Enhydra-Oyster Project

Enhydra-Oyster Project

Together Teamlösungen EDV-Dienstleistungen GmbH

Elmargasse 2-4
A-1190
Vienna
Austria
+43 (0) 5 04 04 - 122
+43 (0) 5 04 04 - 11 122
<office@together.at>
http://www.together.at/together/index.html

Table of Contents

Chapter 1. Introduction

The Enhydra-Oyster library is project whose main purpose is to provide a consistent way to send secure MIME data via internet standard S/MIME (Secure Multipurpose Internet Mail Extensions). S/MIME adds a level of security to emails by adding digital signatures, or encryption, or both.

The Oyster Library provides the ability to create and send signed as well as encrypted email messages. This library is JavaMail-compliant, allowing you to use S/MIME messages like regular email messages. It represents a complete implementation of S/MIME version 2 (defined in RFC2311, RFC2312, RFC2313, RFC2314, RFC2315), and partial implementation of S/MIME version 3 (defined in RFC2632 and RFC2633).

The S/MIME uses BouncyCastle cryptographic service provider for Java that includes support for RC2, DES DESede3 (TripleDES), RSA and DSA algorithms, which all can be used for S/MIME messages.

Several code examples are included in this distribution, including creation of a signed message, creation of an enveloped (encrypted) message, creation of a signed and enveloped (encrypted) message and creation of an enveloped (encrypted) and signed message.

Chapter 2. Installation

This version of S/MIME implementation is tested and works on Linux and Windows platforms under the JDK version 1.4.x and JDK version 1.5.x.

The following JARs are included in this release:

- oyster.jar - S/MIME implementation
- oyster_tests.jar - S/MIME tests only
- activation.jar - The activation framework (ver 1.0.2)
- mail.jar - JavaMail (ver 1.3.1)
- bcprov-jdk14-136.jar - Bouncy Castle provider and JCE implementation
- xerces.jar - XML parser
- jtidy.jar - HTML parser

All these jar files (except oyster_tests.jar) should be accessible to your applications in order for you to use the Oyster library. You can place them in your Java home directory under jre/lib/ext, or in your CLASSPATH. To use test examples provided with this release, you should also use oyster_tests.jar.

Note that Oyster library should not be used until the original JCE Policy jar files are swapped with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files. The original JDK JCE Policy jar files, are located in JDK under the directory: **<jdk_home>/jre/lib/security**. The Unlimited Strength Policy files are shipped with this release, and can be found in directory: **util/jce**.

The current release of Enhydra-Oyster project exists as source and binary release.

Building Oyster project from source release

Building of Enhydra-Oyster Project structure is completely ant based.

Before building Enhydra-Oyster, few parameters must be predefined. It can be done manually, by editing file "**build.properties**", which should be placed in the project root directory. Also, all those parameters (or just desired subset of parameters) can be given by user at command line.

Invoking "**configure**" command enables editing of the parameters mentioned above. In the project root, the following command should be issued from the command line:

```
configure
```

This command will create build.properties file, if it already does not exist, and set default values for "build.properties" parameters.

For more information refer to **README** file located in the root of Oyster project. Also, you can obtain help by typing:

```
configure -help
```

After the configuration has been done, building process can be started. This can be done by issuing "**make**" command on the command prompt with defined specific option. In the project root, the following command might be issued from the command line:

```
make                  - builds and optimizes Enhydra-Oyster with javadoc
                        and docbook documentation build (equals to
                        buildAll option).
make help             - displays this screen.
make install          - builds, copies and configures Enhydra-Oyster and
                        configures JDK JCE policy.
make buildAll         - builds and optimizes Enhydra-Oyster with javadoc
                        and docbook documentation build.
make buildNoDoc       - builds and configures Enhydra-Oyster without
                        documentation building.
make buildDoc         - builds Enhydra-Oyster documentation only.
make distributions    - builds and configures Enhydra-Oyster with javadoc
                        and docbook documentation and creates optimized
                        distribution.
make clean            - removes the output folder (in order to start a
                        new compilation from scratch)
make configure        - configures the Enhydra-Oyster and JDK JCE policy.
make restore          - restores JDK JCE policy to default value from
                        backup.
```

Option "**configure**" swaps original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files. The original JCE Policy jar files are backuped and can be restored by invoking "**restore**" option.

