

Sending Secure Electronic Mail (S/MIME) in Java (CAPS) the Easy Way

Michael.W.Czapski@gmail.com

May, 2009

Table of Contents

Introduction.....	1
SecMail Class Library and Pre-requisites Download	1
Setting up Cryptographic Objects	2
(Re) Configuring sample Java Sender	4
Java CAPS 6 Repository JCD Example.....	6
Using Outlook Express to Read Secure Email	8
Summary	25

Introduction

Every now and then one needs to secure communications between parties. Some would say it is necessary to do that all the time and perhaps it is. The issues are the complexity and expense. The complexity comes from having to configure a bunch of tools to support things like encryption and digital signatures for more than a single party. The expense comes from typically having to purchase cryptographic instruments from well known Certification Authorities, and keep on purchasing them all over again every 1 or 2 years. This discussion introduces a class library that offers a set of simple methods for constructing and sending secure electronic mail using the Secure Multipurpose Internet Mail Extensions (S/MIME), the Bounce Castle Cryptographic Libraries and the Java programming language. The intent is to allow a Java CAPS developer, or a Java developer, to add Secure Electronic Mail functionality quickly and easily, and without having to make too much of a time investment learning about PKI-based security and related matters. This addresses the complexity issue. The expense issue is addressed in my Blog Entry, "Producing Free, Private X.509 Certificates for use with PKI-based Solutions", at http://blogs.sun.com/javacapsfieldtech/entry/producing_free_private_x_509. That blog discusses how to roll out a private Certification Authority and obtain X.509 Certificates., and other cryptographic objects, for free.

This document discusses the use of cryptographic software and manipulation of cryptographic objects. Using or discussing cryptography software is illegal in some parts of the world. It is your responsibility to ensure that you comply with any import/export and use laws that apply to you.

SecMail Class Library and Pre-requisites Download

The SecMail Class Library, and most of the class libraries it depends on, is available for download from http://mediacast.sun.com/users/Michael.Czapski-Sun/media/SecMail_and_extra_libs.zip/details. Download the package if you intend to try what this document discusses.

The archive package does not contain the packages that actually implement the cryptographic methods and algorithms. You will need to download bccmail-jdk15-

143.jar and bcprov-jdk15-143.jar from the Bouncy Castle site, at http://www.bouncycastle.org/latest_releases.html. You may need to use different versions depending on the Java version you are using.

To use the SecMail class library, extract the SecMail.jar, activation.jar, mail.jar and log4j-1.2.8.jar from the SecMail_and_extra_libs.zip to a directory where your development environment can find them. For Java CAPS 5.x and 6 Repository-based projects you will need to import these files into your project. Java CAPS 5.x and 6 Repository developer is assumed to know how to do that. For regular Java developer do what you need to do depending on the development environment you use. For NetBeans, for example, add the JARs to the Library for your Java project.

Place the bcmail-jdk15-143.jar and bcprov-jdk15-143.jar, or the later version of the archives, in the same location as the SecMail and other packages.

Setting up Cryptographic Objects

In this discussion it is assumed that a sender, called msender, intends to send a secure electronic mail to a party called mreceiver. msender digitally signs an email message (using its own Private Key), encrypts the messages (using mreceiver's public key which is embedded in mreceiver's X.509 Certificate) and sends it to the nearest SMTP Server. To the SMTP Server secure email is just another MIME Multipart message.

The preceding paragraph implies that msender has access to its own Private Key cryptographic object (to Digitally Sign the message) and to mreceiver's X.509 Certificate cryptographic object (to encrypt the message so that only mreceiver can decrypt it). It is also implied that mreceiver has access to its Private Key (to decrypt the message) and msender's X.509 Certificate (to verify msender's digital signature).

The rest of this section deals with obtaining the appropriate cryptographic objects, manipulating them and getting them to a state where they can be used.

To use X.509 Certificates for electronic mail security in conjunction with the SecMail package one needs a Truststore. A cacerts truststore, in JKS format, is available with every JRE. Use it in place %JAVA_HOME%\jre\lib\security\cacerts (storepass is changeit) or copy it to a convenient location for exclusive use by your solution.

To this cacerts truststore import certificates of all parties with whom you wish to enter into secure communication, remembering to also import certificates of any Certification Authorities that signed these certificates, if they are not already there.

For the most part CA certificates of well known CAs, like Verisign, will already be in the cacerts. Private CA's certificates will not be already there so they need to be imported as well.

Using the tools and techniques discussed in my blog entry "Producing Free, Private X.509 Certificates for use with PKI-based Solutions" at http://blogs.sun.com/javacapsfieldtech/entry/producing_free_private_x_509, produce two sets of cryptographic objects – one for msender and one for mreceiver. When creating a Certificate Signing Request use the email address forms which your mail

system likes, for example msender@vulcan.fed if your email systems sits in the vulcan.fed domain and you are creating crypto objects for msender.

Assume you are using the private democa PKI discussed in my blog entry “Producing Free, Private X.509 Certificates for use with PKI-based Solutions”. Assume also that you will be using a private copy of the cacerts so you will need to copy it to a convenient spot.

```
cd C:\JCAPS6U1Projects\SecMail\pki
copy %JAVA_HOME%\jre\lib\security\cacerts .\
```

Assuming your PKI infrastructure, constructed using the method discussed in the blog above, is rooted at C:\JCAPS6U1Projects\SecMail\pki, the democa certificate will be in ca\democa\democa.pem.crt.

Issue the following import command to add the democa CA X.509 Certificate to the truststore:

```
C:\JCAPS6U1Projects\SecMail\pki>%JAVA_HOME%\bin\keytool -import -v -
alias democa -file ca\democa\democa.pem.crt -keystore cacerts -
storepass changeit -storetype jks -trustcacerts
```

When asked whether to trust this certificate answer “yes”.

The interaction will look similar to this:

```
C:\JCAPS6U1Projects\SecMail\pki>%JAVA_HOME%\bin\keytool -import -v -alias democa -file
ca\democa\democa.pem.crt -keystore cacerts -storepass changeit -storetype jks -
trustcacerts
Owner: EMAILADDRESS=certification@authority.com, CN=democa, OU=democa Security
Division, O=democa Certification Authority, L=Sydney, ST=NSW, C=AU
Issuer: EMAILADDRESS=certification@authority.com, CN=democa, OU=democa Security
Division, O=democa Certification Authority, L=Sydney, ST=NSW, C=AU
Serial number: 5
Valid from: Mon May 04 09:16:05 EST 2009 until: Fri Jul 21 09:16:05 EST 2017
Certificate fingerprints:
    MD5: 35:C0:B9:C9:1E:F5:34:19:8E:06:D4:B9:34:C9:D0:DE
    SHA1: C5:EB:CC:64:44:4D:E4:5C:C5:ED:20:05:DA:D9:9C:9B:9E:03:F2:1B
Trust this certificate? [no]: y
Certificate was added to keystore
[Storing cacerts]
```

Now that the democa is imported, which is only necessary if the end user certificates we will import next were signed by this private CA, we will import end use certificates of all the parties with whom we will communicate. In this case we only need mreceiver’s certificate if we expect to be encrypting messages for mreceiver or verifying digital signatures generated by mreceiver.

The mreceiver:

```
%JAVA_HOME%\bin\keytool -import -v -alias mreceiver -file
mreceiver\mreceiver.pem.crt -keystore cacerts -storepass changeit -
storetype jks -trustcacerts
```

The interaction is shown below.

```
C:\JCAPS6U1Projects\SecMail\pki>%JAVA_HOME%\bin\keytool -import -v -alias mreceiver -
file mreceiver\mreceiver.pem.crt -keystore cacerts -storepass changeit -storetype jks
-trustcacerts
Owner: EMAILADDRESS=mreceiver@some.company.com, CN=mreceiver, OU=mreceiver,
O=mreceiver, ST=NSW, C=AU
Issuer: EMAILADDRESS=certification@authority.com, CN=democa, OU=democa Security
Division, O=democa Certification Authority, L=Sydney, ST=NSW, C=AU
Serial number: 2
Valid from: Mon May 04 10:45:54 EST 2009 until: Fri Jul 21 10:45:54 EST 2017
Certificate fingerprints:
    MD5:  5E:5D:FB:5F:C2:BD:3E:0F:E2:58:D4:CA:19:07:D3:28
    SHA1: A0:11:B7:27:70:24:65:F8:C2:D2:16:B8:F0:55:1B:77:09:EB:E7:60
Trust this certificate? [no]: y
Certificate was added to keystore
[Storing cacerts]
```

mreceiver is the party to whom we will be sending secure messages.

For the msender, ourselves, we will use the PKCS#12 Keystore, which was generated as we followed the steps in the Blog Entry referred to above. We don't need to import msender's certificate because we never use it for cryptographic operations. Everybody else, which engages in secure communications with us, does.

(Re) Configuring sample Java Sender

SecMail.jar contains both the compiled classes and the Java sources of all classes. Amongst others, there is the SecMailSenderBC.java.

Extract this source file and inspect it to see what needs to be done to send a secure email wit or without attachments. I will discuss only the selected statements that may need to be modified to suit your environment.

The msender's keystore location, password and type may need changing:

Line 93:

```
String sSenderKeyStoreFilePath = "C:/JCAPS6U1Projects/SecMail/pki/msen
der/msender.pkcs12.keystore.p12";
```

Change the location of the msender's keystore if different from what is in the example.

Line 94:

```
String sSenderKeyPassPhrase = "msendermsender";
```

Change the passphrase of the msender's Keystore, if different

I assume you are using cryptographic objects generated by the scripts from the Blog Entry. If this is not the case then you know enough to know to change keystore type in line 95 if it is not a PKCS#12 keyatore.

The location and type of the truststore to which you added mreceiver's certificate may need to be changed.

Line 99:

```
String sTruststoreKeyStoreFilePath = "C:/JCAPS6U1Projects/SecMail/pki
/cacerts";
```

Change the location of the truststore if it is different

Line 100:

```
String sTruststoreKeyPassPhrase = "changeit";  
Change the truststore passphrase if it is different
```

Lines 107-109:

```
String sSMTPServer = "localhost";  
String sSMTPAcctUsername = "msender";  
String sSMTPAcctPassword = "msender";
```

Change the host, username and password for the SMTP Server. Secure SMTP Server is not supported – feel free to extend the library to support SMTP over SSL.

