parties.

The SSL mechanism uses both these encryption schemes, in sequence, as the next chapter describes.

5.3 Keys and Certificates

Public keys are stored within *certificates*, which also identify the owner of the public key. An X.509 certificate holds a public key and identifying information such as the `(O) Organization Name` or `(CN) Common Name` or many other items. A certificate is signed either by a certificate authority (CA), by itself, or by another certificate, which in turn is signed by either a certificate authority (CA) or by another certificate, and so on.

For simplicity, this book assumes that signed certificates are signed by a CA. (For information on certificate chains, see the *Caché Security Administration Guide*.)

Certificates for CAs are issued by a trusted issuer, such as VeriSign or others.

5.3.1 Types of Certificates

For Web services, there are three certificates to consider:

- CA (Certificate Authority) certificates.
- The server certificate (that is, the certificate that holds the public key of the server and that identifies the server).
- Client certificates (that is, a certificate that holds the public key of a client and identifies that client). These are less common but can be used for enhanced security.

6

Setting Up SSL Encryption

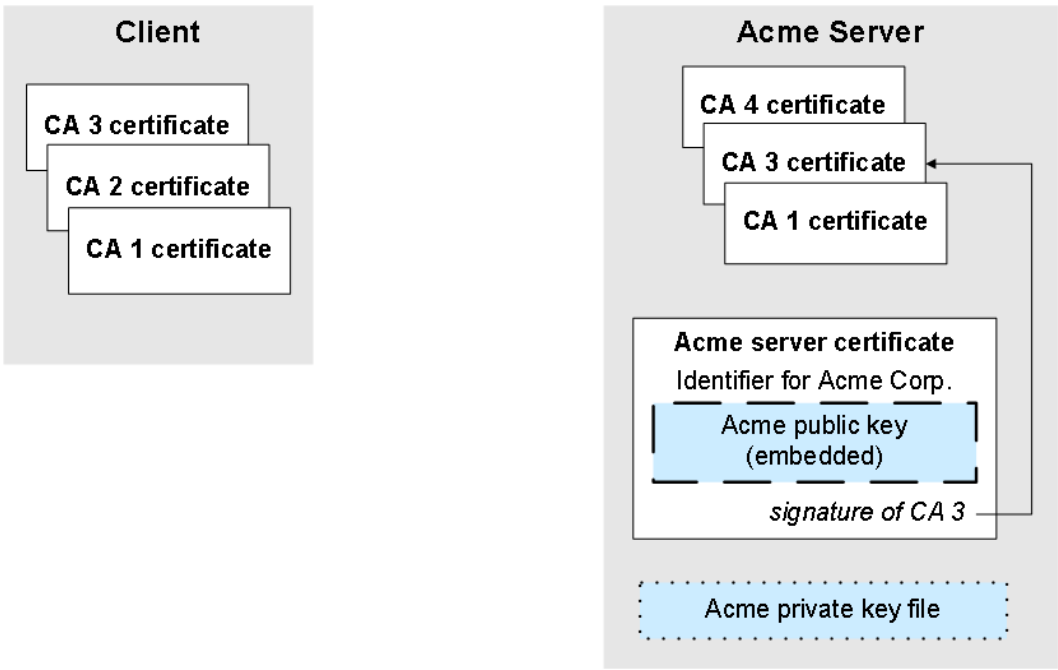
This chapter describes how to set up SSL encryption between a Web service and its clients. It discusses the following topics:

- An overview of SSL encryption with server authentication (one-way authentication)
- How to set up SSL encryption with server authentication
- An overview of SSL encryption with mutual authentication
- How to set up SSL encryption with mutual authentication

6.1 Overview of SSL Encryption with Server Authentication

To help you remember where to place certificates, this section provides an overview of the SSL handshake — the steps that occur when an SSL connection is initialized. For simplicity, this section discusses SSL with server authentication; a later section discusses the additional details for mutual authentication.

First, let us examine a possible scenario, shown in the following figure:

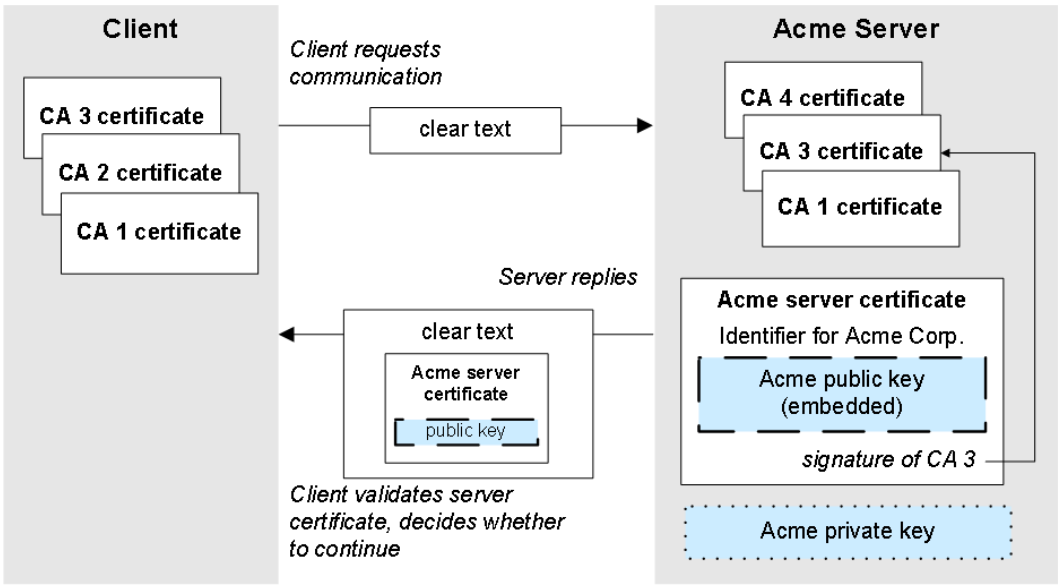


In this example, the client machine has several certificates installed, from three different certificate authorities.

The server machine also has several certificates installed, from a different set of certificate authorities. The server machine also has its own server certificate, which is signed by CA 3 (just as an example). By definition, this certificate includes a copy of the server's public key. The machine also has the corresponding private key, in a location where it cannot be accessed by outsiders.

In the first phase, the client sends a message to the server. This message is in clear text and includes details about the cipher suites available to the client.