Swapping and backup of Policy jar files are also performed within "**install**" option invoke.

For more information refer to **README** file located in root of Oyster project.

Note that if more than one JKD version (run-time or/and source development) is installed on computer, automatic swapping of original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files can fail so it must be done manually (in appropriate JDK version - runtime or/and source development).

The following exception:

```
java.lang.SecurityException: Unsupported keysize or algorithm parameters
        at javax.crypto.Cipher.init(DashoA6275)
```

means that swapping of original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files has not been done at all, or hasn't been done in the appropriate JDK version.

Configuring Oyster project from binary release

Unpacked Oyster project files from binary release can be free copied to any location. To enable library usage, command "configure" must be invoked on the project root directory command line to set Unlimited Strength Policy Files.

Command "**configure**" swaps original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files. The original JCE Policy jar files are backuped and can be restored by invoking "**restore**" command.

For more information refer to README file located in root of Oyster project. Also, you can obtain help by typing:

```
configure -help
```

Note that if more than one JKD version (run-time or/and source development) is installed on computer, automatic swapping of original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files can fail so it must be done manually (in appropriate JDK version - runtime or/and source development).

The following exception:

```
java.lang.SecurityException: Unsupported keysize or algorithm parameters
        at javax.crypto.Cipher.init(DashoA6275)
```

means that swapping of original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files has not been done at all, or hasn't been done in the appropriate JDK version.

Chapter 3. Enhydra-Oyster Classes

The classes that you will need to use to create and send S/MIME messages are all in the **org.enhydra.oyster.smime** package. They are:

- EnvelopedSMIME
- SignedSMIME
- SignedAndEnvelopedSMIME

Detail explanation and tutorial for handling these classes can be found in java documentation supported within this project.

EnvelopedSMIME class

This is the class for creating encrypted S/MIME message.

Encrypting options:

- Encryption can be performed by the following algorithms and corresponding key sizes: RC2_CBC with 40 bits, RC2_CBC with 64 bits, RC2_CBC with 128 bits, DES with 56 bits, DES_EDE3_CBC with 128 bits or DES_EDE3_CBC with 192 bits. Many of email clients can deal only with RC2_CBC with 40 bits and DES with 56 bits (weak encryption).
- One message can be encrypted and sent to one or more recipients.

SignedSMIME class

This is the class for creating signed S/MIME message.

Signing options:

- SMIME message can be signed by internal (implicit) or external (explicit) signing.
- Message can be signed with or without Signed Attributes.
- Signed algorithm can be: SHA1 with RSA, MD2 with RSA, MD5 with RSA or SHA1 with DSA.
- Capabilities Attributes of signed message can be set (optional).

- Defining the Capabilities Attributes can be from group: RC2 40, RC2 64, RC2 128, DES, DES EDE3 for symmetric algorithm, from group: SHA1 with RSA, MD2 with RSA, MD5 with RSA or SHA1 with DSA for signed algorithm, and RSA for encipher algorithm.
- Signers' Certificates and their root authorities can be included in signed messages (optional).
- More then one signer can sign the massage, and it can be done by using the same or different algorithms of signing.

SignedAndEnvelopedSMIME class

This is the class for creating signed and encrypted S/MIME message and encrypted and signed S/MIME message. All options for signed or encrypted message, mentioned before, can also be implemented here with one exception: external signing is not used here. External signing allows email receiving clients without implemented SMIME capabilities to preview the signed SMIME email messages. These possibilities have no importance in the case of both signing and enveloping (or enveloping and signing) because, before or after signing, the message is encrypted, so it is not readable.

Chapter 4. Message types

In this release, the content of an email message can be composed as:

- text/plain (only text without any formatting) or
- text/html (html coded view of the message)

One or more attachments can be added to each email. Also, message can only consist of attachments without any content, or opposite, can consist of content without any attachment. It is advisable that in html composed content, images used for backgrounds or in IMG html tags should be jpg or gif type. Other image formats can make problems with its visibility in particular email clients. For example, Netscape Messenger 4.79 will crush down if other formats (different than gif or jpg) are used.