The annotation below the following statements discuss some aspects of the code. See the source for all there is to know.

```
jse = new SecMail(sSMTPServer, sSMTPAcctUsername, sSMTPAcctPassword);  
Create an instance of the SecMail class and configure it to use the appropriate SMTP Server.
```

```
jse.setLoggingOff();  
jse.setMailDebug(false);
```

Remove or comment out this is you would like to see verbose debug information and set the other to true for more verbose informaion.

```
jse.setEncrypt(true);
```

State whether you wish to have the message encrypted. False will leave encryption off. True will cause the receiver's certificate to be used for encryption.

```
jse.setSign(true);
```

State whether you wish to have the message signed. False will leave signature off. True will cause the sender's private key to be used for signing.

```
jse.setFrom("msender@some.company.com", "Mail Sender", senderKS, "msender", "msendermsender");  
jse.addReplyTo("msender@aus.sun.com.com", "Mick");
```

Configure the anem, the email addresses, the keystore containing the private key and keystore passphrase for the sender (eMail From entity).

```
jse.addTo("mreceiver@some.company.com", "Mail Receiver", truststoreKS1, "mreceiver");
```

Configure eMail To parameters – recipient address and name, and truststore from which the recipients certificate is to be exytracted and the keystore alias for that certificate.

```
jse.setSubject("Email test at " + new Date());
```

Set email subject

```
jse.addText("Hello Michael,\r\nThis is a test of email\r\n\r\nCheers\r\n");
```

Add body text of the message.

```
jse.addFileAttachment("c:/docs/TPMUserGuide.pdf");
```

Optionally add an attachment from a file in the file system

```
jse.addByteArrayAttachment("Hello this is a byte array 1".getBytes(),
    "ByteArray1.txt", "Byte Array 1");
```

Optionally add an attachment from a byte array and name the attachment

```
jse.addByteArrayAttachment("Hello this is a
byte array 2".getBytes());
```

Optionally add an attachment from a byte array without naming it.

```
jse.send();
```

Finally, send the message.

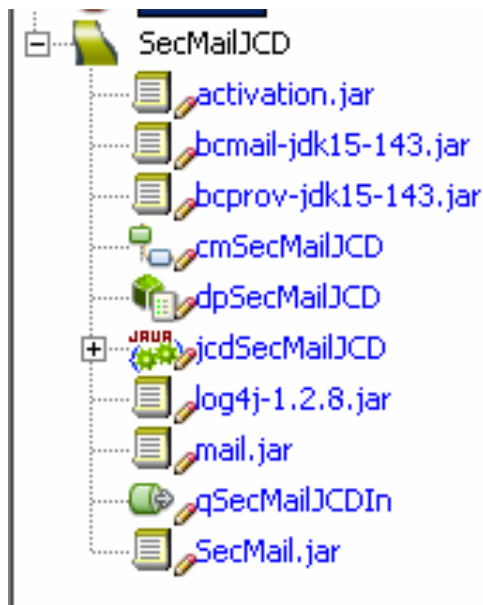
There are a number of methods one can use. One can, for example, add multiple recipients, CC recipients, and so on.

When the code executes it will connect to the SMTP Server, construct the mail message, signing and encrypting as necessary, and send it for forwarding to the mail recipient's mail server.

Feel free to explore the code. Bear in mind that I am not great shakes at Java programming. It's just another language which I learned enough of to be dangerous but not enough to be good at it. For me, in this case, the end justifies the means. If you are great shakes at Java, as lots of people in the works would be, feel free to re-write this code properly ☺

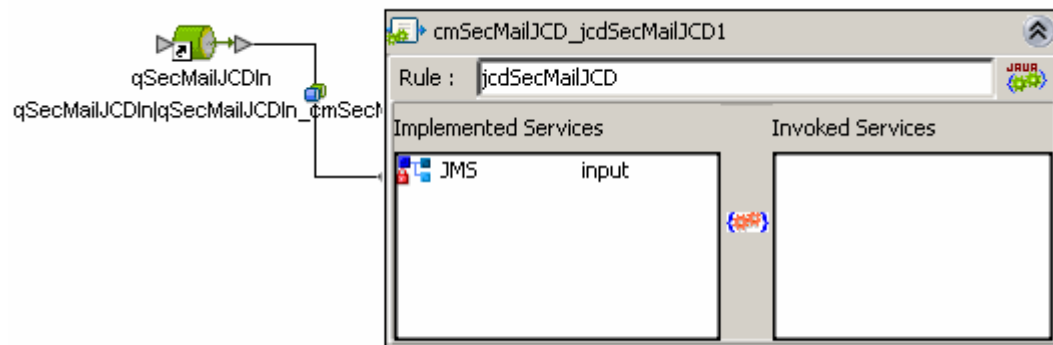
Java CAPS 6 Repository JCD Example

Here is a Java CAPS 6 Repository project that uses the SecMail class library to send secure email. The project hierarchy, including imported JARs is shown below.



Note that there is only 1 JCD here. It is triggered by a JMS message, the content of which it completely ignores. All 'variable' information, keystores, truststores, email addresses, etc., are hardcoded for this example. Your JCD would probably be a great

deal smarter/more dynamic about configuration. This JCD is merely an example of the use of the class library.
Here is the Connectivity Map.



Here is the complete source of the JCD.

```

package SecMailJCD;

import au.org.czapski.utils.crypto.*;
import java.security.KeyStore;
import java.util.Date;

public class jcdSecMailJCD
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive( com.stc.connectors.jms.Message input )
        throws Throwable
    {
        String sSenderKeyStoreFilePath =
            "C:/JCAPS6U1Projects/SecMail/pki/msender/msender.pkcs12.keystore.pl2";
        String sSenderKeyPassPhrase = "msendermsender";
        CryptoUtils sender_cul = new CryptoUtils();
        if (!logger.isDebugEnabled()) {
            sender_cul.setLoggingOff();
        } else {
            sender_cul.setDebug();
        }
        KeyStore senderKS = sender_cul.getKeyStoreFromFile
            ( sSenderKeyStoreFilePath, sSenderKeyPassPhrase, "PKCS12" );
        String sTruststoreKeyStoreFilePath =
            "C:/JCAPS6U1Projects/SecMail/pki/cacerts";
        String sTruststoreKeyPassPhrase = "changeit";
        CryptoUtils truststore_cu2 = new CryptoUtils();
        if (!logger.isDebugEnabled()) {
            truststore_cu2.setLoggingOff();
        } else {
            truststore_cu2.setDebug();
        }
        KeyStore truststoreKS1 = truststore_cu2.getKeyStoreFromFile
            ( sTruststoreKeyStoreFilePath, sTruststoreKeyPassPhrase, "JKS" );
        SecMail jse = null;
        String sSMTPServer = "localhost";
        String sSMTPAcctUsername = "msender";
        String sSMTPAcctPassword = "msender";
        jse = new SecMail( sSMTPServer, sSMTPAcctUsername, sSMTPAcctPassword );
        if (!logger.isDebugEnabled()) {
            jse.setLoggingOff();
        } else {
            jse.setDebug();
        }
        jse.setMailDebug( false );
        jse.setEncrypt( true );
    }
}

```

```

jse.setSign( true );
jse.setFrom(
    "msender@some.company.com"
    , "Mail Sender", senderKS, "msender", "msendermsender" );
jse.addReplyTo( "msender@aus.sun.com.com", "Mick" );
jse.addTo(
    "mreceiver@aus.sun.com"
    , "Mail Receiver", truststoreKS1, "mreceiver" );
jse.setSubject( "Email test at " + new Date() );
jse.addText
    ( "Hello Michael,\r\nThis is a test of email\r\n\r\nCheers\r\n" );
jse.addFileAttachment( "c:/docs/TPMUserGuide.pdf" );
jse.addFileAttachment( "c:/tmp/wah/DischargeSummaryTemplate.odt" );
jse.addByteArrayAttachment(
    "Hello tis is a byte array 1".getBytes()
    , "ByteArray1.txt", "Byte Array 1" );
jse.addByteArrayAttachment( "Hello tis is a byte array 2".getBytes() );
jse.send();
}
}

```

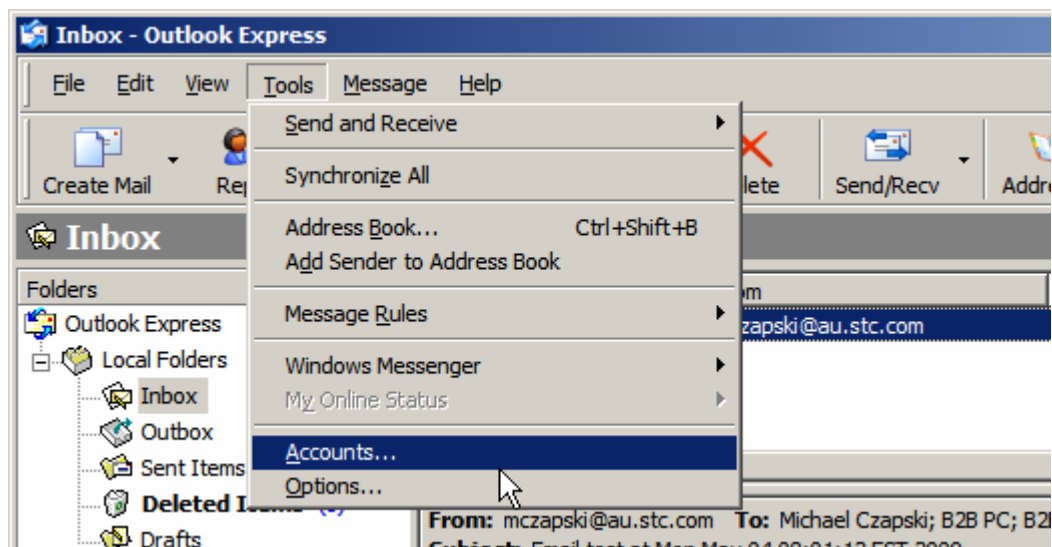
To exercise the project, the export of which is not included but the project is so trivially simple that there should not be a need for it, submit a message to the configured JMS queue.

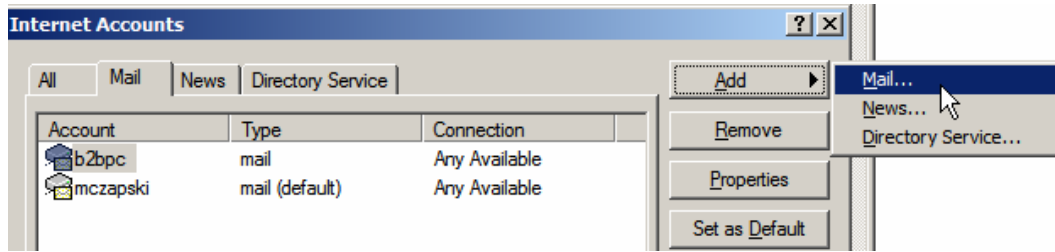
Once the JCD executes an email message will have been sent to the recipient. If all is configured correctly the recipient can use a suitable email client to receive and read the email. Naturally, since the email will likely be encrypted and digitally signed, some steps must be taken at the email client side to enable the email to be readable. The following section, “Using Outlook Express to Read Secure Email”, discusses how Microsoft Outlook Express can be configured to verify correct operation of secure email.