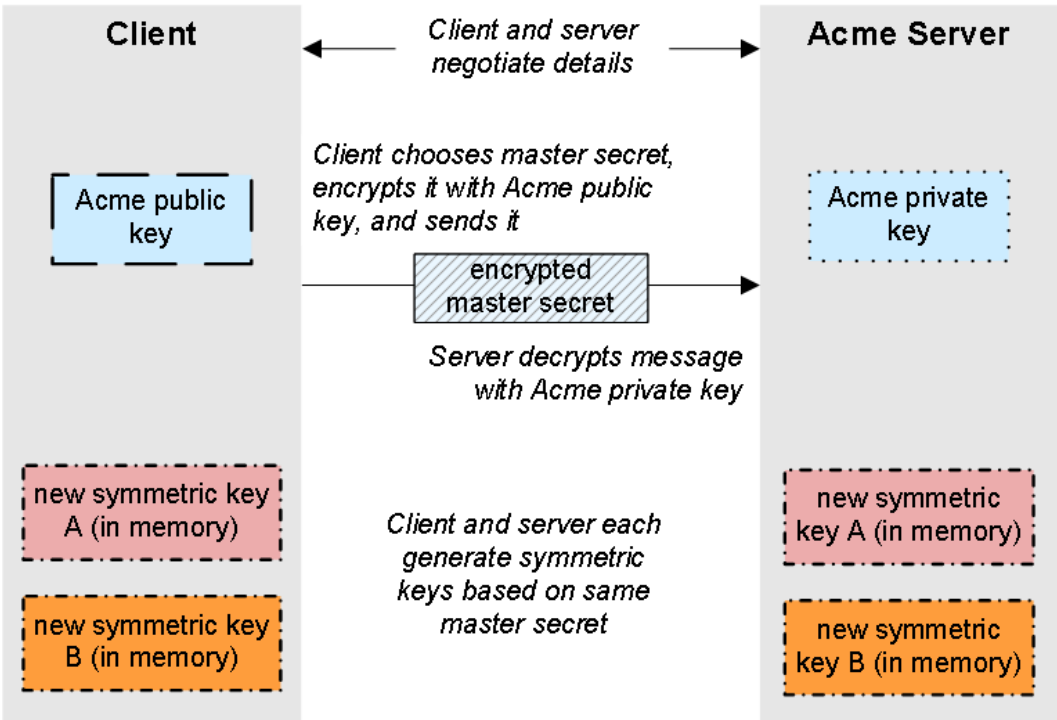
The server responds with another message in clear text, which includes details about the cipher suites to use. This message includes a copy of the server certificate; the server always sends its certificate to anyone who requests communication.



At some time that depends on implementation, the client tries to validate the server certificate. The client sees that the server certificate is signed by CA 3. The client then checks its own stored certificates to see if it recognizes this certificate authority, and to see if the signature, all the identifying details, and other details are correct. The client also checks that the certificate has not expired.

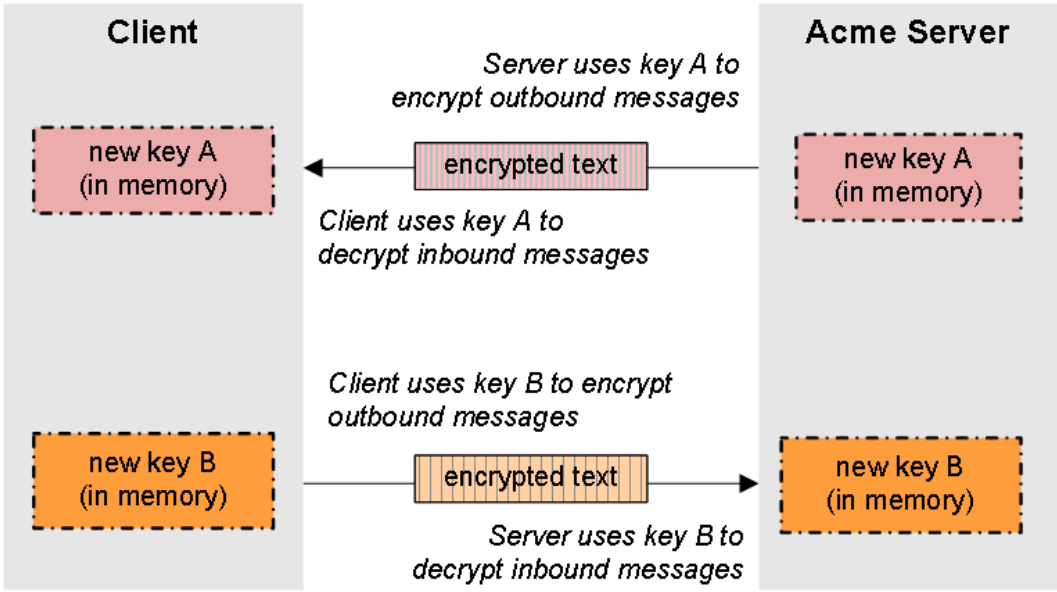
In order for both parties to send encrypted messages, both parties must have the same symmetric keys. The next phase is to choose symmetric keys that both the client and server can use.

First the client and server negotiate the details of how they will each generate these symmetric keys, based on the cipher suites available to both parties. The client then chooses a random master secret, encrypts it with the server's public key, and then sends that to the server.



Now that both parties have the same master secret, both parties use that to generate symmetric keys and hold them in memory. Because the parties have agreed on how to generate these keys, both parties generate the same keys.

The server uses one of the new symmetric keys (call it key A) to encrypt its next message and all later outbound messages.



Because the new key is a symmetric key, the client can decrypt the message.

The client uses the other symmetric key (call it key B) to encrypt its next message and all its later outbound messages, and the server can decrypt them.

6.2 Setting Up SSL with One-way Authentication

6.2.1 Prerequisites

Obtain the following certificates:

- A CA certificate
- A server certificate signed by that CA

For information on generating test certificates, see "Generating Sample Certificates."

6.2.2 Configuring the Server

On the server:

1. Install the CA certificate where needed by the operating system.

   For Windows, see "Installing Certificates for Use by Windows," in the second appendix.

2. Install a Web server that can use SSL. You have the following options:

   - Apache. The Apache that is installed with Caché cannot accept SSL connections. Therefore, you have to install another instance of Apache.

     This document does not contain further details on Apache.

   - (Windows only) IIS (Internet Information Services), used with the Caché Web Server Gateway.

For information on installing IIS and reconfiguring Caché to use it, see the appendix "IIS Installation and Configuration Tips."

3. Install a signed server certificate into IIS or Apache.

   For IIS, see "Installing Certificates for Use by IIS," in the second appendix.

6.2.3 Configuring a .NET Client

On each client machine:

- Install the CA certificate where needed by the operating system. This must be the CA that signs the server certificate.

  See "Installing Certificates for Use by Windows," in the second appendix.

6.2.4 Configuring a Java Client

On each client machine:

- For the CA that signs the server certificate, install the CA certificate into the JVM Keystore.