Chapter 5. Bouncy Castle JCE Provider

The BC (Bouncy Castle) JCE Provider provides support for various cryptographic algorithms, intended primary to support S/MIME signing and encrypting.

For complete information refer to http://www.bouncycastle.org. Brief look of one part of supported cryptographic algorithms and possibilities can be viewed in following lines:

Symmetric (Block) algorithms:

AES, AESWrap, Blowfish, CAST5, CAST6, DES, DESede, IDEA, RC2, RC5, RC5-64, RC6, Rijndael, Skipjack, Twofish, Serpent

Padding Schemes:

No padding, PKCS5/7, ISO10126/ISO10126-2, X9.23/X923, CTS

Modes: ECB, CBC, OFB(n), CFB(n), PGPCFB

where (n) is a multiple of 8 that gives the block size in bits, e.g., OFB8. Note that OFB and CFB mode can be used with plain text that is not an exact multiple of the block size if NoPadding has been specified.

Symmetric (Stream) algorithm:

RC4

Block Asymmetric algorithm:

RSA

Encoding: OAEP - Optimal Asymmetric Encryption Padding, PCKS1 - PKCS v1.5 Padding, ISO9796-1 - ISO9796-1 edition 1 Padding

Key Agreement algorithms:

Diffie-Hellman key agreement is supported by using the "DH", "ECDH", and "ECDHC" (ECDH with cofactors) key agreement instances.

Note: with basic "DH" only the basic algorithm fits in with the JCE API, if you're using long-term public keys you may want to look at the light-weight API.

Digest algorithms:

MD2, MD4, MD5, RipeMD128, RipeMD160, RipeMD256, RipeMD320, SHA1, SHA-256, SHA-384, SHA-512, Tiger

Signature algorithms:

MD2withRSA, MD5withRSA, SHA1withRSA, RIPEMD160withRSA, SHA1withDSA, SHA1withECDSA

Certificates and KeyStore:

The Bouncy Castle provider will read X.509 certificates (v2 or v3). They can be provided either in the normal PEM encoded format, or as DER binaries. The CertificiateFactory will also read X.509 CRLs

(v2) from either PEM or DER encoding.

The Bouncy Castle package has three implementation of a keystore:

- BKS
- Keystore.BouncyCastle, or Keystore.UBER and
- PKCS12 compatabile keystore.

Chapter 6. Example applications

If build of Oyster project was successful, and configuration of project was performed properly, the examples can be started. All examples provided within Enhydra-Oyster project are placed in oyster_tests.jar file, in package org.enhydra.oyster.test. The examples should be invoked from "examples" directory by command files whose extension depends on operating system (.bat on Windows platform or .sh on Linux platform). In order to simplify further explanation, in the following text, called command files will be represented only with their names (which are the same for both operation systems). Note that, specially for Linux, extension should be included in command file calls.

"examples" directory contains, beside command files, separate private key and public key file storages (.pfx and .cer files) created for example purpose, as well as other necessary files used for example. Also, there is a file named 'keystore.ks' which contains all keys and certificate chains stored together in 'BKS' type KeyStore file. Specific key can be obtained from 'keystore.ks' by referencing to its alias, which is the same as the name of corresponding separate .pfx file.

Note: It is assumed that Java1.4 is already installed on your computer, and that you have access to email server.

