SeeBeyond ICAN Suite

Secure Messaging Extension User's Guide

Release 5.0.3

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

eGate, eInsight, eWay, eXchange, eXpressway, eBI, iBridge, Intelligent Bridge, IQ, SeeBeyond, and the SeeBeyond logo are trademarks and service marks of SeeBeyond Technology Corporation. All other brands or product names are trademarks of their respective companies.

© 2001-2004 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20040315114505.

Contents

Chapter 1

Introducing Secure Messaging Extension

This document describes how to install, configure, and use the SeeBeyond Technology Corporation's Secure Messaging Extension, referred to as SME throughout the rest of this document.

The topics in this chapter include:

1.1 Document Organization

This User's Guide is organized into two parts. The first part, consisting of Chapters 1-2, introduces SME and describes the procedures for installing and setting up the program. This part should be of particular interest to a System Administrator or other user charged with the task of getting the system up and running.

The second part, consisting of Chapters 3-6, describes the details of SME operation and configuration, including descriptions of encrypted message formats, instructions on Keystore management, and implementation of sample SME Projects. This part should be of particular interest to a Developer involved in customizing SME for a specific purpose.

1.2 Overview

SME enables eGate to process Events using the S/MIME (Secure Multipurpose Internet Mail Extensions) message format. This format is the IETF RFC 2311 specification for encrypting and/or signing types of data.

SME supports encryption, decryption and authentication of messages and is interoperable with any other client applications that support the S/MIME standard.

SME adds the following features to transactions:

- privacy
- message (Event) authentication
- sender authentication
- nonrepudiation

1.2.1 Components

Components required to run SME include:

- eGate Integrator
- File eWay (required for the sample Project)
- Keys and Certificates

1.3 Supported Operating Systems

Secure Messaging Extension is supported on the following operating systems:

- Windows 2000, Windows XP, Windows Server 2003
- Solaris 8 and 9
- AIX 5.1 and 5.2
- HP-UX 11.0 and HP-UX 11i (RISC)
- HP-UX 11i V2 (11.23)
- HP Tru64 5.1A
- Red Hat Linux 8 (Intel)
- Red Hat Linux Advanced Server 2.1 (Intel)

1.4 System Requirements

To set up and run the SME with the eGate Enterprise Designer, you need the following:

- A TCP/IP network connection.
- Windows Server 2003, Windows 2000, or Windows XP. This is required for the User Interface.
- Microsoft Internet Explorer 6.0 SP1 or above.

Note: *Open and review the Readme.txt prior to installation for any additional requirements.*

1.5 Introducing Secure Messaging Extension (SME)

The SME product has the dual purpose of offering security features, which allow protected transmission of public domains such as the internet, and compression/decompression technology to effectively reduce/expand the size of files.

Security Component

As part of the security component, SME uses Public Key Infrastructure (PKI) technology to ensure the confidentiality of exchanges. This is done by digitally signing and encrypting messages as they are sent, and decrypting and authenticating messages when they are received.

SME performs the encryption and decryption of messages using the Secure/Multipurpose Internet Mail Extension (S/MIME). S/MIME is a specification for securing electronic mail, and is designed to add security to e-mail messages in MIME format.

S/MIME creates one-way hash algorithms that ensure data integrity by verifying no modifications are made to the message while in transit. In addition, the message sender's identity is verified through the use of digital signatures, proving that the message actually originated from the entity who claims to have sent it. For more information on the S/MIME format, see **"Introducing Secure Multipurpose Internet Mail Extension (S/MIME)" on page 11**

Security Services Offered Through SME Include:

- Encryption
- Decryption
- Sign
- Verify

Compression Component

SME compression converts string and binary file formats, such as those found in text, graphics, audio, and video files, into smaller sized files. This is done using Java-based mathematical equations that scan and index repetitive patterns. If a file contains repetitive patterns—such as colors used in an image—then code is written to index the number of and exact placement of those patterns, effectively reducing the size of the file. When you decompress a file, the code that contains the index of repetitive patterns rebuilds the file to its original format.

Compression Services Offered Through SME Include:

- Compression
- Decompression

1.6 Introducing Multipurpose Internet Mail Extension (MIME)

MIME Message Format

As a specification for formatting non-ASCII messages, MIME enables the transfer and acceptance of files via the Internet mail system. MIME-compliant messages may contain any type of data, including the following:

- Text messages in US-ASCII
- Messages of unlimited length
- Binary files
- Character sets other than US-ASCII
- Multi-media: Image, Audio, and Video objects
- Multiple, nested objects in a single message

When later sent over a protocol such as HTTP or FTP, which provide a "binary clean" data path, MIME messages may be left in binary format. However, if the MIME message is sent via SMTP (E-mail) or other text-only protocols, binary objects must be encoded using the Base64 content transfer encoding format, which produces a textual representation of the original binary data.

Messages in MIME format consist of two parts: the header and the body. The header forms a collection of metadata in the form of keyword/value pairs structured to provide information necessary for the transmission and interpretation of the message. The body of the message contains the bulk data to be transferred. In turn, S/MIME defines the security services, adding digital signatures and encryption, thus preventing forgery and interception.

For more information regarding MIME, see the Internet Engineering Task Force Text Messages specification (RFC 822) and the MIME Message Body Format (RFC 2045), at http://www.ietf.org.

The S/MIME Version 3 specification (RFC 2623) is also found at http://www.ietf.org.

1.7 Introducing Secure Multipurpose Internet Mail Extension (S/MIME)

S/MIME is an encryption supported version of the MIME protocol. It is based on the Public Key Cryptography Standards (PKCS), which specify how the RSA public-key cryptographic algorithm should be used to implement enveloped encryption and digital signatures.

The RSA public-key system makes use of two related keys to perform the mathematical algorithms necessary to encrypt or decrypt data: a public key, which may be made available to any prospective correspondent, and a private key known only to the key's owner. A public key can be published openly, thereby assuring the ability of anyone to send secure messages that can only be decrypted by the owner of the respective private key.

Encryption can also be performed using one's private key, and decrypted with the corresponding public key. In this case, the encryption result is known as a digital signature, which guarantees to the intended recipient that the signed message is authentic and genuinely came from the stated originator of the message.

Digital signatures provide data integrity, authentication and non-repudiation of an electronic document. Successful verification of a digital signature ensures the recipient that the “document received” is identical to the “document sent” (data integrity) and confirms the identity of the sender (authentication). It also prevents any subsequent denial by the sender that the document originated with them (non-repudiation).

In practice, public keys are stored as certificates that comply with the X.509 standard. In addition to the public key, a certificate also contains information about the key owner's identity, the key's validity, and the issuer of the certificate, also known as a Certificate Authority.

1.8 Overview of SME Processes

The following diagrams outline the key activities involved in the SME processes, including:

- SME Encryption/Decryption Process
- SME Signature/Verification Process
- SME Compression/Decompression Process

1.8.1 SME Encryption/Decryption Process

This section describes the internal and external flow of the SME encryption, using the key pair encryption method.

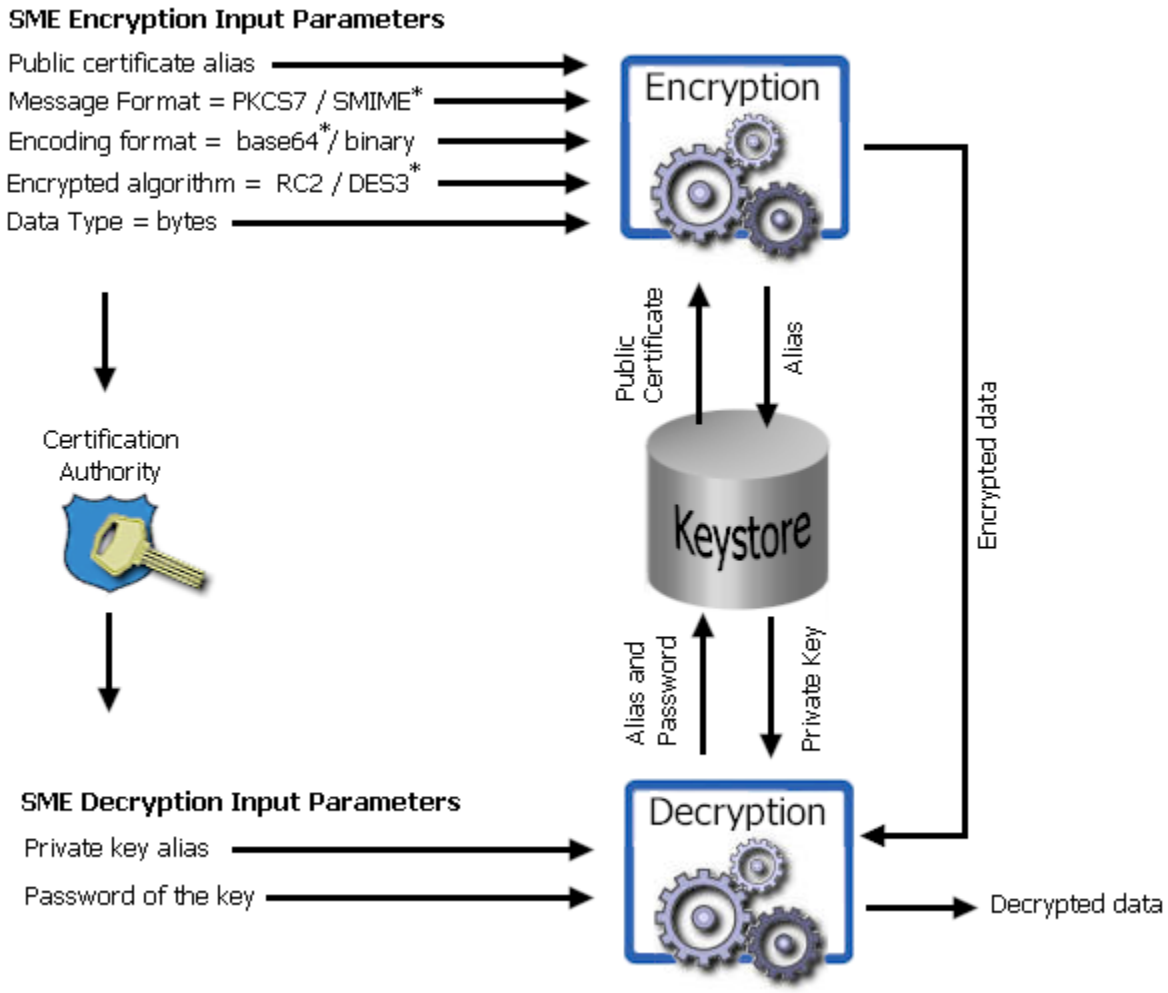
The encryption process begins when the sender's message is encrypted with the public key. The message is also signed by the sender, and the signature itself is encrypted with the sender's private key. When the reader receives the message, the encryption is decoded with the reader's private key. The sender's Public Certificate, located in the Keystore is used to verify the authenticity of the public key.