Using Outlook Express to Read Secure Email

Of the multitude of eMail clients available I have chosen Microsoft Outlook Express to use for this discussion. It is included with Windows and it is easy enough to use. If you have/like a different one feel free to use it. I will not help in configuring it, though.

Before we can successfully receive secure mail we must ensure we have recipient’s account set up. Let’s add an account for user mreceiver.





Use the domain name of your mail system, for example mreceiver@vulcan.fed.

Internet Connection Wizard [X]

Internet E-mail Address [Help]

Your e-mail address is the address other people use to send e-mail messages to you.

E-mail address:

For example: someone@microsoft.com

< Back Next > Cancel

Internet Connection Wizard [X]

E-mail Server Names [Help]

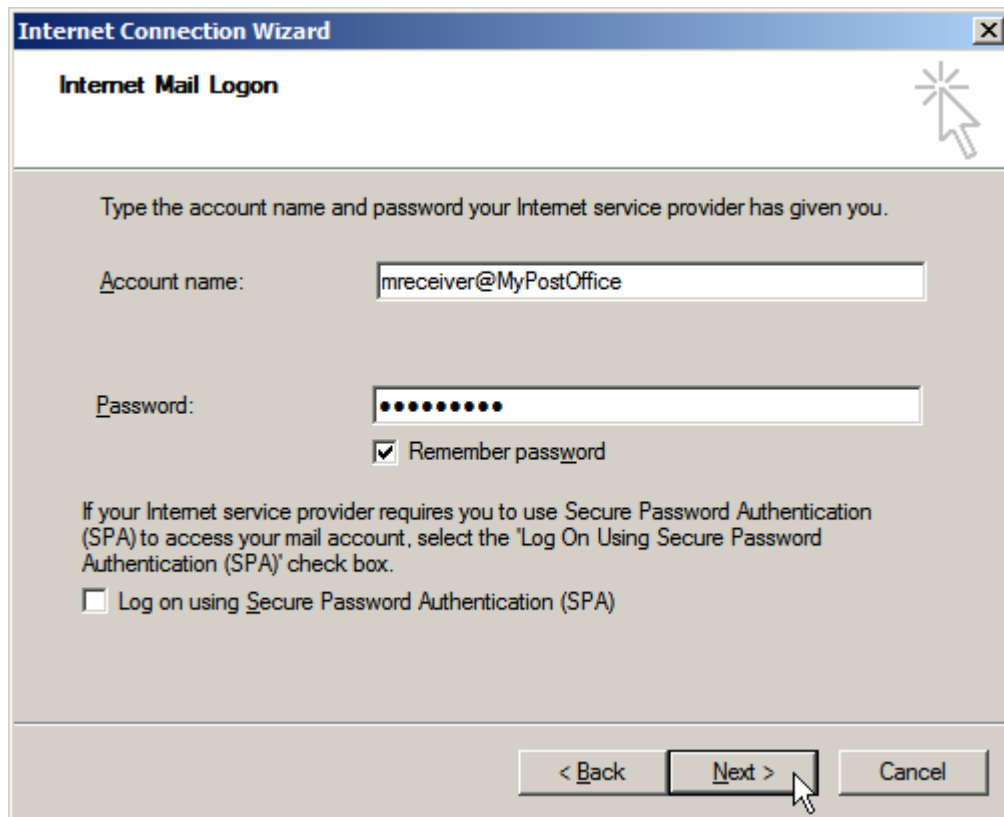
My incoming mail server is a server.

Incoming mail (POP3, IMAP or HTTP) server:

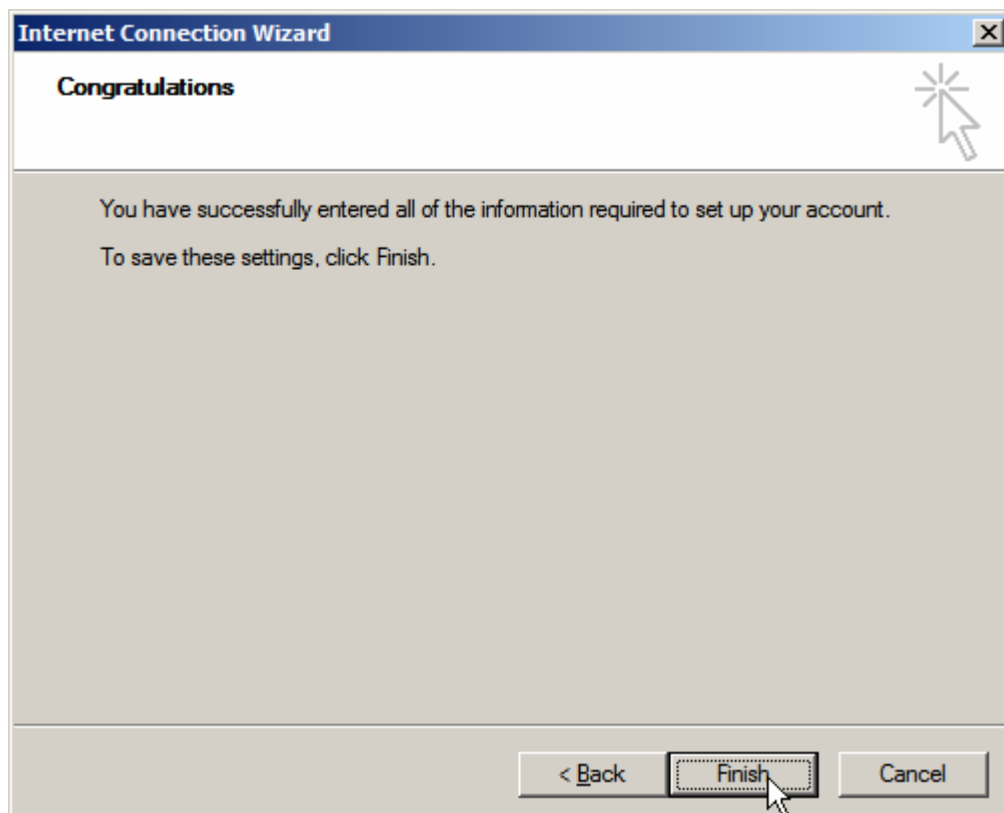
An SMTP server is the server that is used for your outgoing e-mail.

Outgoing mail (SMTP) server:

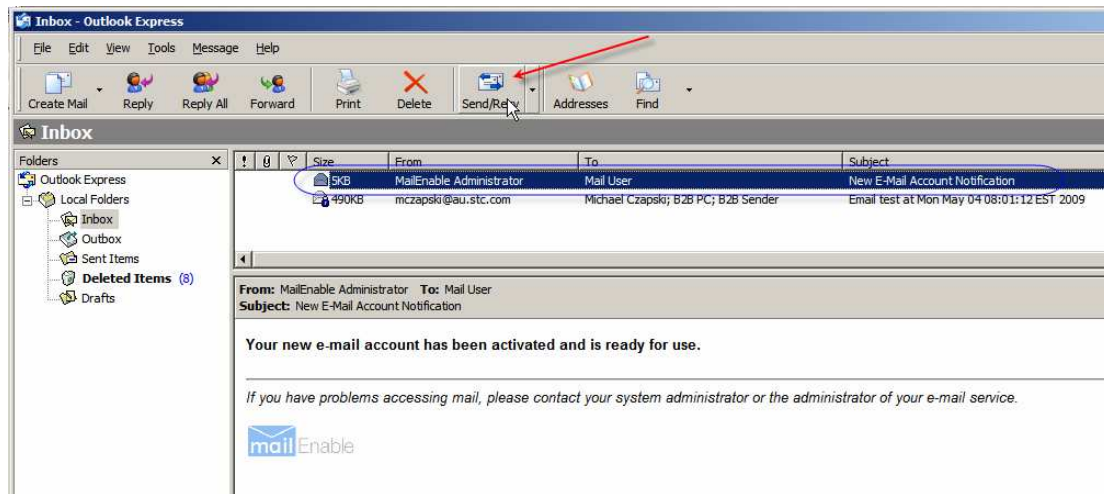
< Back Next > Cancel



The trick here is the mail server configuration. I have MailEnable, <http://www.mailenable.com/>, installed locally and configured with appropriate mailboxes. The mreceiver user has a mailbox with the username of mreceiver and password of mreceiver.