Four basic simple test examples are provided:

- TestEncrypt.java
- TestSigned.java
- TestSigEnc.java and
- TestEncSig.java

along with twelve more tests:

- TestEncryptHtml.java,
- TestEncryptGeneratedHtml.java,
- TestEncryptKeyStore.java
- TestSignedHtml.java,
- TestSignedGeneratedHtml.java,
- TestSignedKeyStore.java
- TestSigEncHtml.java,
- TestSigEncGeneratedHtml.java,
- TestSigEncKeyStore.java
- TestEncSigHtml.java and,
- TestEncSigGeneratedHtml.java

- TestEncSigKeyStore.java

List of arguments with their short descriptions, used in examples TestEncrypt, TestEncryptHtml, TestEncryptGeneratedHtml and TestEncryptKeyStore:

- Arg0 - SMTP host name
- Arg1 - name of recipient's email address
- Arg2 - name of recipient's .cer file (or its alias name in KeyStore test)
- Arg3 - symmetric algorithm name
- Arg4 - path and name for attachment (optional)

List of arguments with their short descriptions, used in examples TestSigned, TestSignedHtml, TestSignedGeneratedHtml and TestSignedKeyStore:

- Arg0 - SMTP host name
- Arg1 - name of recipient's email address
- Arg2 - signed algorithm name
- Arg3 - inclusion of certificates; true/false
- Arg4 - inclusion of signed attributes; true/false
- Arg5 - name of sender's .pfx file (or its or alias name in KeyStore test)
- Arg6 - external signing; true/false
- Arg7 - path and name for attachment (optional)

List of arguments with their short descriptions, used in examples TestSigEnc TestSigEncHtml, TestSigEncGeneratedHtml, TestSigEncKeyStore and TestEncSig, TestEncSigHtml, TestEncSigGeneratedHtml, TestEncSigKeyStore:

- Arg0 - SMTP host name
- Arg1 - name of recipient's email address
- Arg2 - name of recipient's .cer file (or its or alias name in KeyStore test)
- Arg3 - symmetric algorithm name
- Arg4 - signed algorithm name
- Arg5 - inclusion of certificates; true/false
- Arg6 - inclusion of signed attributes; true/false
- Arg7 - name of sender's .pfx file (or its or alias name in KeyStore tests)

- Arg8 - path and name for attachment (optional)

Unfortunately, it's a little tricky to verify the emails that the examples send out. To verify that the messages are signed correctly, you need to tell your mail client to trust the signing certificate (in case that signed message contains sender's certificate), or to import sender's certificate to email client "manually". To decrypt the encrypted message you need to import recipient's certificate and his private key (stored in files with .pfx or .p12 extension).

You should run the examples in listed order, as the verification steps require.

Encrypting Test

First, run encrypting test (TestEncrypt) using the command file enc.bat / enc.sh (win/linux) with 4 or 5 arguments:

```
enc <arg0> <arg1> <arg2> <arg3>
enc <arg0> <arg1> <arg2> <arg3> <arg4>
```

Examples:

```
enc together.at recipient@together.at recipient512.cer RC240
enc together.at recipient@together.at recipient512.cer RC240 ./test/Zip8Test.zip
```

The same syntax can be used for running other tests from group of enveloped tests. They are started by using corresponding command files: enc_html.bat / enc_html.sh (for TestEncryptHtml), enc_gen_html.bat / enc_gen_html.sh (for TestEncryptGeneratedHtml) and enc_kstore.bat / enc_kstore.sh (for TestEncryptKeyStore).

Note: that TestEncryptGeneratedHtml.java always has attachment no matter if his command file starts with 4 or 5 parameters.

Go back to Outlook and retrieve the message. At this point you won't be able to view the message since you don't have the private key corresponding to the public key used to encrypt the message, so you'll need to import the recipient512.pfx included in this distribution.

Importing recipient's private key to Outlook 2000:

If you are using Outlook 2000, go to "Tools", and choose "Options" from the menu. Select the "Security" tab and click on "Import/Export Digital ID". Select "Import existing Digital ID from file". Browse to the recipients512.pfx file for Import file. Enter "together" for the password and "recipient" for Digital ID name. Click on "OK". Select "Yes" to add the Together Root CA to your trusted root list. Close all the dialog boxes, select a different message, and then select the enveloped message. You should be able to view it now.

Importing recipient's private key to Internet Explorer:

From Internet Explorer, go to "Tools", and choose "Options" from the menu. Select the "Content" tab and click on "Certificates". Click on "Import" and then "Next". Browse to the recipients512.pfx file and click on "Next". Enter "together" for the password and click on "Next". Select "Automatically select the

certificate store based on the type of certificate" and once again click on "Next". Finally, click on "Finish". Select "Yes" to add the Together Root CA to your trusted root list. Go back to Outlook and then select the enveloped message. You should be able to view it now.