In addition to verifying the public key, public certificates also contain the sender's personal information, such as name, institution, and e-mail address, and are signed by a trusted Certificate Authority.

During encryption, a Public Certificate alias is used to identity the Public Certificate located in the Keystore. During decryption, the reader's private key alias and password is used to access the Private Key from the Keystore and decrypt the message.

The encryption/decryption process illustrated in **Figure 1 on page 13**, details the SME Input Requirements for both encryption and decryption of data.

Figure 1 Secure Messaging Extension Encryption Process



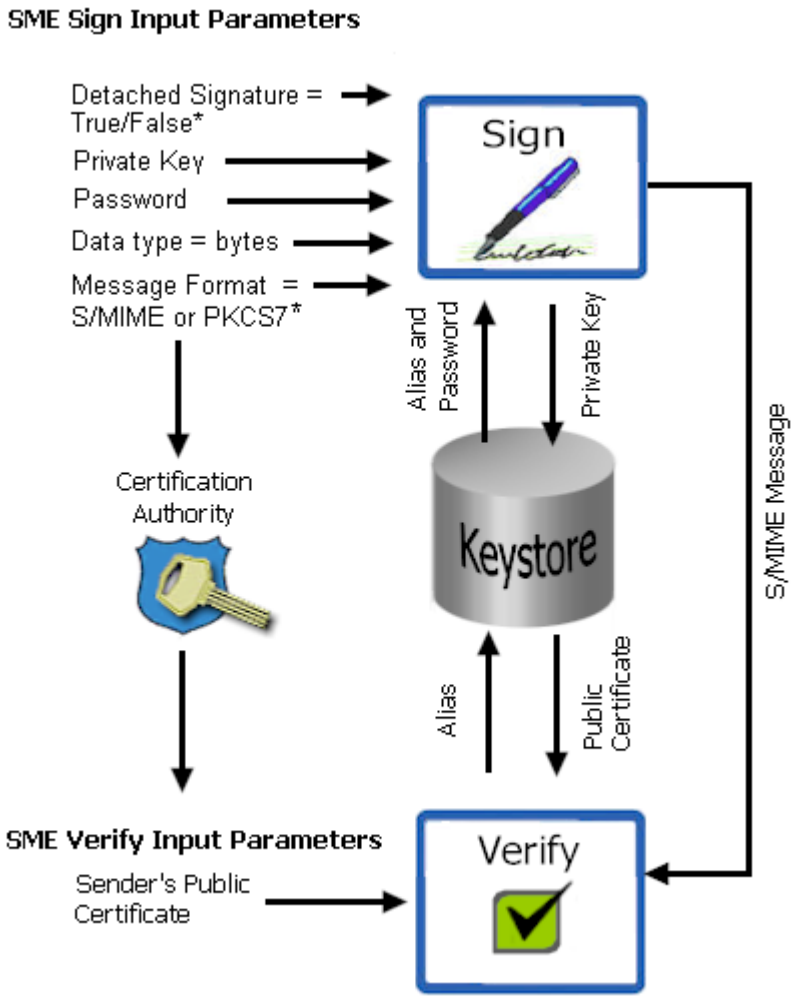
Note: *Input parameters listed with a "*" symbol denote the default used.*

1.8.2 SME Signature/Verification Process

The SME signature/verification process begins when a subscriber publishes a certificate to a Certificate Authority. Published certificates contain the subscriber's identity and public key, and are digitally signed by the Certification Authority. The Certification Authority is also responsible for safeguarding access to the subscriber's private key, which is required during the verification process.

When a subscriber signs and sends a message, the SME Sign process converts the message from MIME to S/MIME format. The S/MIME message format also contains the digital footprint of the subscribers private key, so when the message is received by another user, the public key held by the Certification Authority "reads" and then verifies the digital signature created by the private key.

Figure 2 Secure Messaging Extension Signature Verification Process



Note: *Input parameters listed with a "*" symbol denote the default used.*

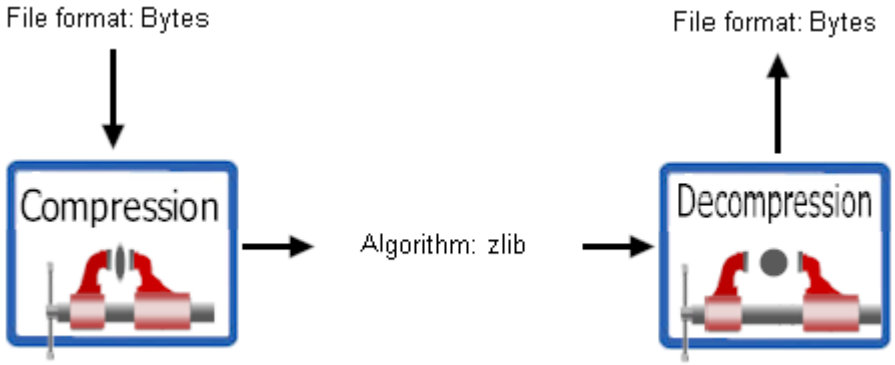
1.8.3 SME Compression/Decompression Process

The SME compression process converts byte type files into PKCS#7 format using the zlib compression library. For more information on the PKCS#7 see **"PKCS#7 encrypted message format" on page 19**.

For more information on the zlib compression library, visit the gzip home page at:

http://www.gzip.org

Figure 3 Secure Messaging Extension Compression Process



Chapter 2

Installation

This chapter describes the procedures for installing SME.

2.1 Before Installing Secure Messaging Extension

Open and review either the Readme.txt for any additional information or requirements, prior to installation. The Readme.txt is located on the Repository CD-ROM.

2.2 Installing the Secure Messaging Extension

During the installation process, the Enterprise Manager, a web-based application, is used to select and upload the SME component (SMEWebServices.sar) from the eGate installation CD-ROM to the Repository.

Installing eGate

The eGate installation process includes the following components:

- Installing the eGate Repository
- Uploading products to the Repository
- Downloading components (including eGate Enterprise Designer and Logical Host)
- Viewing product information home pages

To install the SME component on an eGate supported system, follow the instructions for installing the eGate Integrator in the ICAN Installation Guide, and include the following steps:

1. During the procedures for uploading files to the eGate Repository using the Enterprise Manager, after uploading the **eGate.sar** file, select and upload the following file:
   - **SMEWebServices.sar** (to install the SME component)
   - **FileeWay.sar**
   - **SMEWebServicesDoc.sar** (to install the User's Guide, JavaDoc, and sample Projects)
2. In the Enterprise Manager, click the **DOCUMENTATION** tab.
3. Click **Secure Messaging Extension**.
4. In the right pane, click **Download Sample**, and select a location for the .zip file to be saved.

2.3 Additional Files Required to Run SME

Additional policy JAR files are needed to run SME. The type of JAR files required depends on the JVM used. Refer to your JVM vendor for exact details on the specific policy JAR file requirements.

Use the following table to determine which JRE is included in the eGate logical host.

Table 1 JRE Versions Listed by Operating System

Operating System	JRE	URL location
Windows, Solaris, Linux, HP-UX, Tru64	1.4.2	http://java.sun.com/j2se/1.4.2/download.html
AIX	1.4.1	http://java.sun.com/products/archive/j2se/1.4.1_07/index.html

To download the required JAR files:

1. Scroll to the bottom of the web page listed in Table 1 for the JRE.
2. Click the link to the Unlimited Strength Jurisdiction Policy Files 1.4.1 or 1.4.2.
3. Click the link to download the ZIP file containing the required policy jar files.

Required policy jar files include:

- **local_policy.jar**
- **US_export_policy.jar**

Then, for each of your logical hosts, replace the versions of these files in:

<logicalhost>/jre/lib/security/

In addition, if you are running a repository on AIX, also replace the versions of these files in:

<AIXrepository>/jre/1.4.1/security/

Chapter 3

Encrypted Message Formats, Digital Signature Formats, and Certificate Formats

This chapter provides an overview of the encrypted message formats, digital signatures and certificates that are handled by SME. In addition, this chapter describes how to use Microsoft™ Internet Explorer tools to transfer certificate formats accepted by the SME.

3.1 Encrypted Message Formats

This section provides examples of encrypted message formats.

PKCS#7 encrypted message format

The PKCS#7 format, as specified by RFC 2315, is used for basic digitally signed and/or encrypted data. This format does not provide a MIME header, and produces mostly binary data, except for a few character strings in an embedded certificate, as shown in the following example:

```
0     *†H†÷  0  1,$0,  0^0,10UUS10\U

California1\0/UMonrovia1

0

U

STC10UDevelopment1'0%USTC Test Certificate Authority0*†H†÷
 V<±ïíà»,¯‡¾l-êÒTâž|g®<êÆ<õ¢\)Ç‰‡îQtfrµ»Ÿ½TûRP[Myß÷ ×ÚÚh-Íá-Ù¼–áô)Ã|bF©[_^HESM†2?k_
z¸~½ ï/ÈÕ+¶>æ³G¨šXK8yÃ!·Âyá—œB4U0 *†H†÷0*†H†÷b

4˜mDY  jE¯††,ë-]2žI¯e´G®†Ö¤ŸQÜ&ZÈX,¶Ê!4`RK”ÆE«9ýìÂPÝ Q- ní\=(-÷þÚïL
```

S/MIME2 encrypted message format (base64)

The S/MIME2 format is also used to represent digitally signed and/or encrypted data. This format provides a MIME header and encrypted results, with the binary data encoded as printable characters using the base64 method, as shown in the following example:

```
Content-Type: application/pkcs7-mime; name = "smime.p7m"
Content-Transfer-Encoding:base64
MIAGCSqGSIb3DQEHA6CAMIACAQAxggEkMIIBIAIBADCBiDCBgjELMAkGA1UEBhMCVVMxEzARBgNV
BAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCE1vbnJvdmlhMQwwCgYDVQQKEwNTVEMxFDASBgNVBAsT
C0RldmVsb3BtZW50MScwJQYDVQQDEx5TVEMgVGVzdCBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkCARMw
DQYJKoZIhvcNAQEBBQAEgYBR3Hwe+1JB2pZuR2XdNFS1DISYbgWHaXcmmpRZE+r35Ar5iaNlfRAj
ipc1RBW0HmidnWz3zBGYOml91btVjy2z6dmoDknnksgTI77YX727hESHgjCpxxcs+1kRzzI5ZUlU
WvvXeX/7wNkx3ZgJOrtIiXjfs6t8zW4edd1/13fQgjCABgkqhkiG9w0BBwEwFAYIKoZIhvcNAwcE
CBUeyy6UZb4koIAECOpD8MyUjNZ/BAjB0O2dStz8HgQIiPOI1H4tpfsECARjsNRDbMpqBAgtC3S1
7FnAWQQI8ymbLzoB4kUECF38LESRhXN2BAhcGnYwRqQDMgAAAAAAAAAAAA=
```