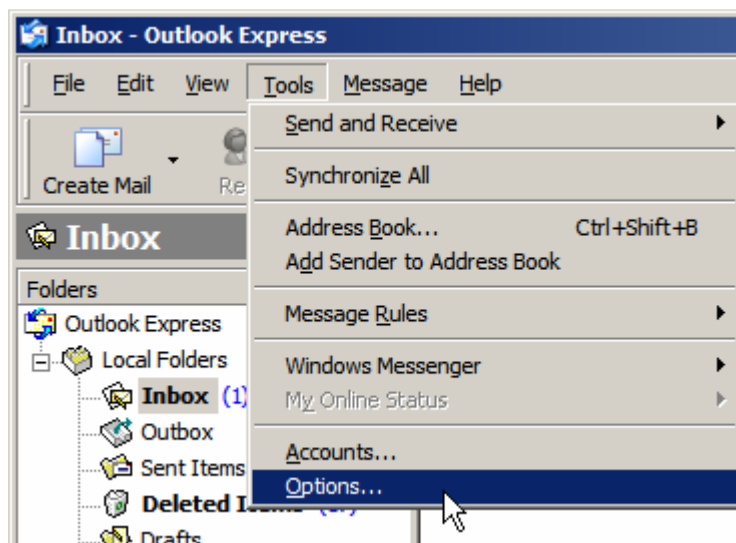


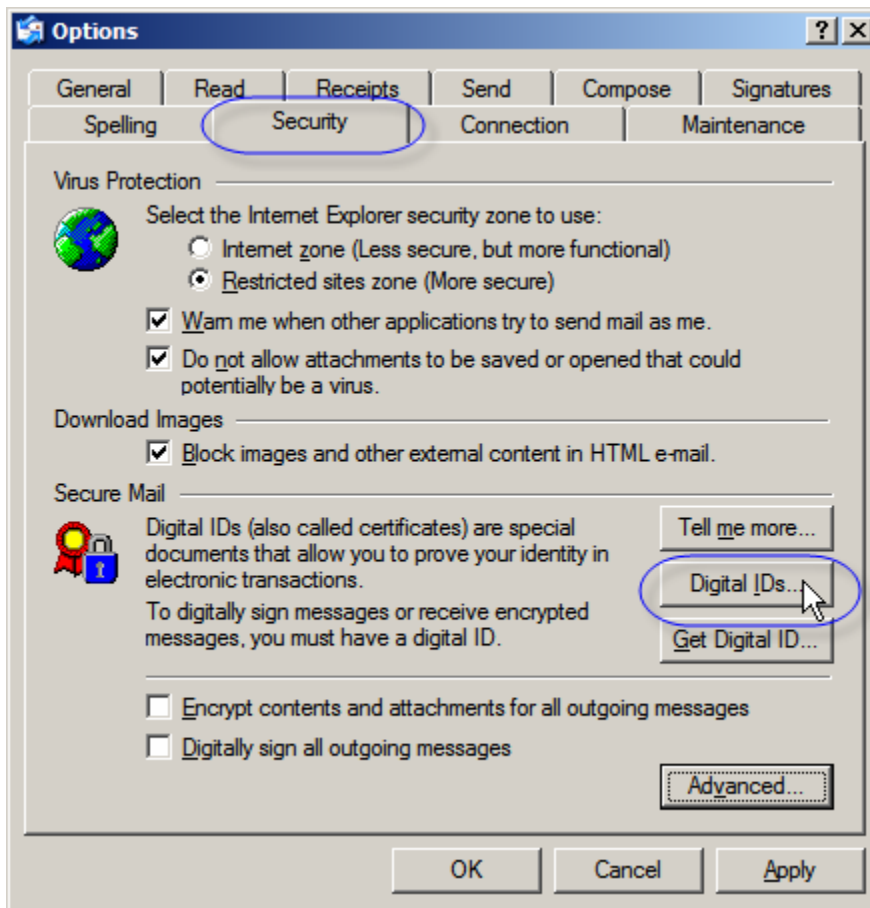
Once the MailEnable account is set up, and the corresponding Outlook Express Account is set up, Send/Receive will get an initial email message from MailEnable.

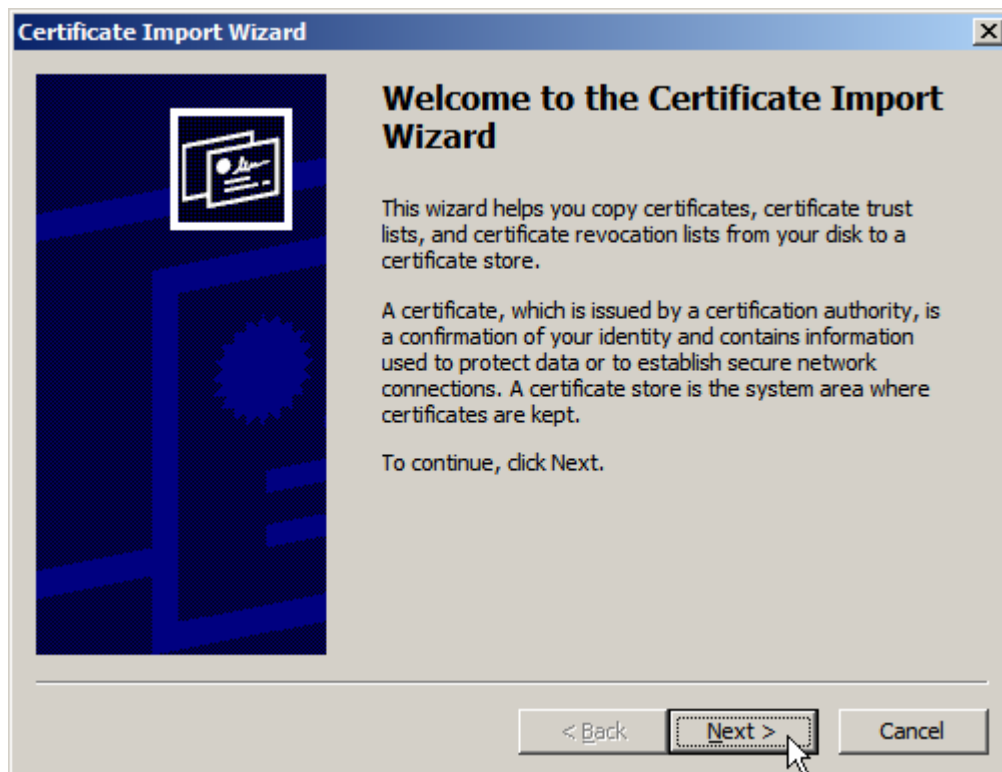
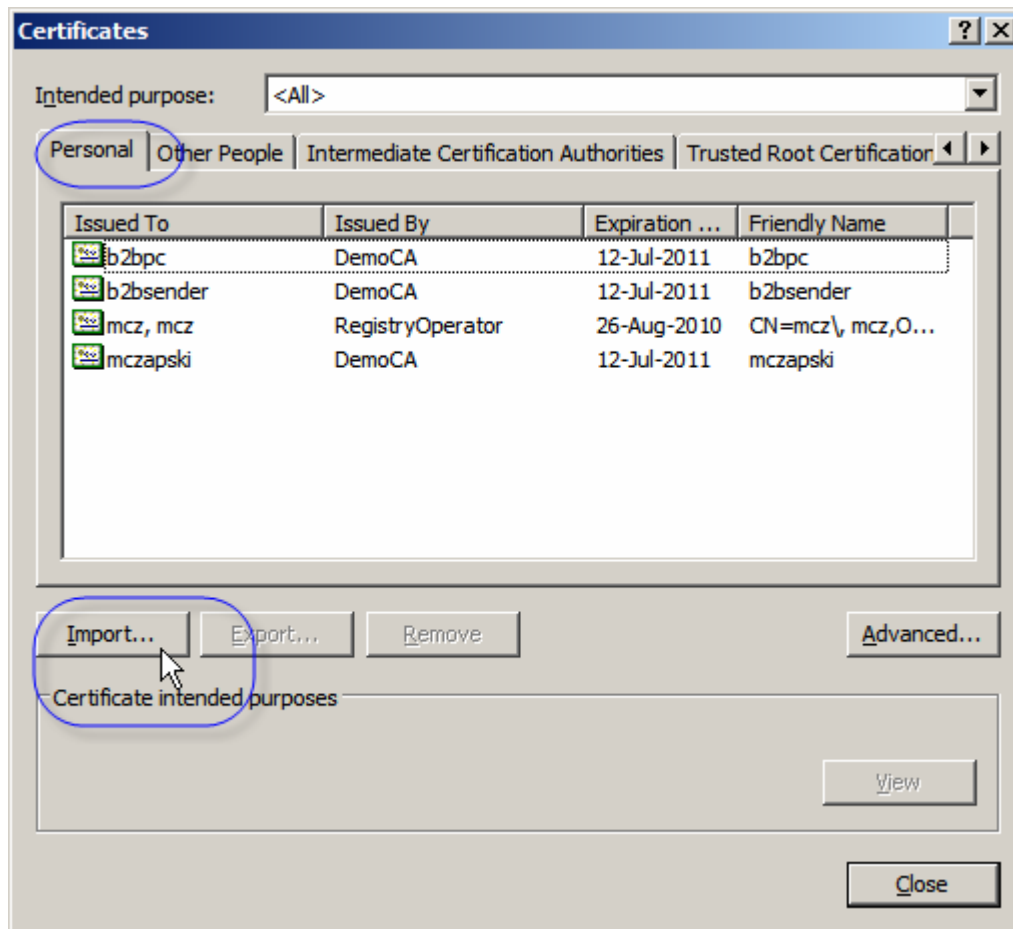


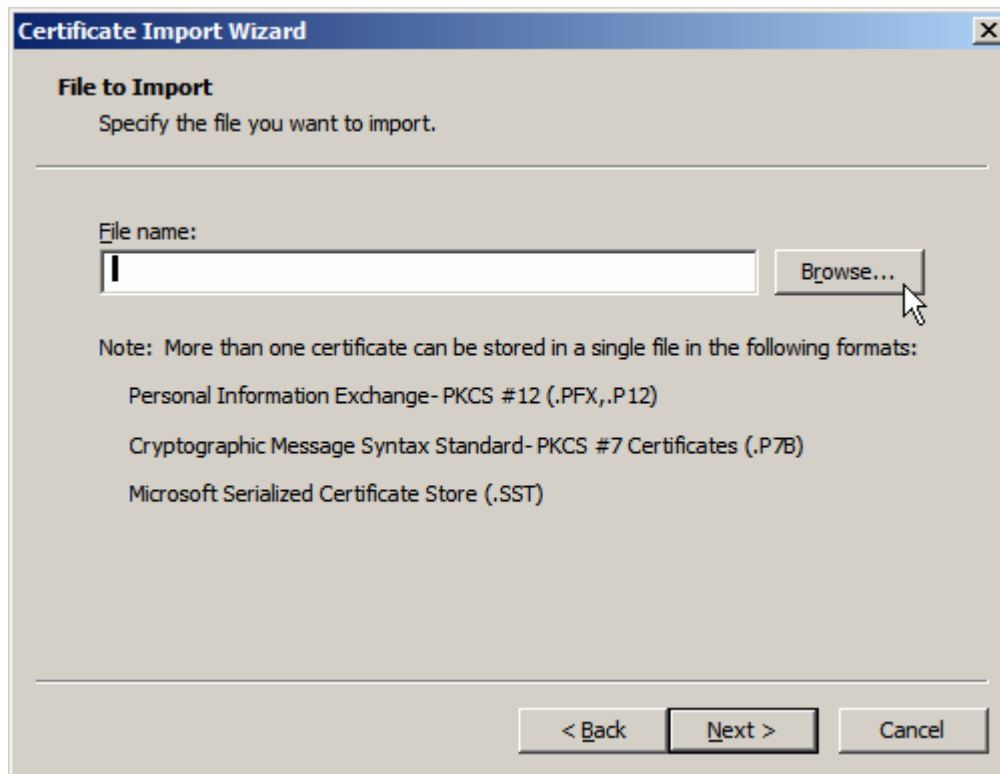
All this gives us a configured mail account for receiving regular electronic mail. This is not going to work for encrypted electronic mail because decryption requires the recipient to know its private key and to tell Outlook Express where to look for it.

Let's add, what Microsoft Outlook Express calls, a Digital ID.