Importing recipient's private key to Netscape Messenger:

Click on the lock in the bottom left corner and then click on Certificates: "Yours". Click on "Import a Certificate", enter password (to protect private key in Communicator Certificate DB), and browse to the recipient512.pfx file. Enter "together" without the quotes as the password used to protect the data. A certificate named "Recipient" should appear in the list. Click on "Signers", then find in the list of root certificates, name "Together Root CA". Click on "Edit" and check all fields, and then click "OK". Click then "Verify", and click "OK". Go back to "Yours", select on Certificate tag "Recipient" and click "Verify". Dismiss the dialog by clicking "OK", select a different message, and then select the enveloped message. You should be able to view it now. Netscape will automatically add the Together Root CA to your trusted root list, even though it is not strictly necessary. If you don't want to trust all certificates issued by the Together Root CA automatically, you should remove it from the list found in Certificates tag.

Signing Test

Run signing test (TestSigned) using the command file sig.bat / sig.sh (win/linux) with 7 or 8 arguments:

```
sig <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6>

sig <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6> <arg7>
```

Examples:

```
sig together.at recipient@together.at SHA1_WITH_RSA true true sender512.pfx true

sig together.at recipient@together.at SHA1_WITH_RSA true true sender512.pfx false
./test/Zip8Test.zip
```

The same syntax can be used for running other tests from group of signed tests. They are started by using corresponding command files: sig_html.bat / sig_html.sh (for TestSignedHtml), sig_gen_html.bat / sig_gen_html.sh (for TestSignedGeneratedHtml) and sig_kstore.bat / sig_kstore.sh (for TestSignedKeyStore).

Note: that TestSignedGeneratedHtml.java always has attachment no matter if his command file starts with 7 or 8 parameters. Also, note that 'Key Store' password is 'together' (same as for .pfx files). The aliases ,used to mark 'Private Keyes' in 'Key Store', have values same as names of corresponding .pfx files.

If you follow the instructions above, you should be able to view the email sent by TestSigned (sig command file) with no further modifications.

You can view this message because same Root CA issues both certificates: one you have already imported (recipient's certificate) and other, which is used for signing this message (sender's certificate).

If you use a certificate with different Root CA for signing message, or in process of invoking recipient's certificates you choose not to add the Together Root CA to your trusted root list, and use Outlook to get the message, initially, the signature will be marked as invalid in Outlook even though the signature is cryptographically valid since the certificate chain is not trusted.

You should get a security warning telling you that you haven't decided yet if you wish to trust the digital

ID. Click on "Edit Trust" and select "Explicitly Trust this Certificate". You'll need to select YES on the following dialog to give Windows permission to add the certificate. Click on a different message and then on the signed message again to force a refresh. Outlook should then show the signature as valid.

Signing and Encrypting test

Run signing and encrypting test (TestSigEnc) using the command file sig_enc.bat / sig_enc.sh (win/linux) with 8 or 9 arguments (first operation is signing and then follows encrypting of the message):

```
sig_enc <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6> <arg7>

sig_enc <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6> <arg7> <arg8>
```

Examples:

```
sig_enc together.at recipient@together.at recipient512.cer RC240 SHA1_WITH_RSA true true
sender512.pfx

sig_enc together.at recipient@together.at recipient512.cer RC240 SHA1_WITH_RSA true true
sender512.pfx ./test/Zip8Test.zip
```

If you have followed the instructions above you should be able to view the email sent by sig_enc.bat with no further modifications.

The same syntax can be used for running other tests from group of signed and enveloped tests. They are started by using corresponding command files: sig_enc_html.bat / sig_enc_html.sh (for TestSigEncHtml), sig_enc_gen_html.bat / sig_enc_gen_html.sh (for TestSigEncGeneratedHtml) and sig_enc_kstore.bat / sig_enc_kstore.sh (for TestSigEncKeyStore).