  See "Installing Certificates for Use by Java," in the second appendix.

6.3 Overview of SSL Encryption with Mutual Authentication

In some cases, you might want the server to authenticate the client, rather than permit the client to be anonymous as described earlier in this chapter. In SSL encryption with mutual authentication, the SSL handshake includes some additional steps:

1. When the server replies initially to the client, the server requests the client certificate. This message is in clear text.
2. The client replies with its certificate. This message is in clear text.
3. The server validates the client certificate. The server determines which CA signed the client certificate. The server then checks its own stored certificates to see if it recognizes this certificate authority, and to see if the signature and all the identifying details are correct. The server also checks that the certificate has not expired.

6.4 Setting Up SSL with Mutual Authentication

For greater security, you can modify the client to use mutual SSL encryption.

6.4.1 Prerequisites

- Set up one-way SSL, as described earlier in this chapter.
- Obtain a client certificate signed by a CA, which may or may not be the same as the CA that signed the server certificate.
- Obtain the certificate for that CA.

For information on generating test certificates, see "Generating Sample Certificates."

6.4.2 Configuring the Server

If the CA that signs the client certificate is not already known to the server:

- Install the certificate for that CA where needed by the operating system.

  See "Installing Certificates for Use by Windows," in the second appendix.
- Install the same certificate into the Web server.

  For IIS, see "Installing Certificates for Use by IIS," in the second appendix.

6.4.3 Configuring a .NET Client

- Install the client certificate where needed by the operating system.

  See "Installing Certificates for Use by Windows," in the second appendix.
- For the CA that signed the client certificate, install the CA certificate where needed by the operating system.

  See "Installing Certificates for Use by Windows," in the second appendix.
- Provide the client certificate along with every SOAP request. For a .NET client, just add a new instance of a certificate to the ClientCertificates collection, as follows:

```
WSCookbook service = new WSCookbook();

string addr = "localhostClient.cer";
service.ClientCertificates.Add(new X509Certificate(addr));
```

6.4.4 Configuring a Java Client

- Install the client certificate where needed by the operating system.

  See "Installing Certificates for Use by Windows," in the second appendix.
- For the CA that signed the client certificate, install CA certificate into the Java Keystore.

  See "Installing Certificates for Use by Java," in the second appendix.
- Provide the client certificate along with every SOAP request. For a Java client, because Web connections are handled by the JAVA VM, we have to declare a few VMARGS in order to correctly set up the VM to accept mutually authenticated SSL connections. (These must be supplied when calling the VM):

```
-Djavax.net.ssl.trustStore=(exact path ex: c:\cacerts)
-Djavax.net.ssl.trustStorePassword=(password)
-Djavax.net.ssl.keyStore =(exact path ex: c:\certificate.jks)
-Djavax.net.ssl.keyStorePassword=(password)
```

Optionally if the keystore is not in the .jks format:

```
-Djavax.net.ssl.keyStoreType=(type ex: pkcs12)
```

You can also do this programmatically by using:

```
System.setProperty("javax.net.ssl.keyStore", "path");
```

7

Setting Up XML Encryption

- An overview of XML encryption
- How to set up certificates for use by a Caché Web service
- How to modify a Caché Web service to use XML encryption

7.1 Overview

XML Encryption encrypts the SOAP body with an X.509 certificate. Specifically, the public key in the certificate is used to encrypt the data, which is then included as an <EncryptedData> element in the SOAP body. This technique encrypts the data so that only the application that receives the message can understand it. (In contrast, with SSL encryption, the Web server has access to the unencrypted message.)

7.2 Setting Up Certificates in Caché

When a client sends an encrypted message to the Web service, the Web service first tries to validate the attached certificate. One step is to examine the signature on the certificate, which should be the signature of a CA known to Caché. For this reason, you must install the CA certificate into Caché.

Caché must also have the server certificate and private key installed, so that the Web service can send it.

For information on installing certificates into Caché, see "Installing Certificates for Use by Caché," in the second appendix.

7.3 Modifying Your Web Service

To use the server certificate to encrypt the messages sent by a Web service:

1. First add the SECURITYIN parameter to your Web service:

```
Parameter SECURITYIN = "REQUIRE";
```

This can equal “REQUIRE”, “ALLOW” or “IGNORE”.

- REQUIRE forces all incoming clients to use encryption.
- ALLOW accepts incoming clients using encryption and will choose to respond with an encrypted message if credentials match.
- IGNORE ignores any incoming encryption and responds only in clear text.

2. Next retrieve the signing credentials:

```
//Grab Credentials
set x509alias = "Sign"
set pwd="signingkey"
set cred = ##class(%SYS.X509Credentials).GetByAlias(x509alias, pwd)
```

You use the **GetByAlias()** method of %SYS.X509Credentials to retrieve the credentials. The second argument is the password for the private key; you need to include this only if you did not load that password into Caché when you installed the certificate.

3. Next we will create an EncryptedKey object and add the signing credentials to that:

```
//get EncryptedKey element and add it
set enc=##class(%XML.Security.EncryptedKey).CreateX509(cred,,$$$SOAPWSReferenceThumbprint)
```

4. Finally we must add this EncryptedKey object to the outgoing message:

```
do ..SecurityOut.AddElement(enc)
```

5. From this point on, everything that the Web service generates will be encrypted, and all unencrypted incoming messages will be deferred.

8

Adding and Using Custom SOAP Headers

This chapter describes how to add custom SOAP headers to a Caché Web service and how to generate clients that recognize those custom headers. It discusses the following topics:

- How to add custom SOAP headers
- How to modify the WSDL to include the custom headers
- How to generate and use the proxy client classes

8.1 Adding Custom SOAP Headers in a Caché Web Service

First create a header class, for example:

```
Class ExamplePackage.HeaderExample Extends %SOAP.Header
{

Parameter NAMESPACE = "http://localhost:57772/csp/cookbook";

Property Information As %String;

}
```

Now in order for Caché to recognize your header in an incoming SOAP message, you must add that header to the *SOAPHEADERS* parameter much like this:

```
Parameter SOAPHEADERS = "HeaderExample:ExamplePackage.HeaderExample";
```

To add the header to the outgoing SOAP message, you add an instance of it to the *HeadersOut* array. For example, your Web service might include a utility method like this:

```
/// Add HeaderExample to headers
Method AddInformation(information As %String)
{
    //create a new MySession object
    set h = ##class(ExamplePackage.HeaderExample).%New()

    //add the information to h
    set h.Information = information

    //send this header in the soap message
    do ..HeadersOut.SetAt(h, "HeaderExample")
}
```

The Web methods would invoke this utility method.