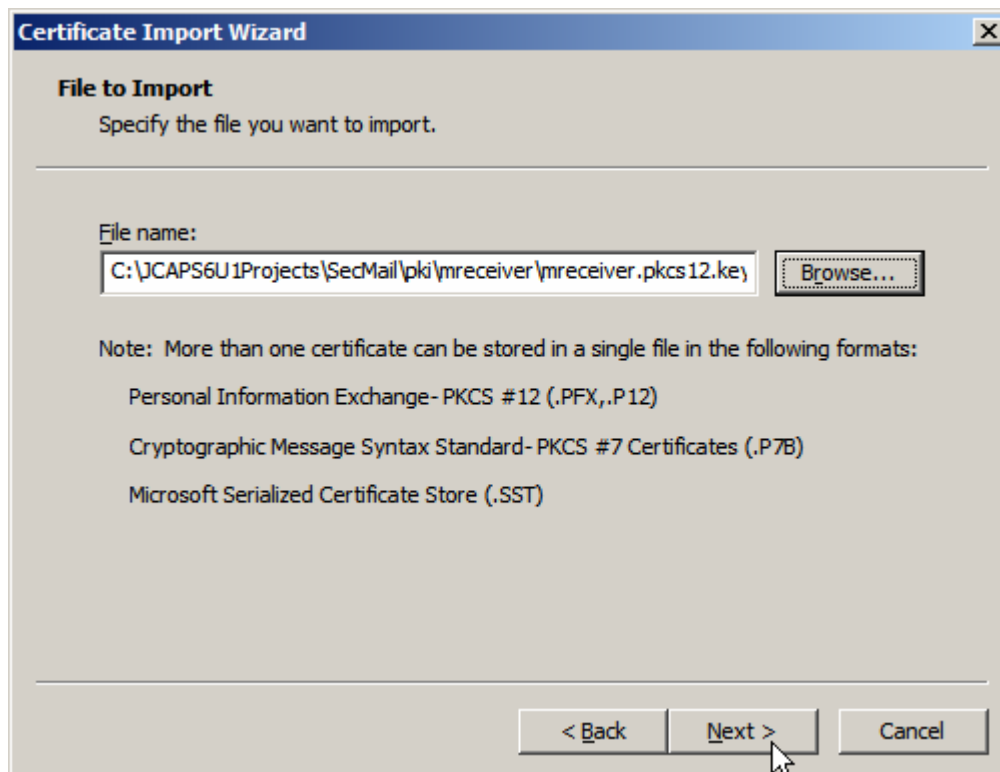




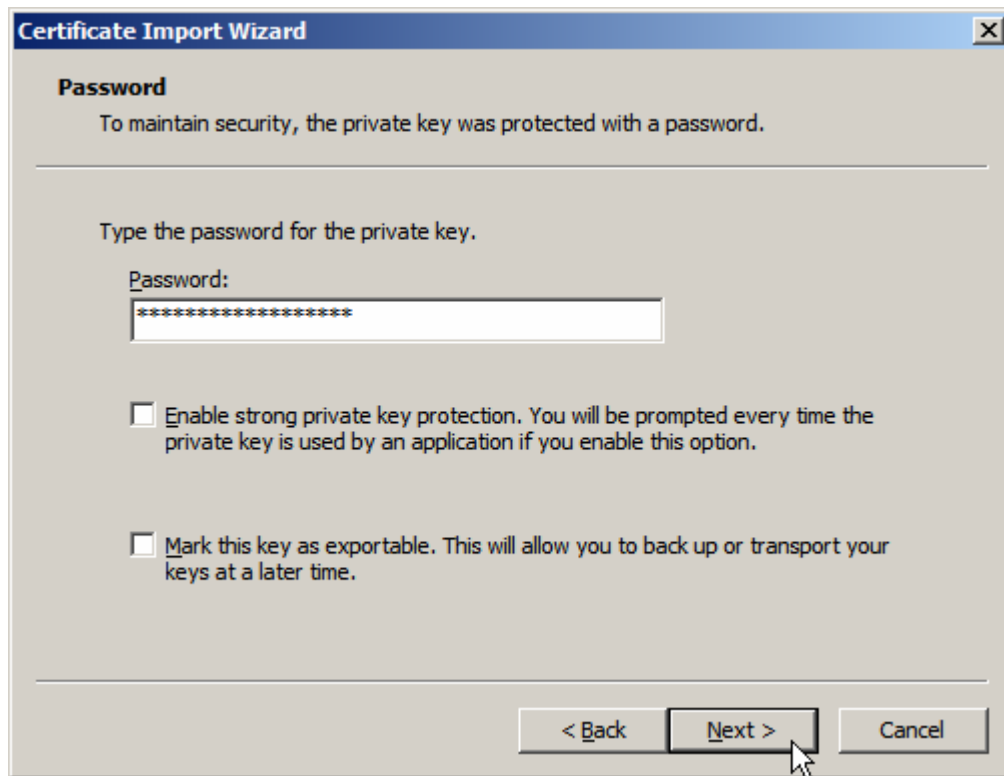


If you followed instructions in the Blog Entry, referenced way back at the beginning of the document, to create cryptographic objects for mreceiver, you will have an object called mreceiver.pkcs12.keystore.p12 with the passphrase of mreceivermreceiver.

Locate that file in the file system and import it.



Enter the passphrase.



Certificate Import Wizard [X]

Password

To maintain security, the private key was protected with a password.

Type the password for the private key.

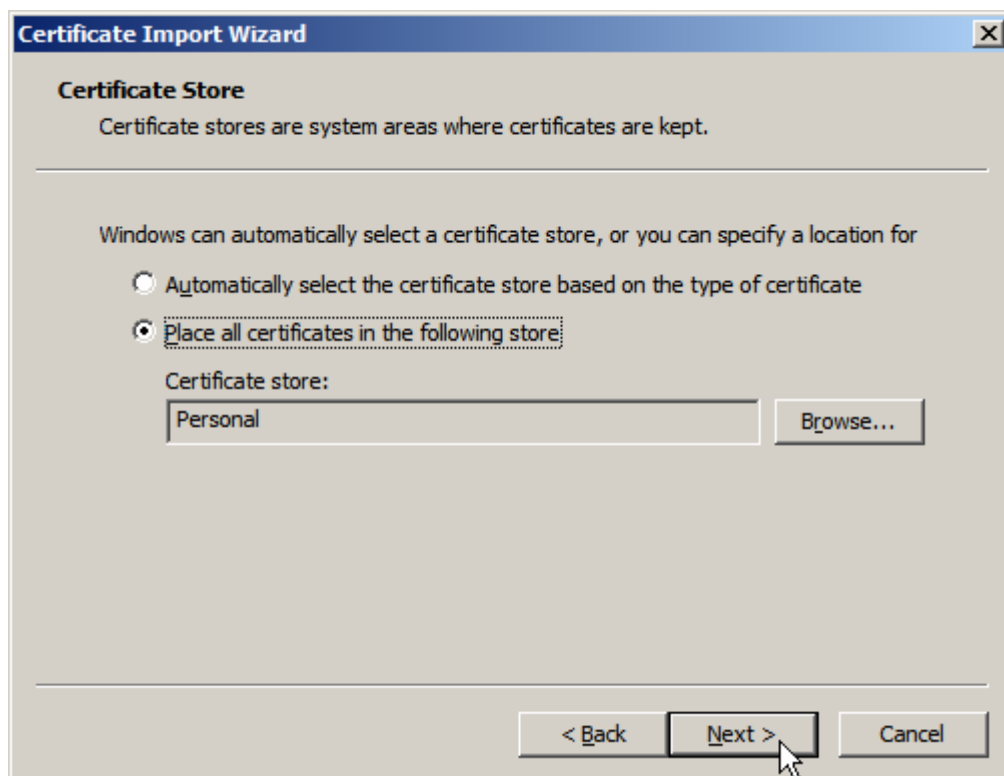
Password:
[*****]

Enable strong private key protection. You will be prompted every time the private key is used by an application if you enable this option.

Mark this key as exportable. This will allow you to back up or transport your keys at a later time.

< Back Next > Cancel

Accept default store.



Certificate Import Wizard [X]

Certificate Store

Certificate stores are system areas where certificates are kept.

Windows can automatically select a certificate store, or you can specify a location for

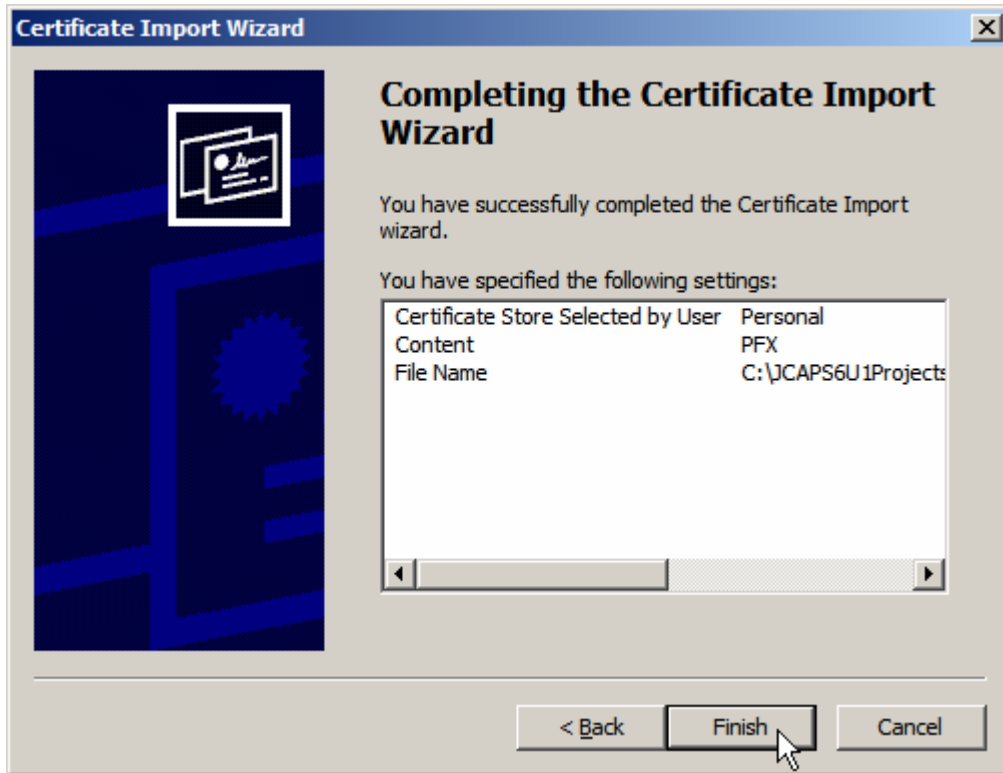
Automatically select the certificate store based on the type of certificate

Place all certificates in the following store:

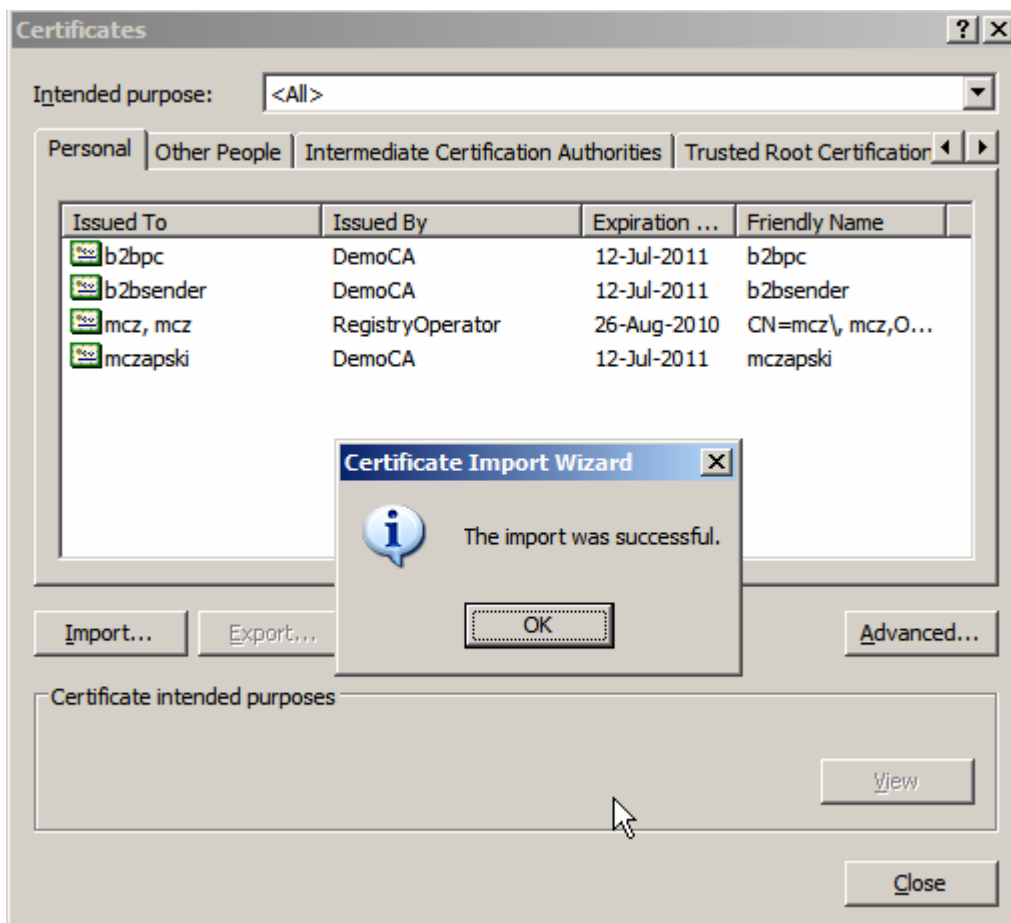
Certificate store:
[Personal] Browse...

< Back Next > Cancel

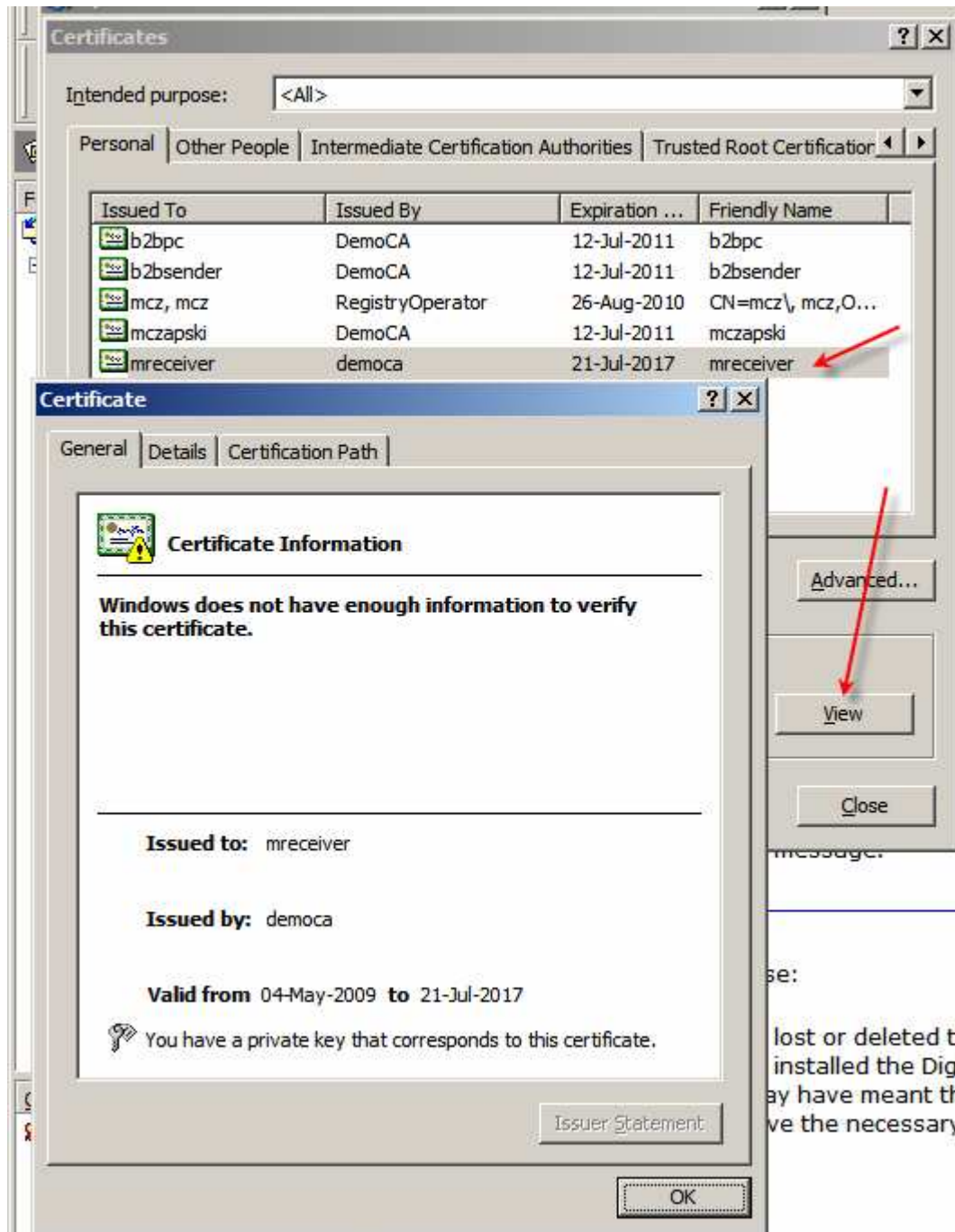
And Finish.



If all went well you will be told that the import was successful.



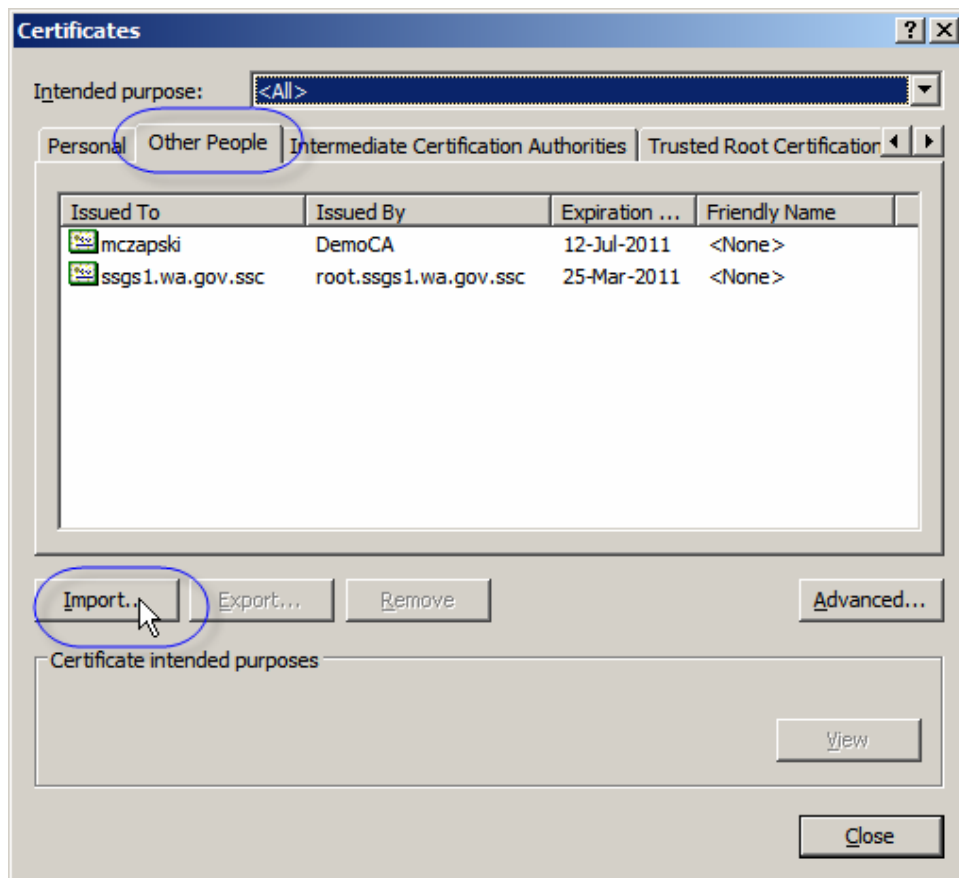
Select the certificate and click View to review the content.



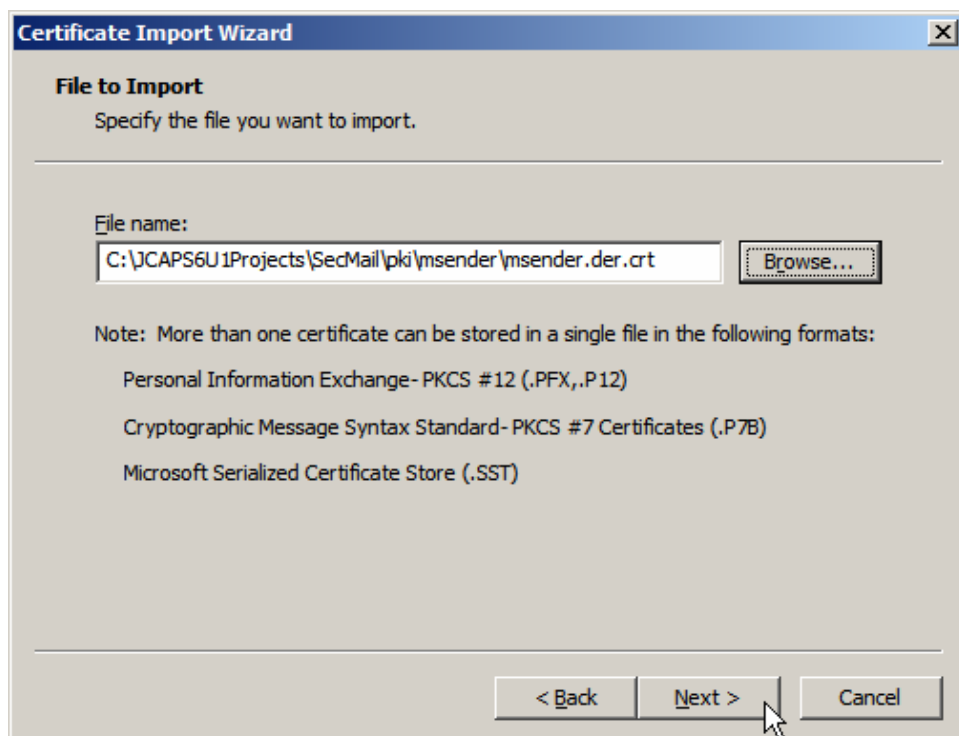
Note that importing the PKCS#12 Keystore, which is what we just did, provides the private key to use for message decryption.