Note: that TestSigEncGeneratedHtml always has attachment no matter if his command file starts with 8 or 9 para

Encrypting and Signing Test

Run encrypting and signing test (TestEncSig) using the command file enc_sig.bat/enc_sig.sh (win/linux) with 8 or 9 arguments (first operation is encrypting and then follows signing of the message):

```
enc_sig <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6> <arg7>

enc_sig <arg0> <arg1> <arg2> <arg3> <arg4> <arg5> <arg6> <arg7> <arg8>
```

Examples:

```
enc_sig together.at recipient@together.at recipient512.cer RC240 SHA1_WITH_RSA true true
sender512.pfx

enc_sig together.at recipient@together.at recipient512.cer RC240 SHA1_WITH_RSA true true
```

```
sender512.pfx ./test/Zip8Test.zip
```

If you have followed the instructions above you should be able to view the email sent by enc_sig.bat with no further modifications.

The same syntax can be used for running other tests from group of enveloped and signed tests. They are started by using corresponding command files: enc_sig_html.bat / enc_sig_html.sh (for TestEncSigHtml), enc_sig_gen_html.bat / enc_sig_gen_html.sh (for TestEncSigGeneratedHtml) and enc_sig_kstore.bat / enc_sig_kstore.sh (for TestEncSigKeyStore).

Note: that TestEncSigGeneratedHtml always has attachment no matter if his command file starts with 8 or 9 parameters.

Removing certificates

If you like, after finishing with examples introduced above, you can remove all certificates and private keys invoked in purpose of testing.

Removing recipient's private key from Outlook 2000 and Internet Explorer:

When you're done with the examples, you can remove the Together certificate and the private key from your certificate list. From Internet Explorer, go to "Tools", and choose "Options" from the menu. Select the "Content" tab and click on "Certificates". Select "Recipient" from the list and click on "Delete". If you have selected to add the Together Root CA to your trusted root list during the process of invoking the private key and the certificates, described earlier, you can remove it by going to the Trusted Root Certificate Authorities tab.

Removing recipient's private key from Netscape Messenger:

When you're done with the examples, you can remove the Together certificate and the private key from your certificate list. Click on the lock in the bottom left corner and select certificates tag: "Yours". Select Recipient ID from the list and click "Delete".

Important notes for your own testing

If you already have your own certificate and private key and don't want to install the Together recipient certificate, you can encrypt examples with your own certificate. You'll need to overwrite the recipient512.cer file with a DER encoded copy of your certificate, or to change argument "recipient512.cer" with file name of your own certificate file in command file.

Making your own .cer files from Internet Explorer and Outlook 2000:

You can export your certificate from Internet Explorer by going to the "Tools", and choosing "Options" from the menu. Next, click on the "Content" tab, and then click on "Certificates". Select your certificate from the list ,click on "Export" and follow the rest of the instructions.

If you are using Outlook 2000, go to the "Tools", and choose "Options" from the menu, click on the "Security" tab, and then select "Import/Export Digital ID". Select "Export your Digital ID to a file", click on

"Select", and then highlight the certificate you'd like to export. Click on "View Certificate", go to the "Details" tab, and click on "Copy to a File." After that, select "No, do not export the private key" and click on "Next". Choose "DER encoded binary X.509" (.CER) and click on "Next". Browse to the recipient512.cer file and overwrite it or save it with another name (then you must change parameter given to appropriate command file). Click on "Finish".

When you run the TestEncrypt.class (enc.bat), TestSigEnc.class (sig_enc.bat) and TestEncSig.class (enc_sig.bat) with your own .cer file they will encrypt with your public key, so you don't need to install a new certificate and private key.

Making your own .pfx files from Internet Explorer and Outlook 2000:

If you would like to use your private key and your own certificates in signing process of the examples instead of supported senders pfx files, you should export your private key with certificate (or certificate chain) into .pfx file following same procedure as described above for certificates, only exporting into .pfx file. In wizard which will guide you through process of creating pfx files you should first press radio button by the option: "Yes export the private key", and then click "Next". Then, on next card check "Include all certificates in the certification path if possible", and than click "Next". After that, you should be asked for password, and on following card you should browse path for location and define your pfx file name. If you define name of .pfx file other than sender512.pfx, you must change parameter in appropriate command file of the examples if you want to use them. Click "Finish" on the end card. This procedures can slightly differ depending on operating system and mail client or browser version but in general, the procedure is very similar.