S/MIME2 encryption message format (binary)

This format represents a message as binary, non-printable data, with appropriate MIME headers, as shown in the following example:

```
Content-Type: application/pkcs7-mime; name = "smime.p7m"
Content-Transfer-Encoding:binary
0     *†H†÷  0  1,$0,  0^0,10UUS10\U
California1\0/UMonrovia10
U
STC10UDevelopment1'0%USTC Test Certificate Authority0*†H†÷
 V<±ïíà»,¯Qt£rµ»Ÿ½TûRP[Myß÷ ×ÚÚh-Íá-Ù¾—áô)Ã|bF©[_^HESM†2?k …Bmm_t1Gòz
~½ ï/ÈÕ+¶>æ³G¨šXK8yÃ!·Âyá—œB4U0 *†H†÷0*†H†÷b
4˜mDY  jE¯††,ë-]2žI¯e´G®†Ö¤ŸQÜ&ZÈX,¶Ê!4`RK”ÆE«9ýìÂPÝ Q- ní\=(-÷þÚïL
```

3.2 Digital Signature Formats

Although signatures are normally found attached to the message or file that they sign, detached signatures are also supported. A detached signature may be stored and transmitted separately from the message it signs.

Table 2 lists the features of each encrypted message format for attached signatures.

Table 2 Formats for attached signatures

PKCS#7 Format	S/MIME2 Format
▪ Includes original document in plain text, digital signature, and certificates involved, encapsulated, and encoded in Abstract Syntax Notation One (ASN.1) standard format. ***Note:*** *ASN.1 is an ISO/IEC standard for encoding rules used in ANSI X.509 certificates and PKCS documents.* **Example** 0 *†H†÷ 0 10+ 0 *†H†÷ $: This is only a test message! ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0% ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9P°KÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þ^=>P}°æå·ÌZ§R˜øüÂÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0^0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö›»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑ^©pCËŸÂÍ,•Ÿ¶I/’{- ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥›Ì°]2Åpæ2*Ì |êÏË{lÊ—0%#t‹¥Œåœ ô› VÝ¹k	▪ Includes: ▪ MIME headers ▪ PKCS#7 attached signature object **Example** Content-Type: application/pkcs7-mime; name = "smime.p7m" Content-Transfer-Encoding:binary 0 *†H†÷ 0 10+ 0 *†H†÷ $: This is only a test message! ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0% ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9P°KÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þ^=>P}°æå·ÌZ§R˜øüÂÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0^0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö›»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑ^©pCËŸÂÍ,•Ÿ¶I/’{- ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥›Ì°]2Åpæ2*Ì |êÏË{lÊ—0%#t‹¥Œåœ ô› VÝ¹k

Table 3 lists the features of each encrypted message format for detached signatures.

Table 3 Formats for detached signatures

PKCS#7 Format	S/MIME2 Format
▪ Includes signature and certificate without the signed data. ***Note:*** *RNIF1.1 uses PKCS#7 and detached format* **Example** 0 *†H†÷ 0 10+ 0 *†H†÷ ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0% ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAmf8ófsoU¢&mUþ„g,”>©kfÄXqÜ±Q½êÔú9PªKÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þ^=>P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0^0,10UUS10\U California1\0/UMonrovia1 0 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö›»/éR8¶ZaÖ” ¡ÝXS*¿fuõURÑ^©pCËŸÂÍ,•Ÿ¶I/’{- ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥›Ì°]2åpÆ2*Ì |êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k .	▪ Includes a MIME multipart message consisting of the original data in one segment, and the binary format signature in a second segment. **Example** Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="SHA1"; boundary="Boundary_12e4421e_NOEWUDYA" --Boundary_12e4421e_NOEWUDYA Content-Type: text/plain This is only a test message! --Boundary_12e4421e_NOEWUDYA Content-Type: application/pkcs7-signature; name="smime.p7s" Content-Transfer-Encoding: binary Content-Disposition: attachment; filename=smime.p7s 0 *†H†÷ 0 10+ 0 *†H†÷ ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia1 0 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0% ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAmf8ófsoU¢&mUþ„g,”>©kfÄXqÜ±Q½êÔú9PªKÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þ^=>P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0^0,10UUS10\U California1\0/UMonrovia1 0 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö›»/éR8¶ZaÖ” ¡ÝXS*¿fuõURÑ^©pCËŸÂÍ,•Ÿ¶I/’{- ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥›Ì°]2åpÆ2*Ì |êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k --Boundary_12e4421e_NOEWUDYA--

Table 3 Formats for detached signatures

PKCS#7 Format	S/MIME2 Format
	▪ Includes a MIME multipart message consisting of the original data in one segment, and the base64-encoded signature in a second segment **Example** Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="SHA1"; boundary="Boundary_12e4421e_FNGBRNRI" --Boundary_12e4421e_FNGBRNRI Content-Type: text/plain This is only a test message! --Boundary_12e4421e_FNGBRNRI Content-Type: application/pkcs7-signature; name="smime.p7s" Content-Transfer-Encoding: base64 Content-Disposition: attachment; filename=smime.p7s MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAMIAGCSqG SIb3DQEHAQAAoIICbTCCAmkw ggHSAgETMA0GCSqGSIb3DQEBBAUAMIGCMQswCQYDVQQGEwJVUzET MBEGA1UECBMKQ2FsaWZvcm5p YTERMA8GA1UEBxMITW9ucm92aWExDDAKBgNVBAoTA1NUQzEUMBIG A1UECxMLRGV2ZWxvcG1lbnQx JzAlBgNVBAMTHlNUQyBUZXN0IENlcnRpZmljYXRlIEF1dGhvcml0 eTAeFw0wMjA1MDkxODQ2MzNa Fw0wMzA1MDkxODQ2MzNaMHcxCzAJBgNVBAYTAlVTMRMwEQYDVQQI EwpDYWxpZm9ybmlhMREwDwYD VQQHEwhNb25yb3ZpYTESMBAGA1UEChMJU2VlQmV5b25kMQwwCgYD VQQLEwNSQUQxHjAcBgNVBAMT FVNlZUJleW9uZCBUZXN0IFVzZXIgMTCBnzANBgkqhkiG9w0BAQEF AAOBjQAwgYkCgYEAropHa5XJ g3evpQFTrqJfeyEw1aKEJksfx+lMm8QsnTHEbqdsj8+7ttXvrKml JGx5bbSezzkIl81v0Uwfc3XJ QSOaA2teIxr8swvFDcWnXfFzSkFtkKM482Zzb1WiJhZtVf6EZywD nT6pAmujxFhx3LFRverU+jlQ ukvNfpL6ky8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQBfE2IVmo9G 7xRvN3IZC+emq0jqE0HfbJMi eg1nf9vmQcycWBT6LJHVOt6IPZtQfbDmf+W3zFqnUpj4/ MUGzCjg2EnjtYD30Y9qI5vyF1IxL52M ODBA7PvNgq0vYYYY239a/f21t3Mh378dYXlTkZ0jfQmF9/ znXyLrtdDJtTS9pjGCAS0wggEpAgEB MIGIMIGCMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5p YTERMA8GA1UEBxMITW9ucm92 aWExDDAKBgNVBAoTA1NUQzEUMBIGA1UECxMLRGV2ZWxvcG1lbnQx JzAlBgNVBAMTHlNUQyBUZXN0 IENlcnRpZmljYXRlIEF1dGhvcml0eQIBEzAHBgUrDgMCGjANBgkq hkiG9w0BAQEFAASBgJCd1gU+ uw/pUji2WmHWlCChE91YUyq/ o3X1VQVS0YipcEPLn8LNLI2ftkkvknuWqtNJykY2Mp5T8ICHFy/x SbJlXvwj4oTg8Ga3bgUdKJ1hRbFj0yzFpT7MsF0yxXDGMirMCnzq z8t7bMqBlzAlIw10i6WM5ZyA 9JsaH1bdE7lrAAAAAAAA --Boundary_12e4421e_FNGBRNRI--

3.3 Signing and Attaching Signatures

In an S/MIME message with a detached signature, the signature is calculated over on the entire payload data, in addition to its MIME header(s). The default Content-Type for such a MIME part is text/plain.

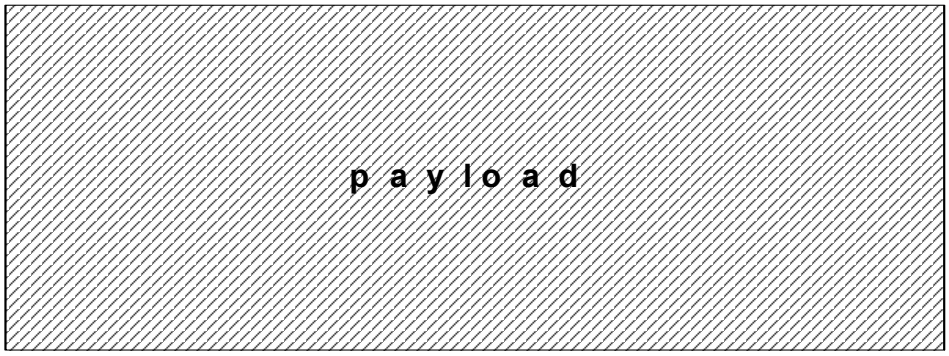
If signing a Content-Type other than text/plain, the user must generate a Content-Type header line for the payload. All other MIME headers and boundaries, including those of the detached signature part, are produced by SME.