In addition to our private key, necessary to decrypt messages others encrypted for us, we need to import certificates of all senders who will be digitally signing messages we will receive. If we fail to do this the message will be decrypted, if encrypted, but digital signatures will not be valid so we will not be able to trust the messages which came from the party claiming to have sent them.

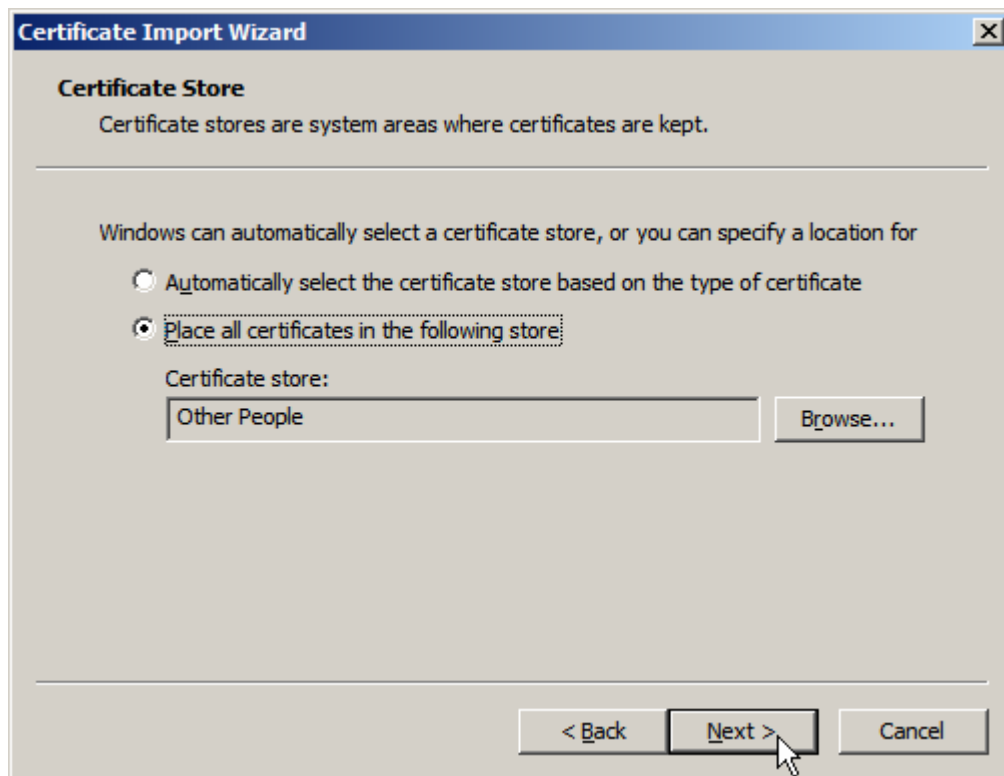
Switch to the Other People tab and click Import.



Locate the other party's X.509 Certificate, we will use msender's certificate, and click Next.



Accept store and click Next.



Click Finish. The certificate will be imported and will be able to be used for verification of digital signatures from msender.

If msender sends a signed and encrypted message to mreceiver Outlook Express will show the message with a person icon over the envelope icon and provide feedback shown below.

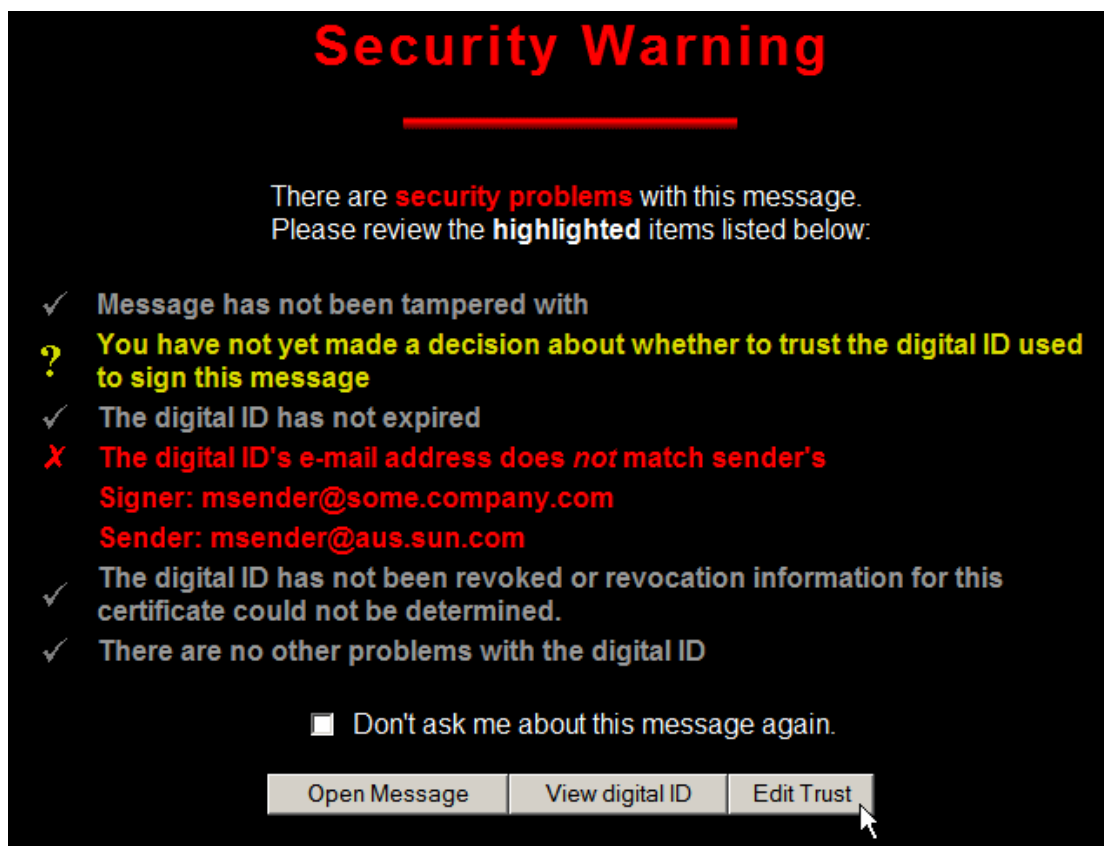


Click Continue.

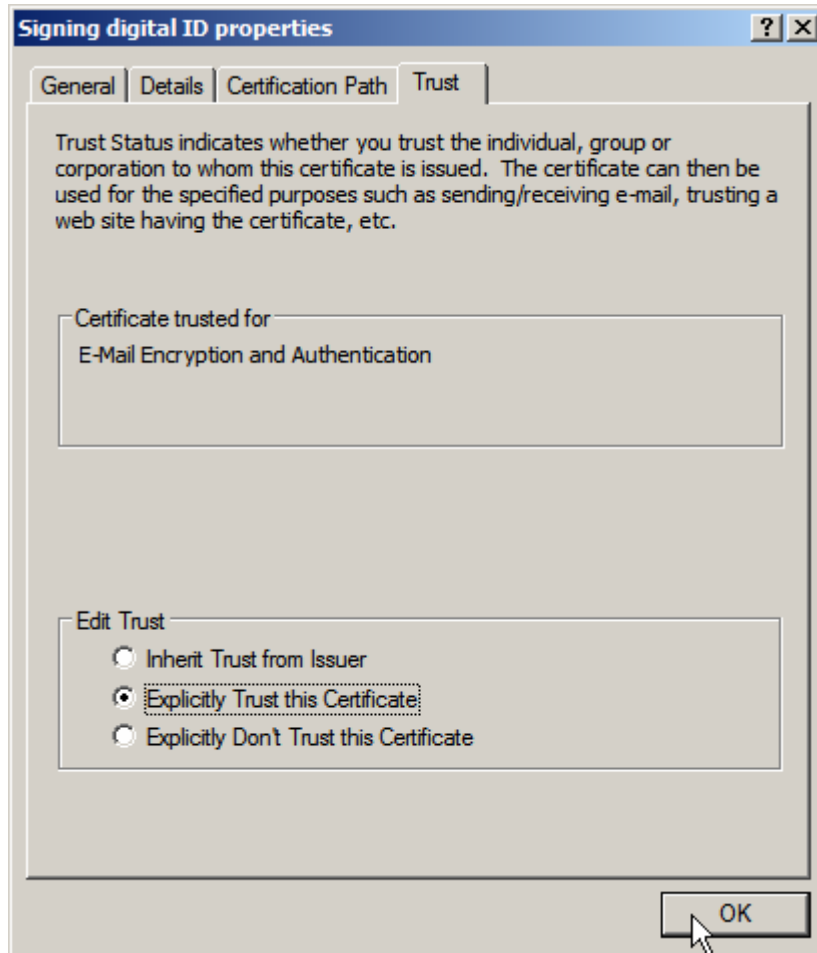
We just imported msender's certificate and we did not tell Outlook Express to trust it. Accordingly, Outlook Express will display a warning page.



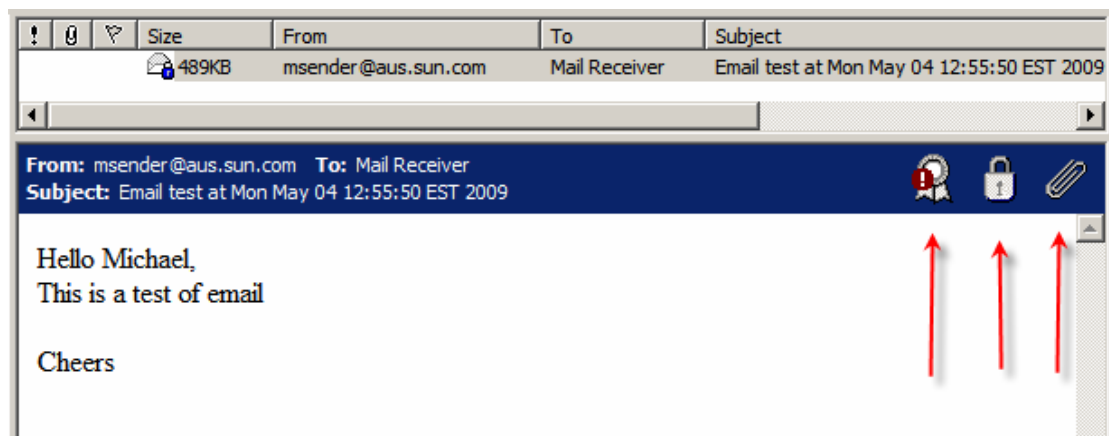
Let's click Edit Trust.