Making your own .p12 files from Netscape Messenger:

If you are using Netscape Messenger click on the lock in the bottom left corner and select certificates tag: "Yours". Select Recipient ID from the list and click "Export". You will be asked for password, and than browse to path and location where you want to export your private key and certificates. Note that extension of saved file will be .p12 which is Netscape's way of storaging private key and certificates.

Be careful when you export your own certificates in .cer files or your certificates and private key in .pfx or .p12 files. To use this files on regular way in examples, you must know for which type of algorithms and key length they are provided. For example, if you have private and public key combination generated for DSA signing process with key size 1024, you can't use this keys for RSA with MD5 algorithm signing, and you can't at all use this key for any kind of encryption.

Next step to employ your own pfx file is to replace following lines in in source code of TestSigned.java TestEncSig.java and TestSigEnc.java and recompile it.

```
public static String from = "sender@together.at";
public static String password = "together";
```

with

```
public static String from = "<your email address>";
public static String password = "<your pfx password>";
```

Note: that email adress FROM and email adress from used certificates obtained from .pfx or .p12 files must be the same. Do not forget to change parameters given to command files of the examples if your own .cer, .pfx or .p12 files have different names then example supported names.

Chapter 7. Version History

Enhydra-Oyster ver 1.1

The differences and enhancements from ver 1.0.0 are:

- Optimised and modified source for better efficiency and better exploitation possibilities of the object oriented programming.
- Testing the project is done for certificates with key size 1024 and 2048 bits (RSA algorithm), and for certificates 512 and 1024 bits (DSA algorithm).
- Reconstruction of the ways of adding Capabilities Attributes and extension of varieties of Capabilities Attributes to encipher and signing algorithms.
- Added support for DSA signed algorithm.

Enhydra-Oyster ver 1.2

The differences and enhancements from ver 1.1.0 are:

- Improvement in error handling. Redesigned SMIMEException class and changed dealing with exceptions. Now, exceptions are thrown away from SMIME project, and left to user to decide how to handle them.
- Redesigned some parts in source code.
- Updated documentation.
- Involved newer release of Java Cryptography Extension implementation: jce-jdk13-114.jar.
- Added support for building smime120.jar using Ant.

Enhydra-Oyster ver 2.0

The differences and enhancements from ver 1.2.0 are:

- Added possibility of external (explicit) signing of a message.
- Enabled composing of HTML coded content of a message.
- Enabled composing of HTML messages together with their automatically generated alternative (text plain message) by using multipart/alternative Content-Type form of MIME message.
- Added possibilities of adding Content of a message from InputStream as well as from File class object.
- Attachments can be added from InputStream or from file system.
- The process of creating and sending email message is no longer done with MailcapCommandMap. Instead of that, it is now done with DataSource objects.
- Included new tests.
- Updated documentation.

Enhydra-Oyster ver 2.0-1

The differences and enhancements from ver 2.0.0 are:

- Fixed problems with previewing of messages with html composed content in Netscape Messenger.
- Fixed bugs in test examples.
- Improvement in reading Certificate Chain from .pfx and .p12 files with implementation of method getCertificateChain() in PFXUtils class.

Enhydra-Oyster ver 2.1-1

The differences and enhancements from ver 2.0.1 are:

- Optimised and modified three main classes for better efficiency and better exploitation possibilities of the object oriented programming.
- Construction of cryptographics objects with external composed MimeMessage object
- Added more setContent() methods to allow adding external text/plain messages in multipart/alternative MimeMultiparts, instead of autogeneration based on given html message.
- Added object for sending pure MIME messages (withouth S/MIME crypto possibilities)
- Fixed problem with external signing of ASCII files with Unix line break characters.

- More test examples.
- Involved newer release of Java Cryptography Extension implementation: jce-jdk13-118.jar.