An example XML message, digitally signed with a base64-encoded detached S/MIME signature is shown below.

```
MIME-Version: 1.0
Content-Type: multipart/signed;
protocol="application/x-pkcs7-signature"; micalg=sha1;
     boundary="----FA4D3A12E6192B82B05284F061C7CE55"

This is an S/MIME signed message

------FA4D3A12E6192B82B05284F061C7CE55
Content-Type: application/xml




------FA4D3A12E6192B82B05284F061C7CE55
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"


------FA4D3A12E6192B82B05284F061C7CE55--
```

3.4 Private Key Format

Private keys, used by SME in the decryption and signing processes, are required to be in PKCS#12 format. If a key has been generated through a browser-based process and appears among your personal certificates in Microsoft Internet Explorer, it may be exported to a PKCS#12 file for use by SME. Procedures on converting and exporting certificate formats are included in section 3.5 of this chapter.

Note: *Remember the password you specify to encrypt the exported file; it is needed during the SME configuration process, in order to allow decryption and use the key.*

3.5 Certificate Formats

A Certificate, also called a Public Key Certificate, is an electronic message issued by a Certificate Authority that is used to match the value of the public key to the identity of the person, device, or service that holds the corresponding private key.

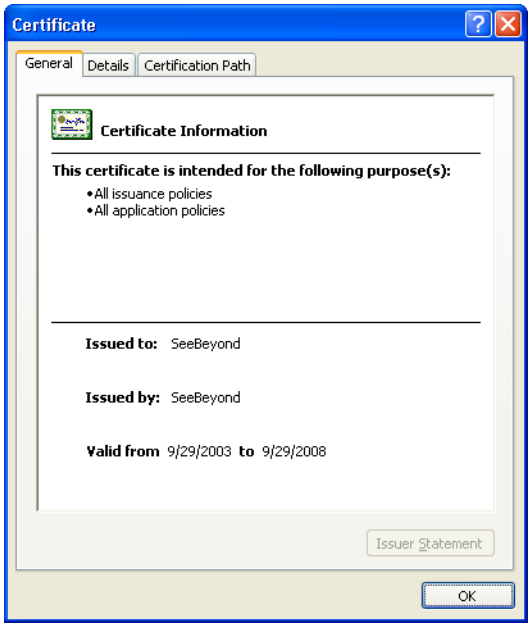
SME only accepts certificates in PKCS#7 format and DER encoded binary X.509.

Microsoft Internet Explorer (IE) provides a Certificate Wizard tool to convert between formats.

Using IE to convert one certificate format to another

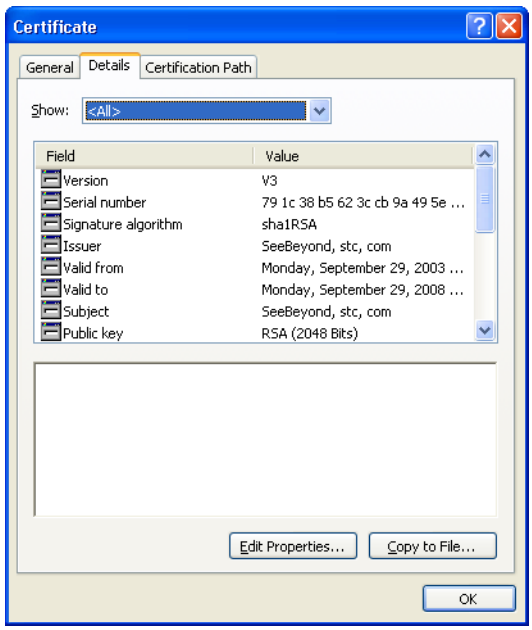
1 Double-click the certificate file to open the certificate properties, as shown in Figure 4.

Figure 4 Certificate File



2 Select the **Details** tab, as shown in **Figure 5 on page 26**.

Figure 5 Certificate Detail Tab

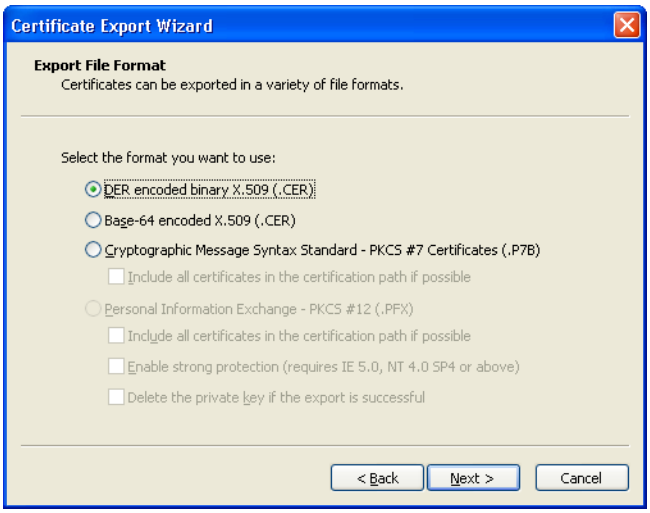


3 Click **Copy to File**. The Certificate Export Wizard appears, as shown in Figure 6.

Figure 6 Certificate Export Wizard

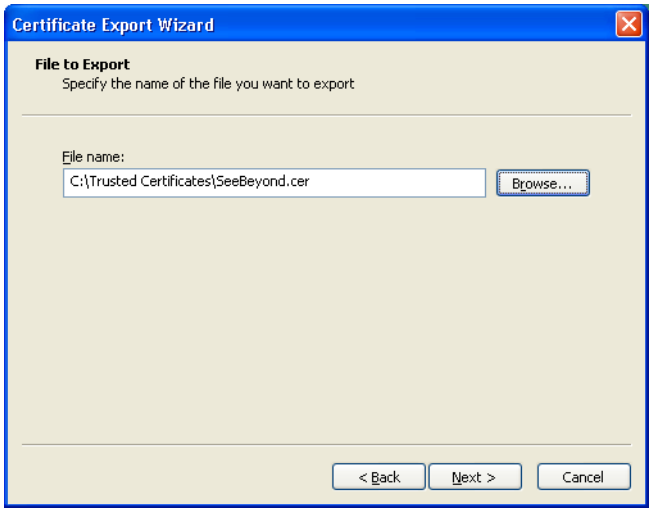
4 Click **Next** to open the certificate export file format, as shown in **Figure 7 on page 27**, and select the format.

Figure 7 File Format window



5 Click the **Next** button. The File to Export window appears, as shown in Figure 8.

Figure 8 File to Export window



6 Browse to and select the file name for the Certificate and choose the **Next** button. Details of the completed certificate appear, as shown in **Figure 9 on page 28**.

Figure 9 Completed Certificate Details window

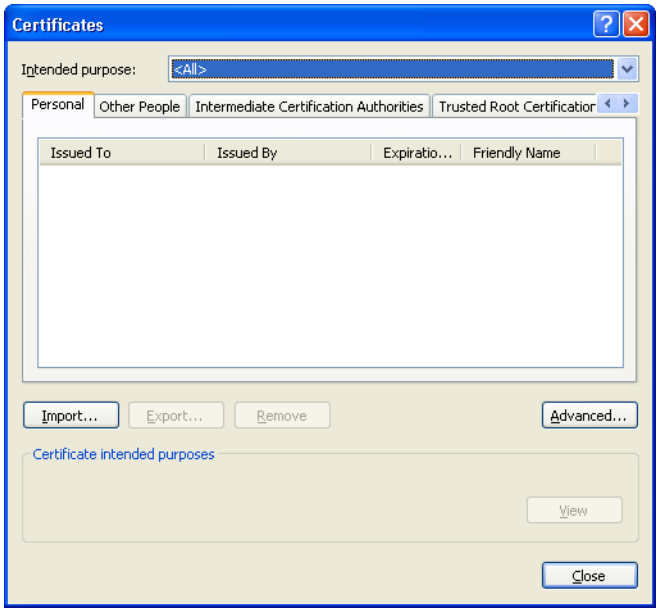
7 Click the **Finish** button to exit the Wizard.

To transfer the certificate formats using Microsoft Internet Explorer

1 From the Tools menu, click **Internet Options**.

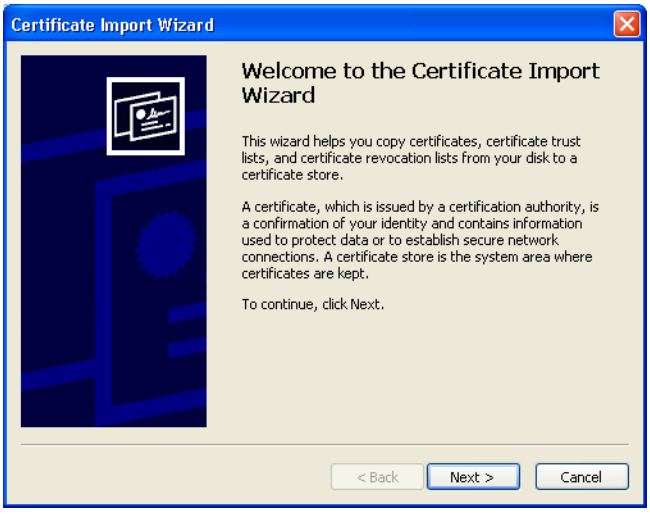
2 Click the **Content** tab and then click **Certificates**. The Certificates dialog appears, as shown in Figure 10.

Figure 10 Internet Explorer Certificates



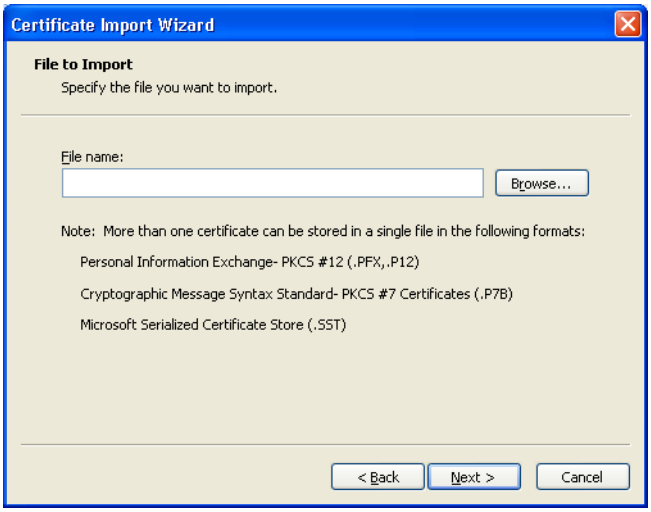
3 Click the **Import** button, the Certificate Import Wizard appears, as shown in **Figure 11 on page 29**.