To retrieve the header from an incoming SOAP message, look for it in the *HeadersIn* array:

```
/// Get header information
Method GetInformation() As %String
{
    //initialize information
    set information = "0"

    //read the header if it exists
    set head = ..HeadersIn.GetAt("HeaderExample")

    //if the header exists set the sid to that
    if (head '= $$$NULLOREF)
    {
        set information = ..HeadersIn.GetAt("HeaderExample").Information
    }

    Quit information
}
```

8.2 Editing the WSDL

Because Caché adds the custom headers at run time, the custom headers are not included in the WSDL document that Caché generates. If you need to use the WSDL to generate clients, save the WSDL document as a file and edit the WSDL manually so that it contains information about the headers. This section demonstrates how to do this with a simple example Web service:

```
Let us take this example Web Service:
Class ExamplePackage.ExampleService Extends %SOAP.WebService
{

    Parameter SERVICENAME = "MainService";

    Parameter NAMESPACE = "http://localhost:57772/csp/cookbook";

    Parameter SECURITYIN = "ALLOW";

    Parameter SOAPSESSION = 0;

    Parameter SOAPHEADERS = "HeaderExample:ExamplePackage.HeaderExample";

    Method test() As %String [ WebMethod ]
    {
    QUIT "Hello World!"
    }

}
```

The WSDL for this looks like (with line breaks added for readability).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://localhost:57772/csp/cookbook"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
targetNamespace="http://localhost:57772/csp/cookbook">
  <types>
   <s:schema elementFormDefault="qualified" targetNamespace="http://localhost:57772/csp/cookbook">

      <s:element name="test">
        <s:complexType>
          <s:sequence/>
        </s:complexType>
      </s:element>
      <s:element name="testResponse">
        <s:complexType>
          <s:sequence>
            <s:element name="testResult" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="testSoapIn">
    <part name="parameters" element="s0:test"/>
  </message>
  <message name="testSoapOut">
    <part name="parameters" element="s0:testResponse"/>
  </message>
  <portType name="MainServiceSoap">
    <operation name="test">
      <input message="s0:testSoapIn"/>
      <output message="s0:testSoapOut"/>
    </operation>
  </portType>
  <binding name="MainServiceSoap" type="s0:MainServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="test">
      <soap:operation
          soapAction="http://localhost:57772/csp/cookbook/ExamplePackage.ExampleService.test"
          style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="MainService">
    <port name="MainServiceSoap" binding="s0:MainServiceSoap">
     <soap:address location="http://localhost:57772/csp/cookbook/ExamplePackage.ExampleService.cls"/>

    </port>
  </service>
```

To add our custom header, we must first define a namespace for it:

```
xmlns:Headers="http://localhost:57772/csp/cookbook"
```

We must add this namespace declaration to the <definitions> element in the WSDL:

```
<definitions
xmlns:Headers="http://localhost:57772/csp/cookbook"
xmlns="http://schemas.xmlsoap.org/wsdl/"
...
>
```

Next we must add a definition of the HeaderExample class:

```
<s:schema elementFormDefault="qualified" targetNamespace="http://localhost:57772/csp/cookbook">
      <s:element name="HeaderExample" type="Headers:HeaderExample"/>
      <s:complexType name="HeaderExample">
        <s:sequence>
          <s:element name="Information" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:schema>
```

This must go within the <type> element in the WSDL.

We must next acknowledge that MySession is going to be part of the SOAP message:

```
<message name="HeaderExampleHeader">
    <part name="HeaderExample" element="Headers:HeaderExample"/>
</message>
```

The last thing we have to do is make sure that the WebService proxy makes sure to look for the header and sends the header with each SOAP message:

```
<soap:header message="s0:HeaderExampleHeader" part="HeaderExample" use="literal"/>
```

Simply add that tag after every <soap:body> tag:

```
    <operation name="test">
<soap:operation
 soapAction="http://localhost:57772/csp/cookbook/
 ExamplePackage.ExampleService.test" style="document"/>
      <input>
        <soap:body use="literal"/>
  <soap:header message="s0: HeaderExampleHeader"
   part="HeaderExample"use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
  <soap:header message="s0: HeaderExampleHeader"
   part="HeaderExample"use="literal"/>
      </output>
    </operation>
```

Tip: Make sure you add the <soap:header> tag in every operation.

8.3 Generating and Using the Client

8.3.1 Generating and Using a .NET Client

.NET automatically recognizes the header and generates the proxy classes correctly.

The .NET proxy automatically sets and adds an instance of the header as needed.

8.3.2 Generating and Using a Java Client

For Java, the wsimport utility does not recognize the header by default. Make sure to use the –XadditionalHeaders flag when using this utility. For example:

```
wsimport wsdl-location -keep  -XadditionalHeaders
```

Where *wsdl-location* is the URL of the WSDL.

The JAVA proxy does not automatically handle the custom header. You must first initialize a Holder for that object, and pass that holder as an argument:

```
import javax.xml.ws.Holder;

ExampleService example = new ExampleService();

ExampleServiceSoap exampleSoap = example.getExampleServiceSoap();

Holder<ExampleHeader> h = new Holder<ExampleHeader>();

System.out.println(exampleSoap.test(h));
```

A

Batch Script to Generate Sample Certificates

This appendix shows a batch script that you can use to generate sample certificates for use in simple testing.

CAUTION: Do not use these certificates in a production environment. Instead obtain certificates from an issuer such as Verisign.

This appendix is not intended as a comprehensive user guide for OpenSSL. It demonstrates only a single, simple procedure.

A.1 Prerequisites

1. Download OpenSSL. See "Useful Software," earlier in this book.
2. OpenSSL requires a configuration file (typically called openssl.cnf). When you download OpenSSL, download a sample.
3. Choose a directory to work in and copy the sample OpenSSL configuration file into it.
4. Open the openssl.cnf file and make edits as appropriate. This file specifies default values that OpenSSL uses.

   In particular, if this file sets the following variables, OpenSSL will not prompt you for these items:

   ```
   countryName = US
   stateOrProvinceName = Massachusetts
   localityName = Cambridge
   organizationName = InterSystems
   organizationalUnitName = Development
   ```

5. Copy the script from the next section of the documentation into a text file named Generate_Certificates.bat

   Place this file in the same directory as openssl.cnf.

A.2 Sample Batch Script

The following batch script generates CA, server, and client certificates:

```
@ECHO OFF

REM Set this local variable before running this batch file
SET CN=localhost
REM Other variables are set in openssl.cnf

ECHO.
ECHO ######################################################
ECHO # This batch file will generate three certificates: #
ECHO # CA.cer - The Certificate Authority Certificate    #
ECHO # Server.cer - The Server Certificate               #
ECHO # Client.cer - The Client Certificate               #
ECHO ######################################################