Enhydra-Oyster ver 2.1-2

The differences and enhancements from ver 2.1.1 are:

- Involved newer release of Java Mail implementation (ver 1.3.1) newer release of java Activation (ver 1.0.2)
- Fixed problems connected to java.lang.SecurityException which is thrown because class "javax.mail.internet.HeadersUtil"' used in Oyster project is not member of Java Mail implementation.
- Involved newer release of Java Cryptography Extension implementation: jce-jdk13-119.jar.

Enhydra-Oyster ver 2.1-3

The differences and enhancements from ver 2.1.2 are:

- Building of Enhydra-Oyster Project structure is now completely ant based.
- Involved newer release of Java Cryptography Extension implementation: jce-jdk13-122.jar
- Constructors in main classes have been changed. Additional argument 'charset' has been added. For easy implementation of new constructors in old codes new argument can be put to null. Also, few new methods have been added to main classes (reset, setCharsetEncoding and initMimeMessage).

Enhydra-Oyster ver 2.1-4

The differences and enhancements from ver 2.1.3 are:

- Oyster project is now runnable under JDK 1.4.x. Note that before using Enhydra-Oyster, the original JCE Policy jar files must be swapped with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files, which are supplied in this project release.
- The latest release of Java Cryptography Extension and Provider implementation: bcprov-jdk14-123.jar is included.

- All package names, and jar file names are changed.
- The documentation is updated

Enhydra-Oyster ver 2.1-5

The differences and enhancements from ver 2.1.4 are:

- Manuals have been written in docbook format, and during build process they will be transformed to pdf and to html format
- Optional keystore support has been implemented.
- More tests have been provided.

Enhydra-Oyster ver 2.1-6

The differences and enhancements from ver 2.1.5 are:

- Oyster project is tested under JDK 1.5.x
- The latest release of Bouncy Castle Java Cryptography Extension and Provider implementation: bcprov-jdk14-132.jar is included.
- The way of adding cappabilities properties in process of signing is improved.
- Throwing and maintaining of exception is simplified.

Enhydra-Oyster ver 2.1-7

The differences and enhancements from ver 2.1.6 are:

- The latest release of Bouncy Castle Java Cryptography Extension and Provider implementation: bcprov-jdk14-136.jar is included.
- Fixed error in adding of cappabilities properties in process of signing.

Chapter 8. Notes

For all e-mail clients used in tests

Most e-mail clients have support for RC2/40 encryption (weak encryption). For better protection, stronger encryption methods, such as tripleDES are recommended.

All e-mail clients, when perform signing, as default include Signed Attributes and Certificate Chain of sender in message. In our test examples and generally in the whole SMIME project, this way of signing corresponds to arguments "**inclusion of certificates**" set to true and "**inclusion of signed attributes**" set to **true**.

For Netscape Messenger

For testing SMIME with multiple imports of the different .pfx files, it's strongly recommended to import .pfx files with DSA keys before .pfx files with RSA keys. First import should be performed with recipentDSA512.pfx and recipentDSA1024.pfx, and then with other .pfx files.

For Microsoft Outlook and Outlook Express

For using stronger encryption methods then RC2/40 encryption **"Internet Explorer High Encryption Pack"** can be downloaded from address: http://www.microsoft.com/windows/ie/downloads/recommended/128bit/default.asp

7bit Content-Transfer-Encoding

It is not the only condition, for having 7bit Content-Transfer-Encoding, to use characters within range 0 – 127. According to RFC2045: "7bit data" refers to data that is all represented as relatively short lines with 998 octets or less between CRLF line separation sequences [RFC-821]. No octets with decimal values greater than 127 are allowed and neither are NULs (octets with decimal value 0). CR (decimal value 13) and LF (decimal value 10) octets only occur as part of CRLF line separation sequences.

Swapping JCE Policy Files

The following exception:

```
java.lang.SecurityException: Unsupported keysize or algorithm parameters
       at javax.crypto.Cipher.init(DashoA6275)
```

means that swaping of original JCE Policy jar files with Unlimited Strength Java(TM) Cryptography Extension (JCE) Policy Files has not been done at all, or hasn't be done in the appropriate JDK version.