Figure 11 Certificate Import Wizard



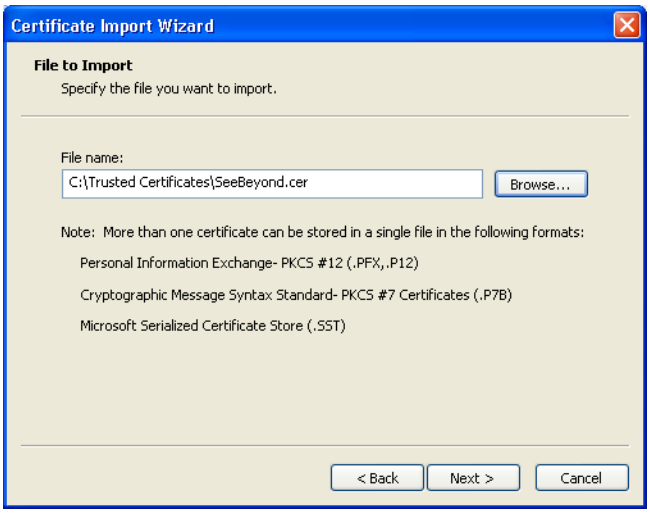
4 Click the **Next** button, the File to Import window appears, as shown in Figure 12.

Figure 12 File to Import window



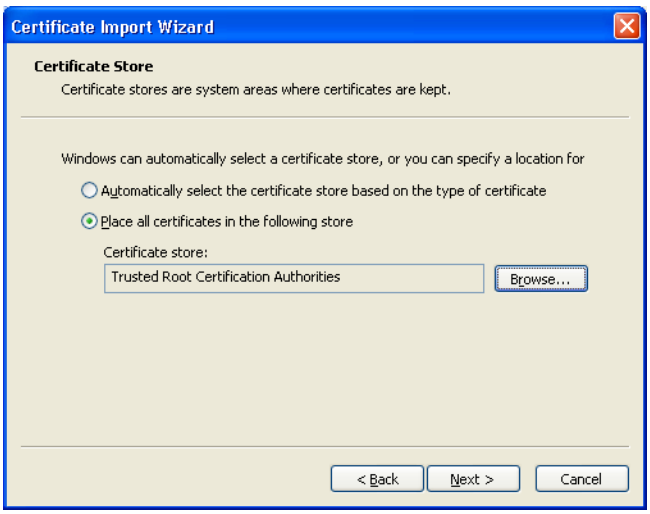
5 Click the **Browse** button and locate the certificate to open.

Figure 13 File to Import window



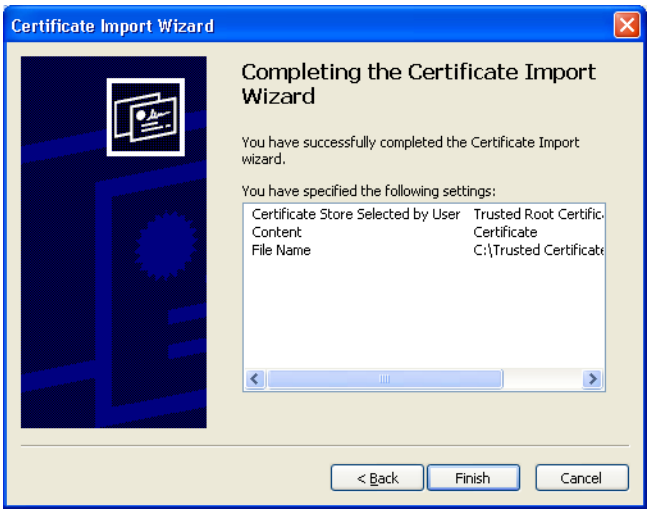
6 Click the **Next** button. The Certificate Store window appears, as shown in Figure 14.

Figure 14 Certificate Store window



7 Browse to the location where you want the certificate stored and click the **Next** button. Details of the completed certificate import appear, as shown in **Figure 15 on page 31**.

Figure 15 Completed Certificate Details window



8 Click the **Finish** button to exit the Wizard.

Chapter 4

Managing Keystores and Trueststores

This chapter describes the procedures for creating and managing Private Keys, Public Keys, and truststore certificates.

This Chapter Includes

4.1 Overview

Keystores are repositories for the sensitive cryptographic key information required for self-authentication. Key entries are typically private keys and are accompanied by the certificate chain for the corresponding public key.

Truststores hold all public key certificates belonging to the other party, or in the case with SME, the message sender. Certificates held in the Trust Store are considered Trusted Certificates since the Key Store owner trusts that the public key in the certificate belongs to the identity provided by the subject or owner of the certificate.

During runtime, one Keystore is created for each ICAN Environment, but several truststores may exist to accommodate the different relationships between trading partners. ICAN commonly groups both Keystores and truststores under the common name "Keystore", however, both are considered separate entities.

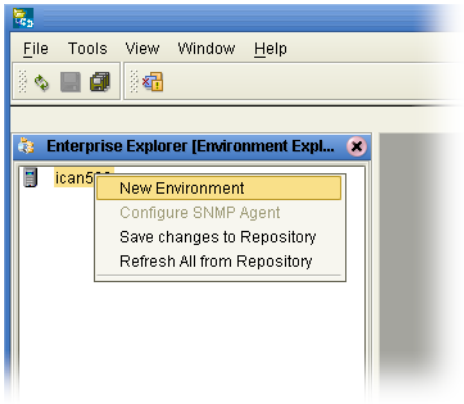
4.2 Steps Required to Create and Manage Private Keys

eGate Integrator includes Keystore and truststore management functionality. Using Environment Explorer, you first create a new Keystore environment, and then import or export private keys, create new truststores, and manage public certificates.

To Import a New Certificate:

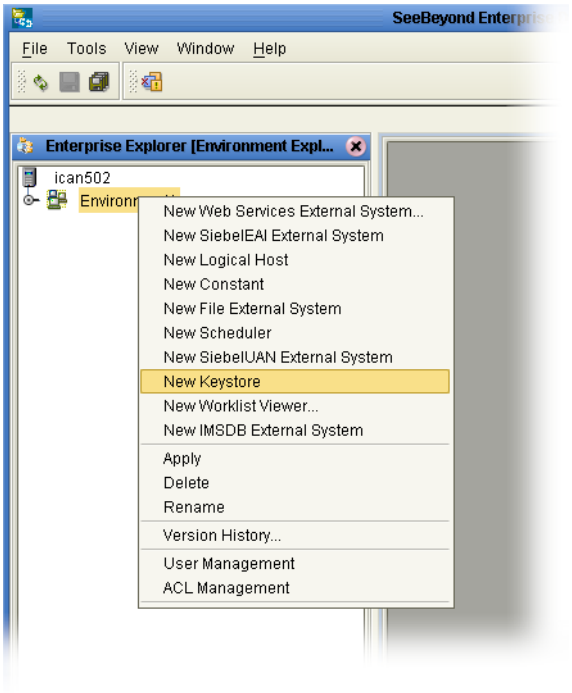
1 From the Environment Explorer, right-click the Environment icon and choose **New Environment**.

Figure 16 Creating a New Environment



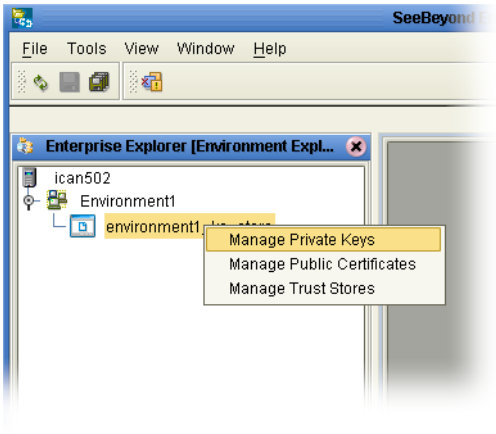
2 Right-click the new Environment and choose **New Keystore** from the selection menu, as shown in **Figure 17 on page 34**. This creates a new Keystore called Environment-ks-store.

Figure 17 New Keystore selection



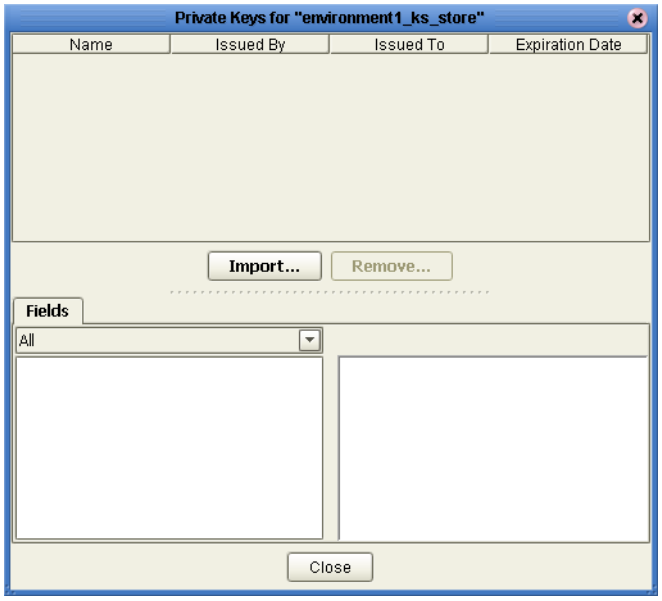
3 Right-click Environment-ks-store and choose **Manage Private Keys** from the selection menu.

Figure 18 Manage Private Keys



4 The **Private Keys for "environment1_ks_store"** window appears.

Figure 19 Private Keys for "environment1_ks_store" window



5 Click the **Import** button, the Import Private Key window appears.

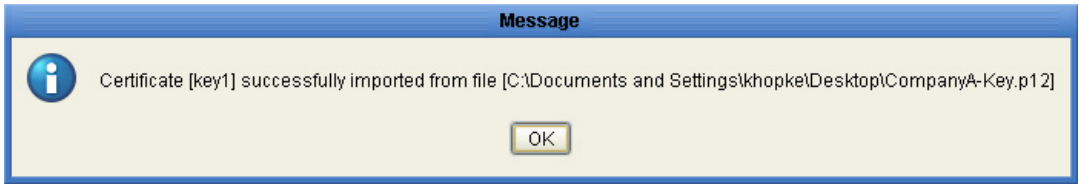
Enter the following information:

- **Alias** – the name you want associated with the certificate
- **File** – the location of the certificate
- **Password** – the password required to access the Private Key

Figure 20 Import Private Key window

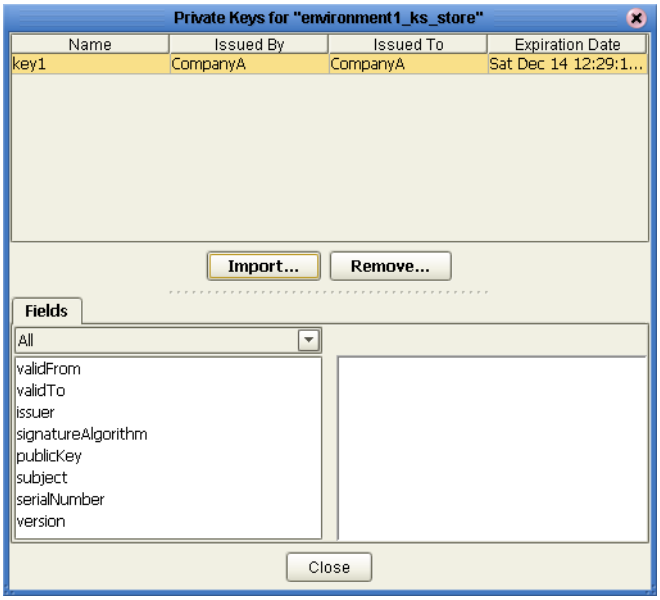
6 Click the **Import** button. A Message window appears confirming the import.