REM CA Certificate stuff

ECHO.
ECHO.
ECHO Generating CA key:
ECHO.
openssl genrsa -aes256 -out CA.key 2048


ECHO.
ECHO.
ECHO Generating CA Certificate:
ECHO.
openssl req -new -x509 -days 3095 -key CA.key -out CA.cer -config openssl.cnf -subj /CN=%CN%_CA


REM Server certificate stuff

ECHO.
ECHO.
ECHO Generating Server key:
ECHO.
openssl genrsa -aes256 -out Server.key 2048


ECHO.
ECHO.
ECHO Generating Server Certificate Request:
ECHO.
openssl req -new -key Server.key -out Server.csr -config openssl.cnf -subj /CN=%CN%_Server

ECHO.
ECHO.
ECHO Signing Server Certificate Request with CA certificate.
ECHO.
openssl x509 -req -days 365 -in Server.csr -CA "CA.cer" -CAkey "CA.key" -set_serial 01 -out Server.cer

ECHO.
ECHO.
ECHO Generating Client key:
ECHO.
openssl genrsa -aes256 -out Client.key 2048


ECHO.
ECHO.
ECHO Generating Client Certificate Request:
ECHO.
openssl req -new -key Client.key -config openssl.cnf -out Client.csr -subj /CN=%CN%_Client


ECHO.
ECHO.
ECHO Signing Client Certificate Request with CA certificate.
ECHO.
openssl x509 -req -days 365 -in Client.csr -CA "CA.cer" -CAkey "CA.key" -set_serial 02 -out Client.cer

ECHO.
ECHO ######################################################
ECHO #                        END                         #
ECHO ######################################################
ECHO.
```

This batch file does the following:

1. Defines local variables (`C`, for example) that are used in subsequent steps. Edit them as needed.

2. Generates the file `CA.key`:

   ```
   openssl genrsa -aes256 -out CA.key 2048
   ```

   Here `-aes256` specifies a particular cipher suite; there are other options. For further information, see the OpenSSL documentation.

   This step generates a public/private key pair. It does the following:

   a. Creates the file.

   b. Prompts you for the password (a "pass phrase") to use to secure the private key. This ensures that you are the only person who can access the private key.

   c. Prompts you to confirm the password.

   d. Updates the file.

3. Generates `CA.cer`, which is the CA certificate file:

   ```
   openssl req -new -x509 -days 3095 -key CA.key -out CA.cer -config openssl.cnf -subj /CN=%CN%_CA
   ```

   This step prompts you for the private key password and then generates the certificate file.

   Depending on whether you have edited your openssl.cnf file, OpenSSL might also prompt you for information to identify the certificate.

   This certificate includes the public key generated in the previous step.

4. Generates the key file `Server.key`:

   ```
   openssl genrsa -aes256 -out Server.key 2048
   ```

   This step generates a public/private key pair. As earlier, OpenSSL prompts you for a password and asks you to confirm it. This ensures that you are the only person who can access the private key for the server's public/private key pair.

5. Generates `Server.csr`, which is the unsigned server certificate file:

   ```
   openssl req -new -key Server.key -out Server.csr -config openssl.cnf -subj /CN=%CN%_Server
   ```

   This step prompts you for the private key password and then generates the certificate file.

   Depending on whether you have edited your openssl.cnf file, OpenSSL might also prompt you for information to identify the certificate.

   This certificate includes the public key generated in the previous step.

6. Generates `Server.cer`, which is the signed server certificate file:

   ```
   openssl x509 -req -days 365 -in Server.csr -CA "CA.cer" -CAkey "CA.key" -set_serial 01 -out Server.cer
   ```

   This step prompts you for the private key password of the CA certificate. (Note that a certificate issuer would perform this step for you rather than supplying you with its private key password.)

   This certificate file is signed with the CA certificate.

7. Generates the key file `Client.key`:

   ```
   openssl genrsa -aes256 -out Client.key 2048
   ```

This step generates a public/private key pair. As earlier, OpenSSL prompts you for a password and asks you to confirm it. This ensures that you are the only person who can access the private key for the client's public/private key pair.

8. Generates `Client.csr`, which is the unsigned client certificate file:

   ```
   openssl req -new -key Client.key -config openssl.cnf -out Client.csr -subj /CN=%CN%_Client
   ```

   This step prompts you for the private key password and then generates the certificate file.

   Depending on whether you have edited your openssl.cnf file, OpenSSL might also prompt you for information to identify the certificate.

   This certificate includes the public key generated in the previous step.

9. Generates `Client.cer`, which is the signed client certificate file:

   ```
   openssl x509 -req -days 365 -in Client.csr -CA "CA.cer" -CAkey "CA.key" -set_serial 02 -out Client.cer
   ```

   This step prompts you for the private key password of the CA certificate. (Note that a certificate issuer would perform this step for you rather than supplying you with its private key password.)

   This certificate file is signed with the CA certificate.

A.3 Using the Script

To use this sample script:

1. Open a command prompt.
2. Navigate to the directory that contains the sample script (Generate_Certificates.bat) and your openssl.cnf file.
3. Execute the script by entering the following command:

   ```
   Generate_Certificates
   ```

4. Respond to the prompts as needed.

B

Installing Certificates

This appendix describes how to install certificates into various locations. This appendix is not intended to be comprehensive.

- For use by Windows
- For use by Java
- For use by Caché
- For use by IIS

Note that a .pfx file contains a private key, while a .cer or .cer file contains a certificate (which includes a public key).

Also, the certificate file types .pem, .cer, .der, and .crt are equivalent.

B.1 Installing Certificates for Use by Windows

B.1.1 Scenarios

Scenario	Certificate Needed	Notes
SSL with one-way authentication	CA certificate of the CA that signs the server certificate	Needed on both server and client machines
SSL with mutual authentication	CA certificate of the CA that signs the client certificate	Needed on both server and client machines
XML Encryption	• Client certificate and its private key • CA certificate of the CA that signs the client certificate • Server certificate • CA certificate of the CA that signs the server certificate	Details depend upon the .NET libraries used. Further information is currently beyond the scope of this documentation.

B.1.2 Converting Certificates into PCCS#12 Format

If a certificate is in PEM/CER format, first convert it to PKCS#12 format.

You can do this with the OpenSSL Toolkit; see "Useful Software," earlier in this book. For this tool, enter a command like the following in a shell:

```
openssl pkcs12 -export -in newcert.cer -inkey "..\newkey.cer"
-out newcert.pfx -name "New Cert"
```

Note: Do not include the line break, which is included here only for readability.

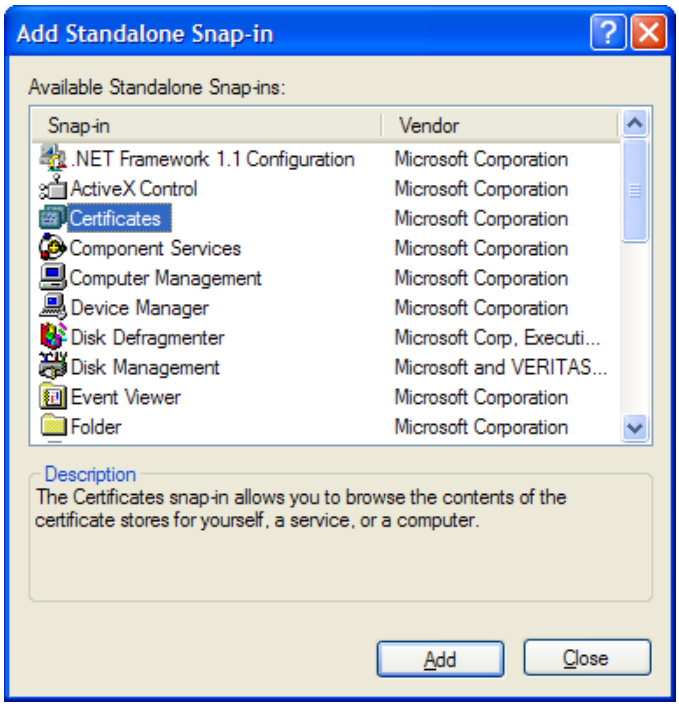
The tool will prompt you for the pass phrase for the .cer file, as well as for the export password. For example:

```
Loading 'screen' into random state - done
Enter pass phrase for ..\newkey.cer:
Enter Export Password:
Verifying - Enter Export Password:
```

B.1.3 Adding the Certificates Snap-ins to the MMC Console

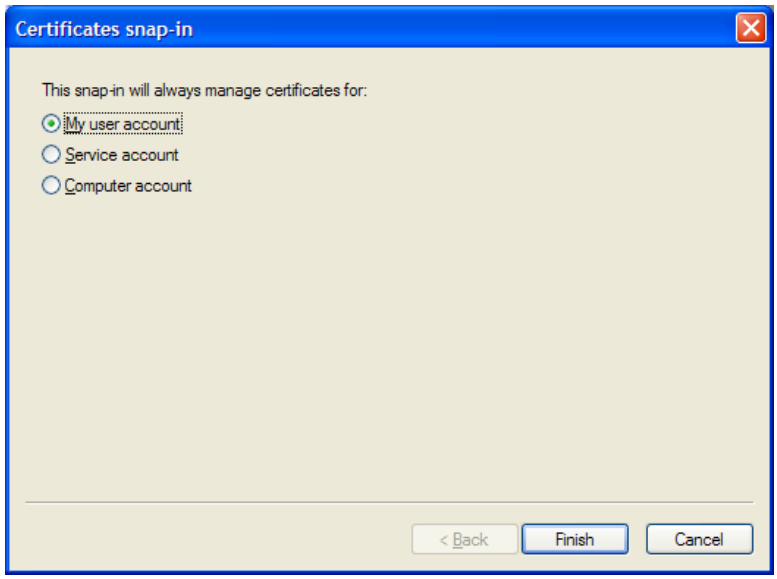
To add the certificates snap-ins to the Microsoft Management Console:

1. Click **Start > Run**.
2. Type `mmc` and press **Return**.
3. If you already have a console configuration that you want to modify, open it via **File > Open**.
4. Click **File > Add/Remove Snap-in**.
5. Click **Add ...**.

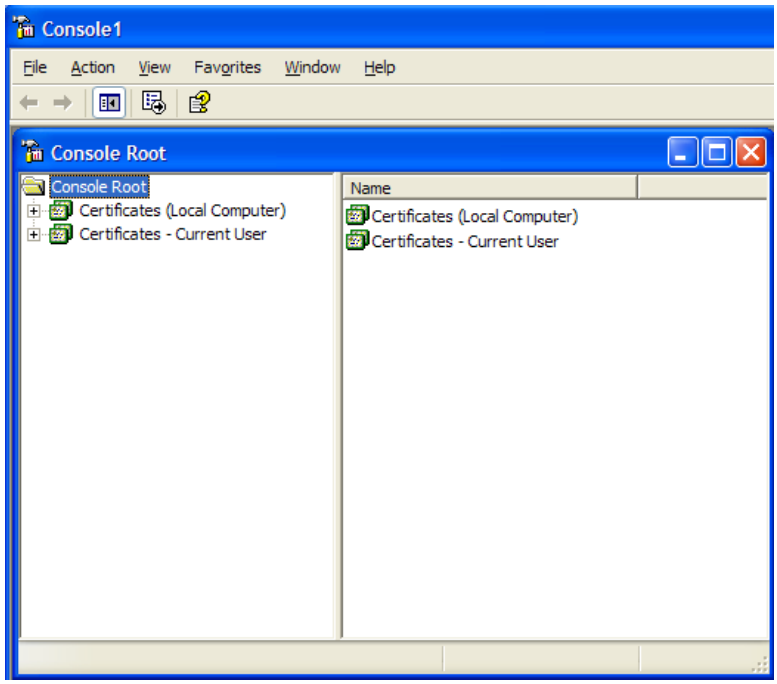


6. Click **Certificates**.

7. Click **Add**.



8. Click the **Computer account** option and then **Next**.
9. Click **Finish**.
10. Click **Certificates** and then click **Add** again.
11. Click **My user account**.
12. Click **Finish**.
13. Click **OK**.
14. Click **Close**.
15. Click **OK**.



B.1.4 Installing a CA Certificate

To install a CA certificate:

1. If a certificate is in PEM/CER format, first convert it to PKCS#12 format, as described earlier in this section.
2. Open the Microsoft Management Console as described in the first steps of "Adding the Certificates Snap-ins to the MMC Console."
3. Use **File > Open** to open the MMC configuration that you saved earlier.
4. Find **Certificates (Local Computer) > Trusted Root Certificate Authorities** on the left side.
5. Right-click **Trusted Root Certificate Authorities** and then click **All Tasks > Import...**.
6. Browse and select the CA certificate file.

B.1.5 Installing a Server Certificate

To install a server certificate:

1. If a certificate is in PEM/CER format, first convert it to PKCS#12 format, as described earlier in this section.
2. Open the Microsoft Management Console as described in the first steps of "Adding the Certificates Snap-ins to the MMC Console."
3. Use **File > Open** to open the MMC configuration that you saved earlier.
4. Find **Certificates (Local Computer) > Personal** on the left side.
5. Right-click **Personal** and then click **All Tasks > Import...**.
6. Browse and select the server certificate file.

B.1.6 Installing a Client Certificate

To install a client certificate:

1. If a certificate is in PEM/CER format, first convert it to PKCS#12 format, as described earlier in this section.
2. Open the Microsoft Management Console as described in the first steps of "Adding the Certificates Snap-ins to the MMC Console."
3. Use **File > Open** to open the MMC configuration that you saved earlier.
4. Find **Certificates-Current User > Personal** on the left side.
5. Right-click **Personal** and then click **All Tasks > Import...**.
6. Browse and select the client certificate file.

B.2 Installing Certificates for Use by Java

On the client machine, Java Virtual Machine uses its own certificate store (the JVM Keystore) instead of the Windows store.

B.2.1 Scenarios

Scenario	Certificate Types Needed
SSL with one-way authentication	CA certificate of the CA that signs the server certificate
SSL with mutual authentication	• CA certificate of the CA that signs the server certificate • CA certificate of the CA that signs the client certificate
XML Encryption	• Client certificate and its private key • CA certificate of the CA that signs the client certificate • Server certificate • CA certificate of the CA that signs the server certificate Details depend upon the Java libraries used. Further information is currently beyond the scope of this documentation.

B.2.2 Details

To do this, you need to use the Java Keytool (Portecle) or an equivalent; see "Useful Software," earlier in this book.

1. Run the portecle.jar file.
2. Go to **File > Open CA Certs Store**.

   (Note that the default JVM CA Certs Store Password is `changeit`.)
3. Go to **Tools > Import Trusted Certificate**.
4. Select the CA certificate file (usually .pem or .cer).
5. Confirm this is a trusted certificate.
6. Accept the certificate as trusted.
7. Enter an alias.
8. Go to **File > Save Keystore**.

B.3 Installing Certificates for Use by Caché

B.3.1 Scenarios

Scenario	Certificate Type Needed
SSL with one-way authentication	None (handled by Web server)
SSL with mutual authentication	None (handled by Web server)
XML Encryption	• CA certificate • Server certificate signed by this CA

B.3.2 Installing the CA Certificate in Caché

Rename the CA certificate to cache.cer and place the file into the *install-dir*`/mgr`, where *install-dir* is the directory into which you installed Caché.

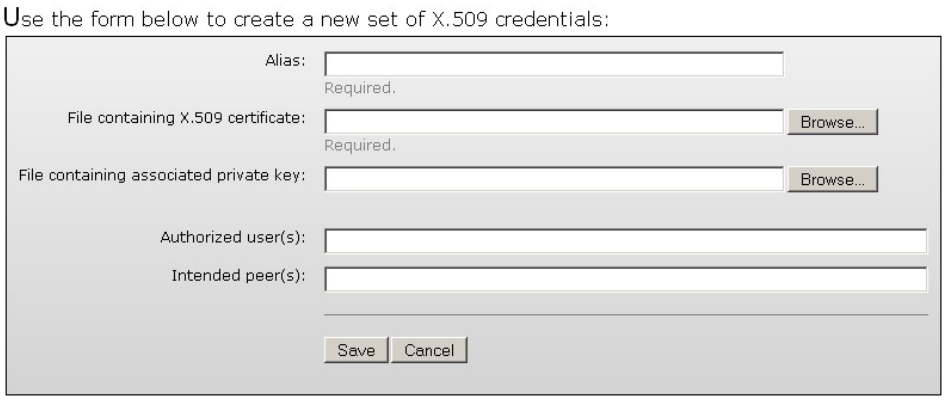
Tip: The certificate file types .pem, .cer, .der, and .crt are equivalent. For example, you can rename a file named sampleCAcert.pem to cache.cer.

B.3.3 Installing the Server Certificate and Private Key