Click the Explicitly Trust this Certificate and click OK.



Back in Outlook Express let's click Open Message.



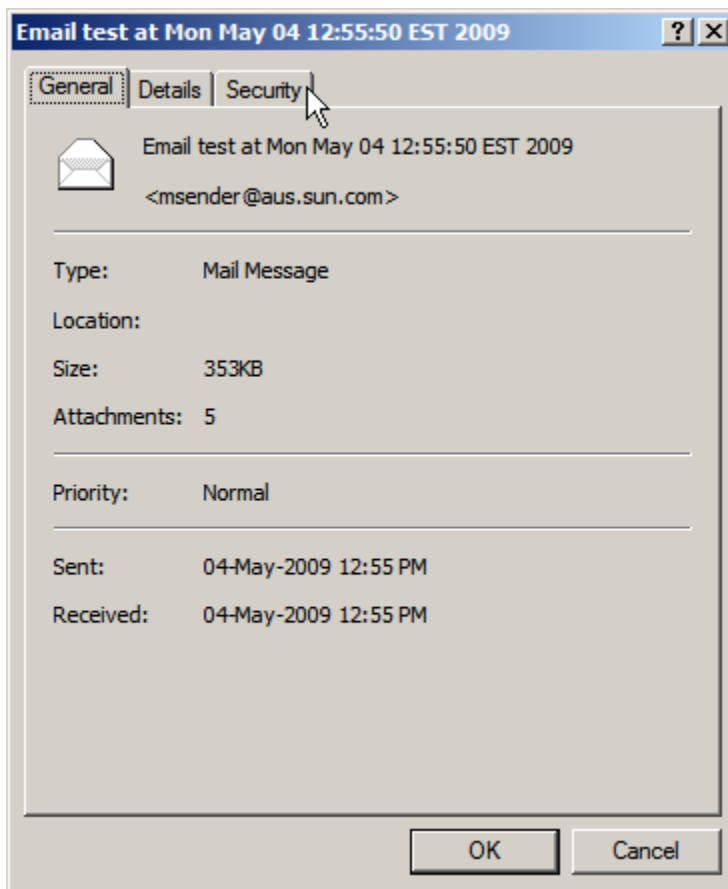
From left to right the arrows point out that the message has been digitally signed (and there is an issue with the signature), it has been encrypted and it has attachments. This is a message in the reading pane. Let's open the message in a window by double-clicking on the message line in the top pane.



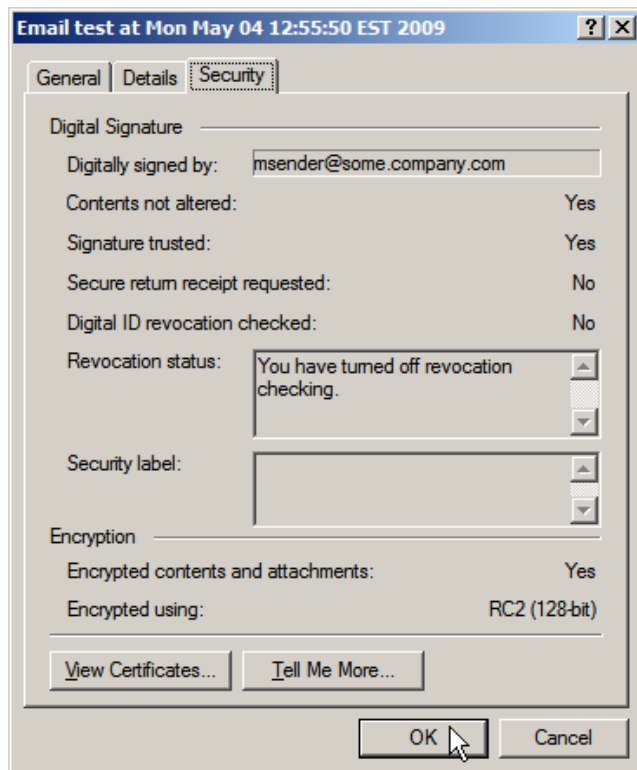
Note Security line: Digitally signed – sender/signer mismatch; Encrypted.
Icons at the far right also indicate that there is an issue with the signature and that message was encrypted.

Let's click on the icon indicating an issue with signing.

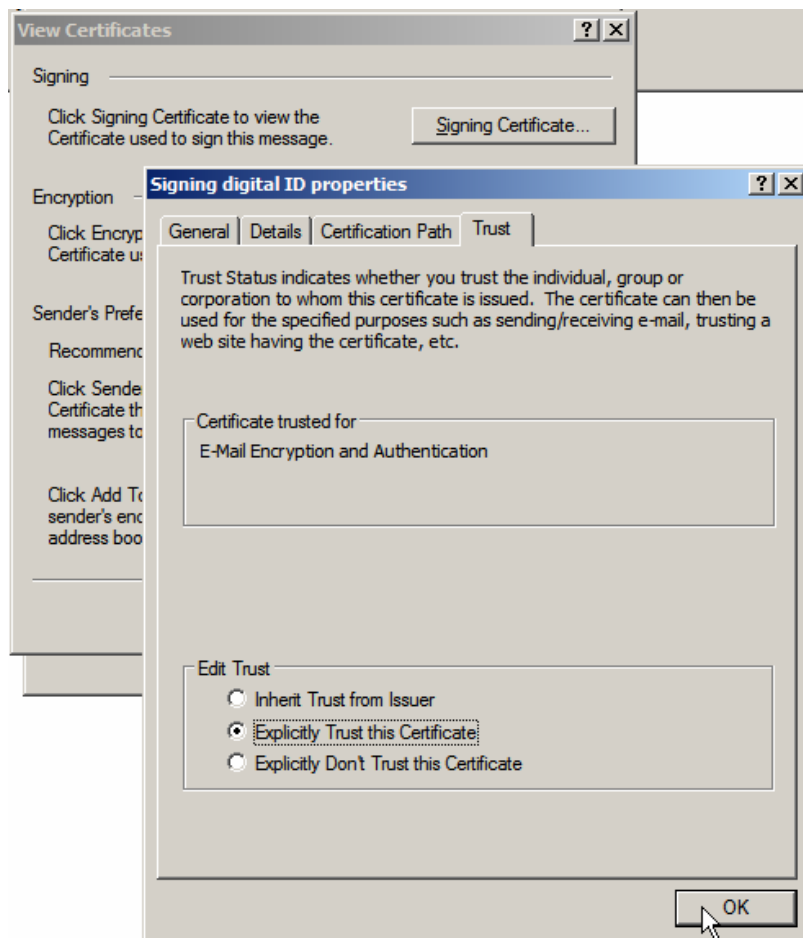
General Tab tells us that the message was received from msender@aus.sun.com (which is how the MailEnable instance I am using is configured).



The Security Tab tells us that the message was signed by msender@some.company.com.



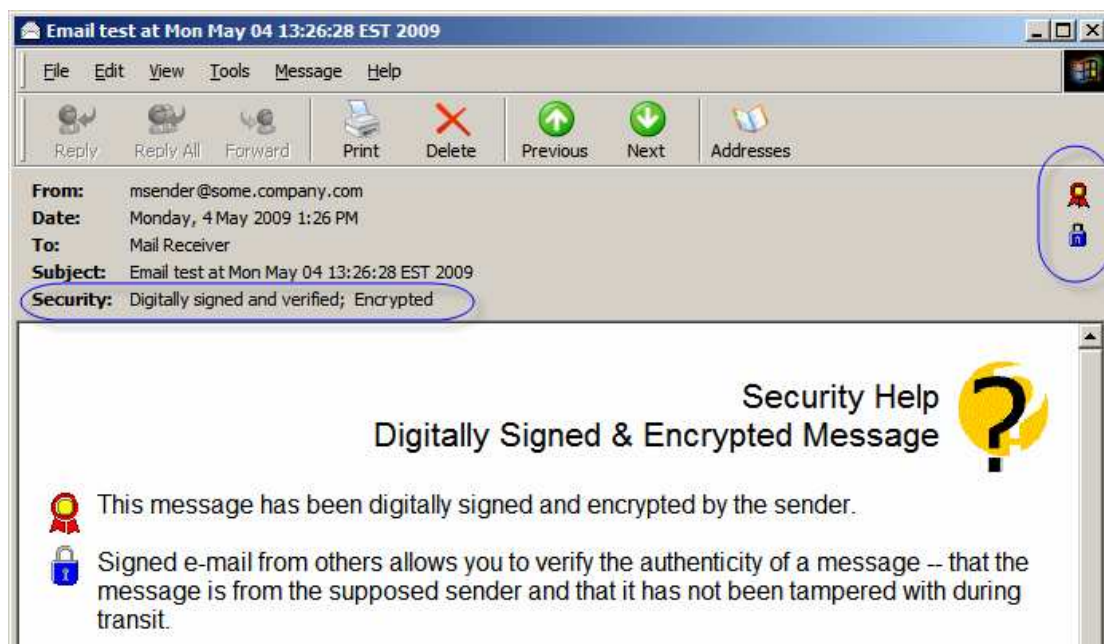
Let's click the View Certificates ... button



Click the Signing Certificate and have a look at the details.

Ideally, the email address in the certificate should match the email address of the sender who is using the certificate to sign electronic mail.

If we make sure the msender's address matches the address in the certificate we will not get the signature validation issue. We will be able to a) trust that nobody but us could possibly have decrypted the message (we have the private key and we did not give it to anyone) and b) the message was signed by the owner of the private key related to their certificate – we can trust the message was composed and signed by the sender.



Summary

Every now and then one needs to secure communications between parties. The issue is complex and expensive to address. The complexity comes from having to configure a bunch of tools to support things like encryption and digital signatures for more than a single party. This discussion introduced a class library that offers a set of simple methods for constructing and sending secure electronic mail using the Secure Multipurpose Internet Mail Extensions (S/MIME), the Bounce Castle Cryptographic Libraries and the Java programming language. The intent was to allow a Java CAPS developer, or a Java developer, to add Secure Electronic Mail functionality quickly and easily, and without having to make too much of a time investment learning about PKI-based security and related matters.