Figure 21 Import Confirmation Message



7 Click the **OK** button, the **Private Keys for "environment1_ks_store"** window appears with a key pair description that displays the name and details of the imported key pair, as shown in Figure 22 on page 36.

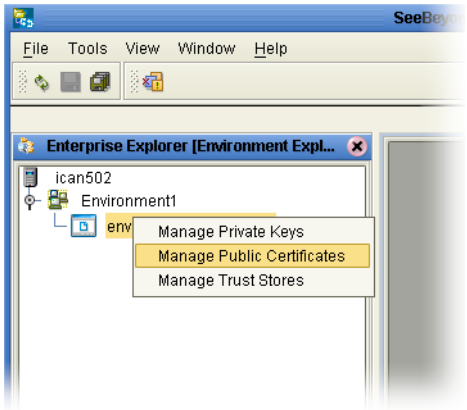
Figure 22 Key Pair Description window



To Manage a Public Certificate:

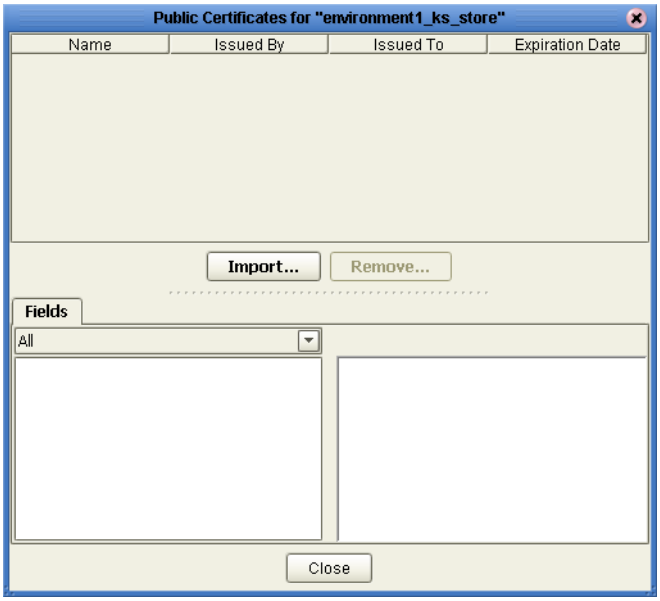
1 From Enterprise Explorer, right-click Environment-ks-store and choose **Manage Public Certificates** from the selection menu.

Figure 23 Manage Public Certificates



2 The **Private Keys for "environment1_ks_store"** window appears, as shown in Figure 24 on page 37.

Figure 24 Manage Public Certificates window

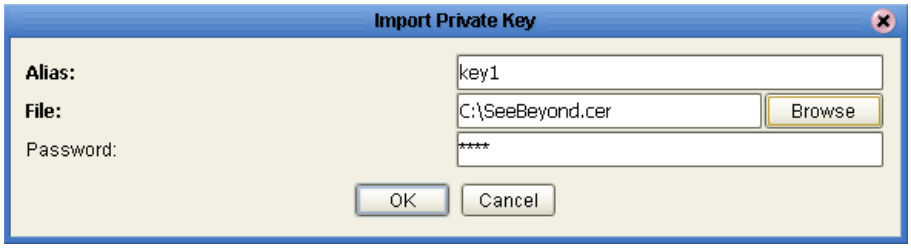


3 Click the **Import** button. The **Import Private Key** window appears.

Enter the following information:

- **Alias** – the name you want associated with the certificate
- **File** – the location of the certificate

Figure 25 Import Certificate window

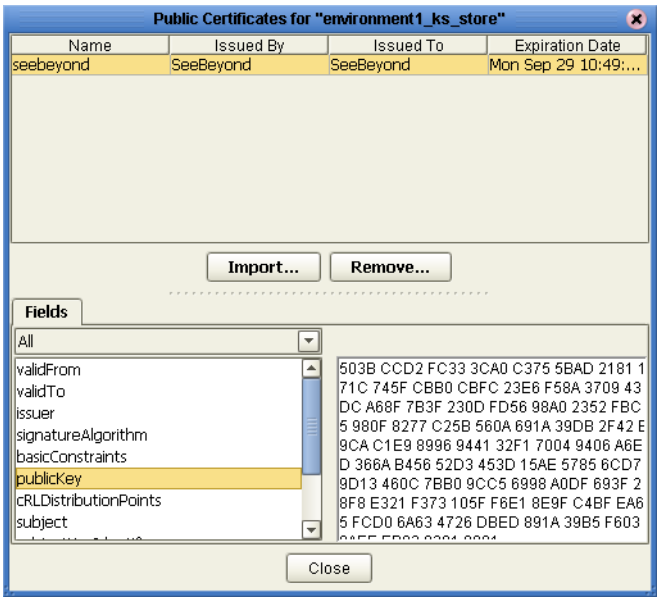


4 Click the **Import** button. A Message window appears confirming the import.

Figure 26 Import Confirmation Message

5 Click the **OK** button. The **Public Certificates for "environment1_ks_store"** window appears. To view the field details, click on the imported certificate, as shown in Figure 27 on page 38.

Figure 27 Public Certificates for "environment_ks_store" window



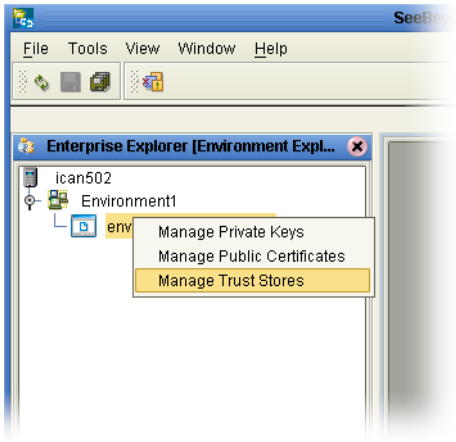
6 Click the **Import** button. A Message window appears confirming the import.

Figure 28 Import Confirmation Message

To Create a New Truststore:

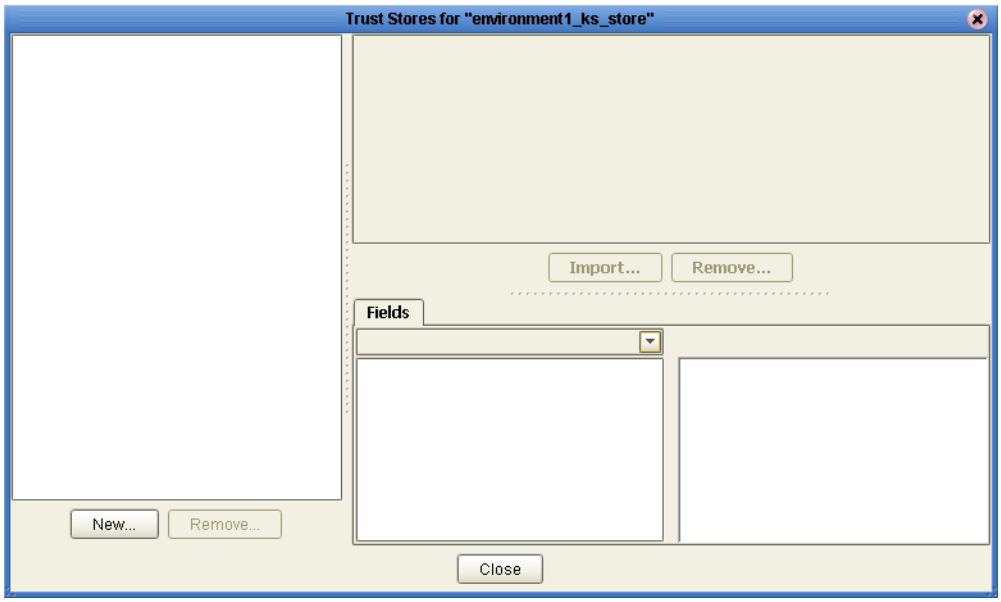
1 Right-click Environment-ks-store and choose **Manage Truststores** from the selection menu.

Figure 29 Manage Truststores



2 The **Truststores for "environment1_ks_store"** window appears.

Figure 30 Truststores for "environment1_ks_store" window



3 Click the **New** button. The **New TrustStore** window appears, as shown in **Figure 31 on page 40**

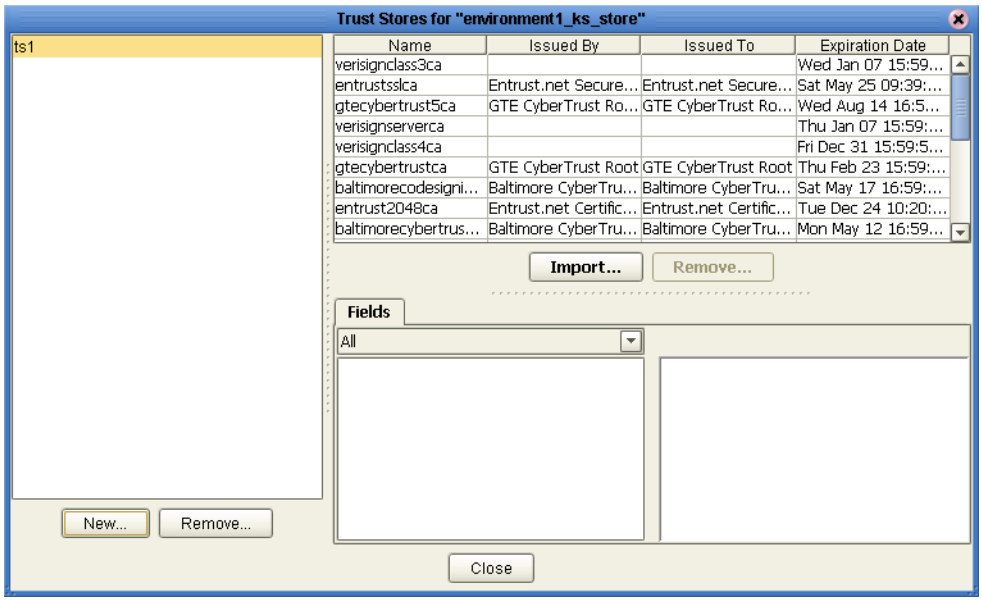
Figure 31 New Truststore

4 Enter an Alias to identify the truststore and click the **OK** button.

5 The **Trust Stores for "environment_ks_store"** window appears.

A number of trust certificates also appears in the right pane. These are industry known Trust Certificates loaded from the JVM of the Enterprise Designer.