To install the server certificate and private key into Caché:

1. Open up the System Management Portal for your current instance of Caché.
2. Click **System Administration, Security Management**.

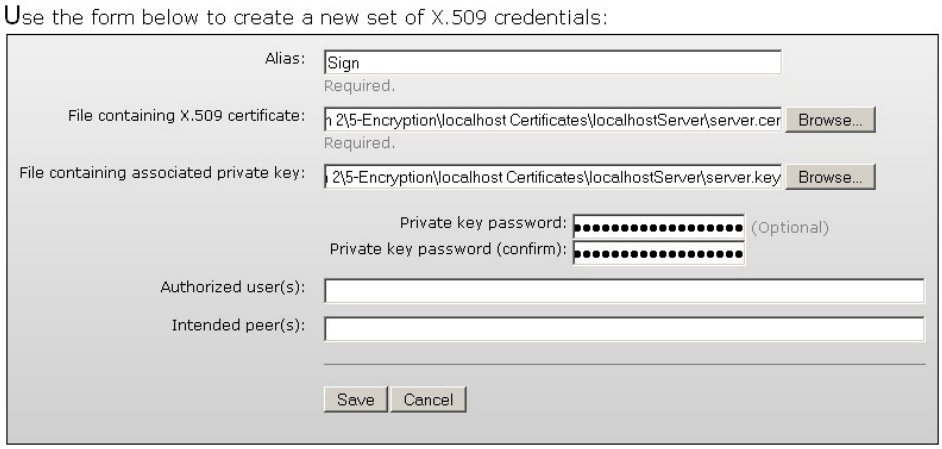
   The system displays another page.
3. Click the link **X.509 Credentials**.
4. Click the link **Create New Credentials**, at the top of the page. Caché then displays the following page:



5. For **Alias**, specify an alias. For example, `Sign`.

6. Click **Browse** and find the server certificate file.
7. Click **Browse** and find the corresponding private key file.

   When the key file is added, you are prompted for the key password. If you want added security, you can choose not to include the password with the key file and instead provide the password manually with every request for this certificate.
8. Enter the corresponding key password.



9. Click **Save**.

Note: If the entry did not save, or you were thrown an error, please make sure that the certificate has been signed by the CA whose certificate you installed into Caché in the previous section.

B.3.4 Installing Client Certificates

To install a client certificate, follow the steps described in the previous section, except that you should not include the private key. A private key must not be known by anyone but the entity that owns it.

B.4 Installing Certificates for Use by IIS

Also see the appendix "IIS Installation and Configuration Tips."

B.4.1 Scenarios

Scenario	Certificate Type
SSL with one-way authentication	Signed server certificate
SSL with mutual authentication	Signed server certificate; also enable certificate trust list
XML Encryption	None (Web server does not participate)

B.4.2 Installing the Server Certificate into the IIS Certificate Store

1. Click **Start > All Programs > Administrative Tools > Internet Information Services**.
2. Find your Web site and right click on it.
3. Click on **Properties**.
4. Go to the **Directory Security** tab.
5. Click on the **Server Certificate** button.
6. Click **Next > Assign an existing certificate**.
7. Find Localhost issued by localhost CA.
8. Click **Next**.
9. Click **Finish**.

C

IIS Installation and Configuration Tips

C.1 Installing IIS and Reconfiguring Caché

To install IIS:

1. Go to the **Control Panel > Add/Remove Programs**.
2. Click **Add/Remove Windows Components**.
3. Select **Internet Information Services (IIS)**.
4. Click **Next**.

   Windows then installs IIS.
5. Click **Finish**.

After installing IIS, modify the Caché installation, as follows:

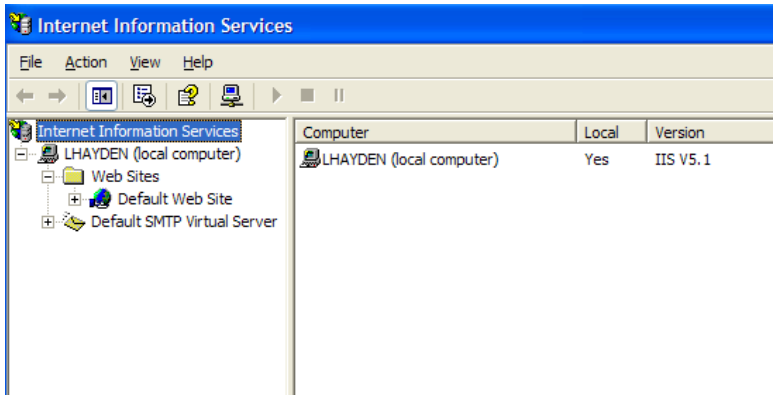
1. Run the Caché installer.
2. Select the current instance of Caché running and then click **Next**.
3. Select **Modify**.
4. Click the plus sign next to **Web Server Gateway**. The installer expands the list.
5. Click the icon next to **CSP for IIS**. The installer displays a drop-down list of choices.
6. Select **This feature will be installed on local hard drive**.
7. Continue through the rest of the installation.

C.2 Configuring IIS

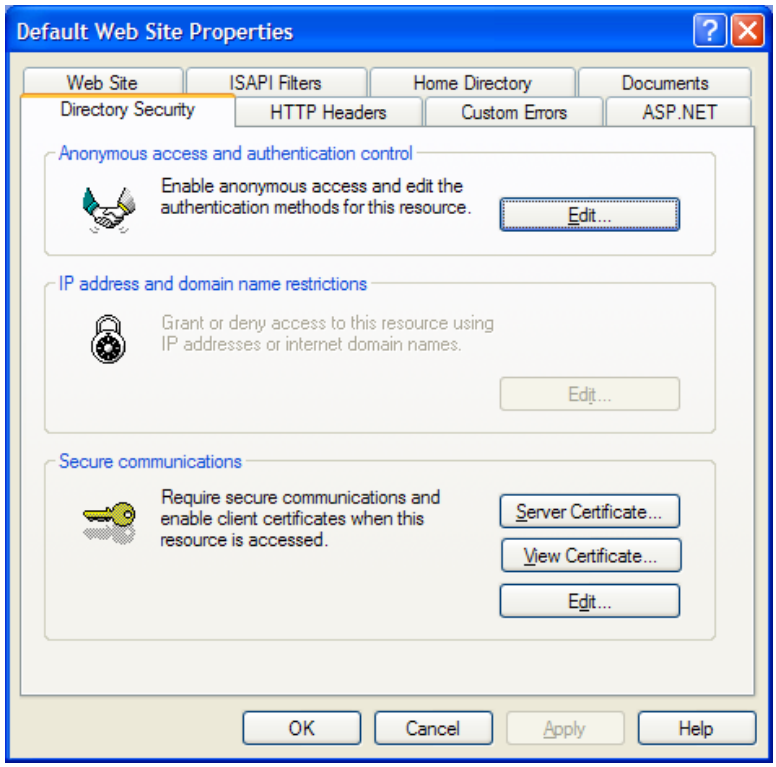
Before configuring IIS, install the certificates to the Windows Certificate Store and to IIS; see the second appendix. Then do the following:

1. Click **Start > All Programs > Administrative Tools > Internet Information Services**.

2. Find your Web site (for example, `Default Web Site`) and right click it.

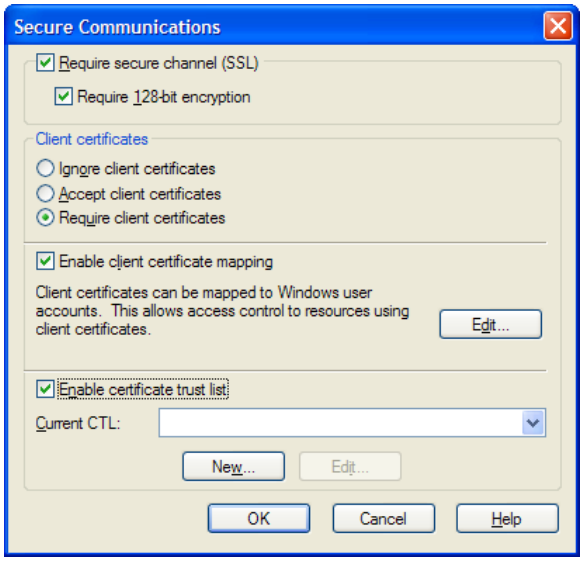


3. Click **Properties**.
4. Click the **Directory Security** tab.



5. In the **Secure Communications** section, click the **Edit** button.

   If this button is not available, check that you have installed a certificate into IIS.



6. Select **Require secure channel (SSL)**.
7. Select **Require 128-bit encryption**.
8. Enable IIS to verify client certificates as follows:
   a. Under **Client Certificates**, select **Require client certificates**.
   b. Select **Enable certificate trust list**.
   c. Below **Enable certificate trust list**, click **New...**.
   d. Click **Next** and then click **Add from Store**.
   e. Scroll to the certificate and click it.
   f. Click **OK**.
   g. Click **Next** twice.
   h. Click **Finish**.
9. Click **OK**.
10. Click **Apply**.

C.3 Testing That It Works

After you have installed all the certificates in your Windows Certificate Store, and then installed the correct server certificate in IIS, you can try to access https://localhost from Internet Explorer. If you get an error message, you have most likely skipped a step.

In Firefox, you cannot access https://localhost unless you have also added the CA certificate to the Firefox Trust Store as well as selected **Ignore Client Certificates** in IIS.