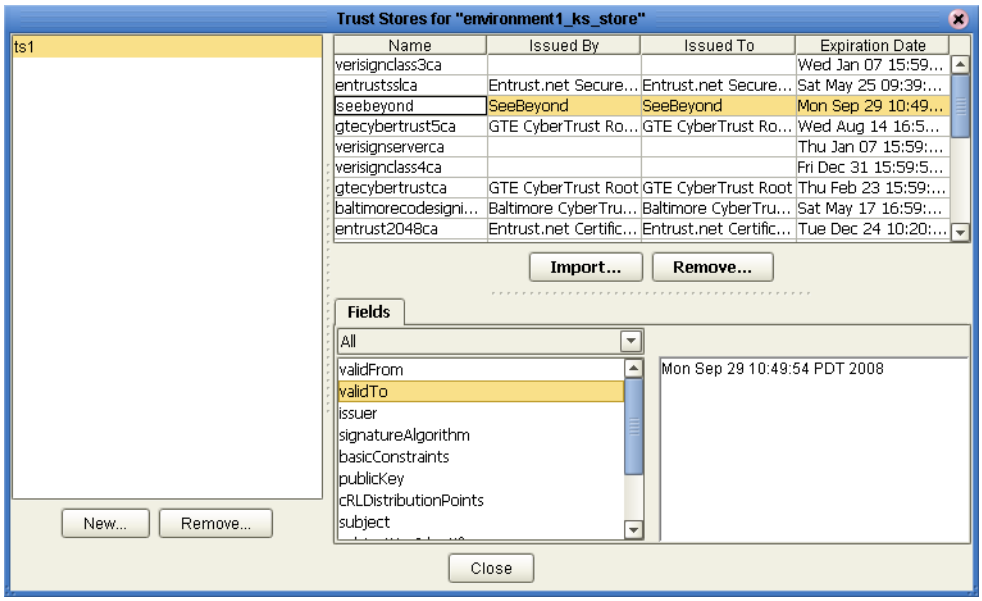
Figure 32 Trust Stores for "environment_ks_store"



To Import a Certificate into a Truststore

1. Click the **Import** button. The **Trust Stores for "environment_ks_store"** window appears.
2. Enter the Alias and File location of certificate and click the **OK** button.
3. A message appears confirming the import. Click the **OK** button.
4. The Manage Truststore Certificate Description window appears, containing the imported certificate.

Figure 33 Truststores with Imported Certificate window



Chapter 5

S/MIME Collaboration Definitions

This chapter lists the various Collaboration Definitions and OTDs used in SME.

SME includes several completed Collaboration Definitions containing the encoded business rules used to compress, decrypt, and create digital signatures.

Every Collaboration Definition is also associated with both an input and an output OTD. The structure and rules defined in each OTD define the necessary data transformations required to complete each function. You select OTDs from the OTD Library, located on the root of the SME node in Enterprise Explorer.

5.1 SME Collaborations

Collaboration Definitions used in SME include:

- **CompressService:** used to compress data
- **DecompressService:** used to decompress data
- **EncryptService:** used to encrypt data
- **DecryptService:** used to decrypt data
- **SignService:** used to electonically sign data
- **VerifySignatureService:** used to verify electonically signed data.

5.2 Available OTDs

Several OTDs are available for use, including:

- SMIMECompressInput_SMIMECompressInput
- SMIMECompressOutput_SMIMECompressOutput
- SMIMEDecompressInput_SMIMEDecompressInput
- SMIMEDecompressOutput_SMIMEDecompressOutput
- SMIMEDecryptInput_SMIMEDecryptInput
- SMIMEDecryptOutput_SMIMEDecryptOutput
- SMIMEEncryptInput_SMIMEEncryptInput
- SMIMEEncryptOutput_SMIMEEncryptOutput
- SMIMESignInput_SMIMESignInput
- SMIMESignOutput_SMIMESignOutput
- SMIMEVerifyInput_SMIMEVerifyInput
- SMIMEVerifyOutput_SMIMEVerifyOutput

Chapter 6

Locating, Importing, and Using Sample Projects

This chapter describes how to use the sample Project included in the installation CD-ROM package.

This Chapter Includes:

Note: *While several key steps are required to create, activate, and deploy a Project, only steps containing information relevant to the SME are included in this chapter. For more detailed information on how to compete a sample Project, see the eGate Integrator Tutorial.*

6.1 Sample Projects Overview

Sample Projects are designed to provide an overview of the security and compression/decompression services offered in SME.

Sample Projects Include:

SME_BPEL_Project – describes how to compress/decompress, sign/verify, and encrypt/decrypt sample data in an eInsight Business Process. For more information, see "Using the Sample Project with eInsight" on page 49.

SME_JCE_Project – describes how to compress/decompress, sign/verify, and encrypt/decrypt sample data using eGate. For more information, see "Using the Sample Project in eGate" on page 58.

6.1.1 Sample Data Used

The data used for the sample Projects are contained within an input file called **SampleData.txt**. See Figure 34 for a description of the data found in the file.

Figure 34 Input Data File

```
start of sample data
AAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFFFFF
GGGGGGGGGGGGGGGGGGGGG
HHHHHHHHHHHHHHHHHHHHH
IIIIIIIIIIIIIIIIIIIII
JJJJJJJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKKKKKKK
LLLLLLLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMMMMMM
11111111111111111111
22222222222222222222
33333333333333333333
44444444444444444444
55555555555555555555
66666666666666666666
77777777777777777777
88888888888888888888
99999999999999999999
00000000000000000000
end of sample data
```

6.1.2 Data Input Parameters

The following tables detail input requirements for the encryption, decryption, sign, and verify processes. You enter these format requirements when you are creating your business rules.

Encryption/Decryption Parameters

For more information on how these requirements are used in the encryption and decryption process, see **Figure 1 on page 13**.

Table 4 SME Encryption Input Parameters

Requirement	Valid Values
Data Entry Type	**bytes**
Public Certificate Alias (required entry)	any alphanumeric
Message Format	**PKCS7** or **SMIM**E (SMIME is the default)
Encoding Format	**binary** or **base64** (base64 is the default)
Encryption Algorithm	**RC2** or **DES3** (DES3 is the default)

Table 5 SME Decryption Input Parameters (continued)

Requirement	Valid Values
Private Key alias (required entry)	any alphanumeric
Password of the key (required entry)	any alphanumeric
Encoding Format	**binary** or **base64** (base64 is the default)
Message Format	**PKCS7** or **SMIME** (SMIME is the default)
Encryption Algorithm	**RC2** or **DES3** (DES3 is the default)

Sign/Verify Parameters

For more information on how these requirements are used in the encryption and decryption process, see **Figure 2 on page 14**.

Table 6 SME Sign Input Parameters

Requirement	Valid Values
Data Entry Type	**bytes**
Public certificate alias (required entry)	any alphanumeric
Password (required entry)	any alphanumeric
Message Format	**SMIME** or **PKCS7** (PKCS7 is the default)
Detach Signature	**boolean** (true/false) If true, the original data is not part of signed data. If false. If false, the original data is embedded in the signed data.

Table 7 SME Verify Input Parameters

Requirement	Valid Values
Sender's Alias (required entry)	any alphanumeric
Detached Signature	**boolean** (true/false) If true, the original data is not part of signed data. If false. If false, the original data is embedded in the signed data.
Signed Message Format MIME	**boolean** (true/false) if true, the signed data format is MIME. If false, the signed data format is PKCS7.

6.1.3 Data Conversion Limitations

The following lists data conversion limitations known in SME.

eInsight

Not all data types are converted reliably in eInsight. Currently, SME expresses data internally in binary format, and the output from SME compression services is also binary, but the output from SME encryption service can either be binary or base64. Unlike base64 which can reliably convert between text and binary data types, binary data converted into text and then back to binary may become corrupted. This means that data produced by compression, or encryption using a binary encoding format must be handled as binary data. When using SME with eInsight, you can pass binary data directly between the SME java collaborations, but not write out or pass between business processes.

eGate

When exporting or importing SME objects in a JCE collaboration, you must first select the input and output methods for reading and writing byteArray (file eWay writeBytes or JMS Bytes messages). Binary data corruption can occur if a File eWay is used in text mode or if writing to a JMS text message requires converting the byte data into a Java string. In these cases, you can treat the Base64 format as text and then convert between java byte and Java string without affecting the data content.

6.2 Locating and Importing the Sample Projects

Sample Projects are included in the **SMEWebServicesDocs.sar**. This file is uploaded separately from the SME sar file during installation. For information, refer to "Installing eGate" on page 16.

Once you have uploaded the **SMEWebServicesDocs.sar** to the Repository, you can begin downloading the sample Projects using the **DOCUMENTATION** tab in the Enterprise Manager to a folder.

Before you can use the sample Project, you must first import it into the SeeBeyond Enterprise Designer using the Enterprise Designer Project Import utility.

To Import the Sample Project

1. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import**.
2. In the **Import Manager** window, browse to the directory that contains the sample Project zip file.
3. Select the sample file and then click **Open**.
4. Click the **Import** button. If the import was successful, then click the **OK** on the **Import Status** window.

5 Close the Import Manager window.

6.3 Running the Sample Projects

Steps required to run a sample Project include:

- Setting the Properties
- Creating the Environment Profile
- Deploying the Sample Project
- Running the Sample Project

6.3.1 Setting the Properties

The sample Project uses both an inbound and an outbound File eWay.

To Configure the File eWays:

1 On the Connectivity Map canvas double-click the **Inbound File eWay**.

2 Select **Inbound File eWay** in the **Templates** window and click **OK**.

3 The **Properties** window for the Inbound File eWay opens. Modify the parameter settings for your system. Change the Directory and Input file name to match the location and name of the sample data file.

4 Click **OK** to close the **Properties** window.

5 On the Connectivity Map, double-click the Outbound File eWay, select **Outbound File eWay** in the templates dialog box and click **OK**.

6 The **Properties** window for the Inbound File eWay opens. Modify the required parameter settings for your system, including the target Directory and Output file name.

7 Click **OK** to close the Properties window.

6.3.2 Creating the Environment Profile

Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. Environments are created using the Enterprise Designer's Environment Explorer and Environment Editor.

For instruction on creating an Environment, see the *eGate Integrator Tutorial*.

6.3.3 Deploying the Sample Project

A Deployment Profile is used to assign Collaborations and message destinations to the integration server and message server. Deployment Profiles are created using the Deployment Editor.

For instruction on creating and activating a Deployment Profile, see the *eGate Integrator Tutorial*.

6.3.4 Running the Sample Project

For instruction on running a sample Project, see the *eGate Integrator Tutorial*.

6.4 Using the Sample Project with eInsight

This section describes how to use the **SME_BPEL_Project** with the ICAN Suite's eInsight Business Process Manager and it's Web Services interface. This section does not provide an explanation of how to create a Project that uses a Business Process. For these instructions, you should refer to the eInsight Enterprise Bus User's Guide.

Before running a sample Project using eInsight, you must:

- Import the sample Project (see "Locating and Importing the Sample Projects" on page 47)
- Configure the File eWays
- Create an Environment for the sample Project
- Create a Deployment Profile

6.4.1 The eInsight Engine and Components

You can deploy an eGate component as an Activity in an eInsight Business Process. Once you have associated the desired component with an Activity, the eInsight engine can invoke it using a Web Services interface. Examples of eGate components that can interface with eInsight in this way are:

- Java Messaging Service (JMS)
- Object Type Definitions (OTDs)

- An eWay
- Collaborations

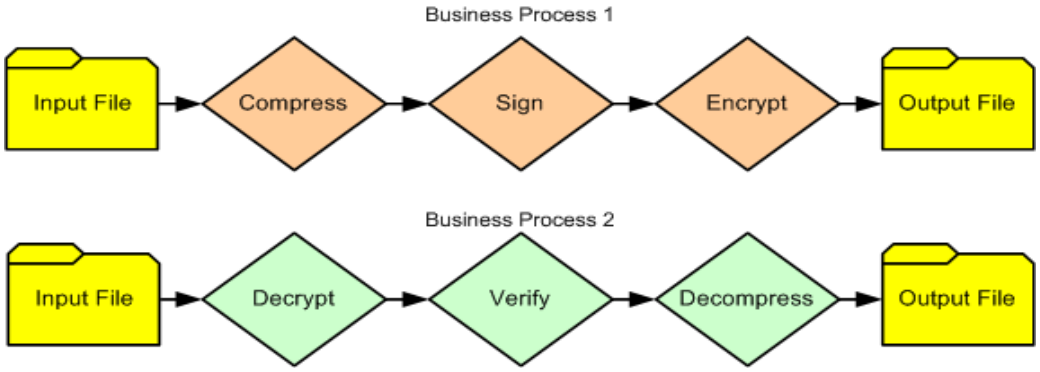
Using the eGate Enterprise Designer and eInsight, you can add an Activity to a Business Process, then associate that Activity with an eGate component. When eInsight runs the Business Process, it automatically invokes that component via its Web Services interface.

6.4.2 The SME_BPEL_Project Sample Project

The **SME_BPEL_Project** contains two business processes. The first business process is designed to compress, sign, and encrypt data from an input file, while the second business process performs a decrypt, verify, and decompress before writing the data to an output file.

The figure below shows the business process used by the sample Project.

Figure 35 SME BPEL Business Processes



6.4.3 Sample Project Business Process

Creation of a business process includes:

- Dragging and dropping business process activities from the Project explorer tree to the eInsight Business Process Designer's modeling canvas.
- Connecting logical business activities together.
- Adding business rules between activities.

Figure 36 on page 51 illustrates a completed business process, containing the compress, sign and encrypt service.

Figure 36 Example Business Process 1

Table 37 on page 51 illustrates a completed business process, containing the decrypt, verify and decompress service.

Figure 37 Example Business Process 2

6.4.4 Business Process Activities

An eInsight Business Process Activity can be associated with the SME web service during system design phase. To make this association, select the desired operators under SME in the Enterprise Explorer and drag it onto the eInsight Business Process Designer canvas.

The SME has the following operators available:

- receive
- compress
- decompress
- sign
- verify
- encrypt
- decrypt
- write

The operation is automatically changed to an Activity with an icon identifying component that is the basis for the Activity. At run time, eInsight invokes each the order defined in the Business Process. Using eInsight's Web Services interface, Activity in turn invokes the SME web service operators.

6.4.5 Configuring the Modeling Elements

Business rules are defined and configured between the business process activities located on the modeling canvas. The sample Project **SME_BPEL_Project** contains business rules between each of the activities listed in the business process flow.

Note: *A detailed description of the steps required to configure modeling elements are found in the eGate Integrator's User's Guide.*

During the first business process, the sample Project:

- Receives a text data; converts and compresses
- Accepts signature input parameters
- Accepts encryption input parameters
- Writes data to a text file

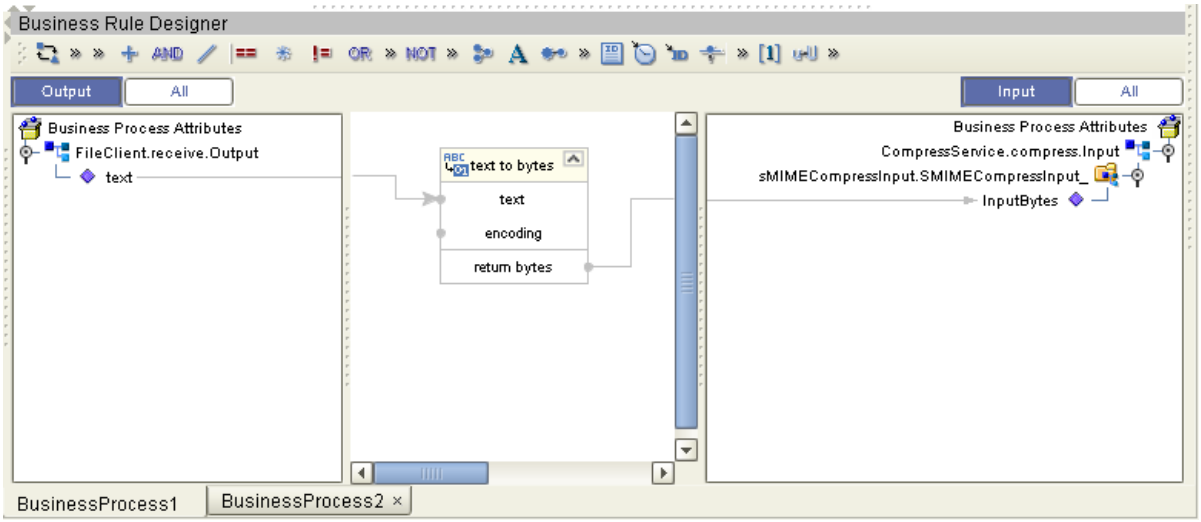
During the second business process, the sample Project:

- Receives text data; accepts decryption input parameters
- Accepts verify (public certificate) input parameters
- Decompresses data
- Writes data to a text file

Converting and Compressing Data

Data is first received in text format, then converted to byte format. Data is compressed using the SMIMECompressInput OTD.

Figure 38 Converting and Compressing Data

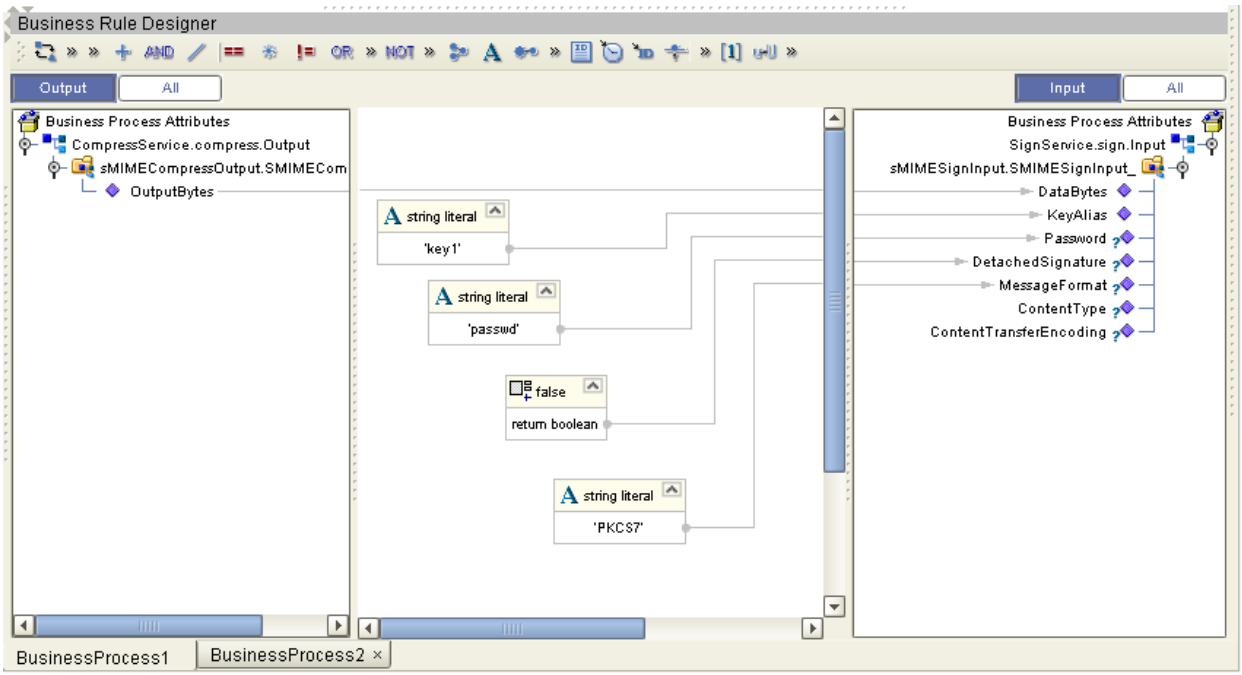


Signing the Data

To sign the data, the SMIMESignInput OTD accepts the compressed data along with the following input string literals:

- The sender's private key (key 1)
- The sender's password (passwd)
- The certificate format (PKCS7)

Figure 39 Signing the Data

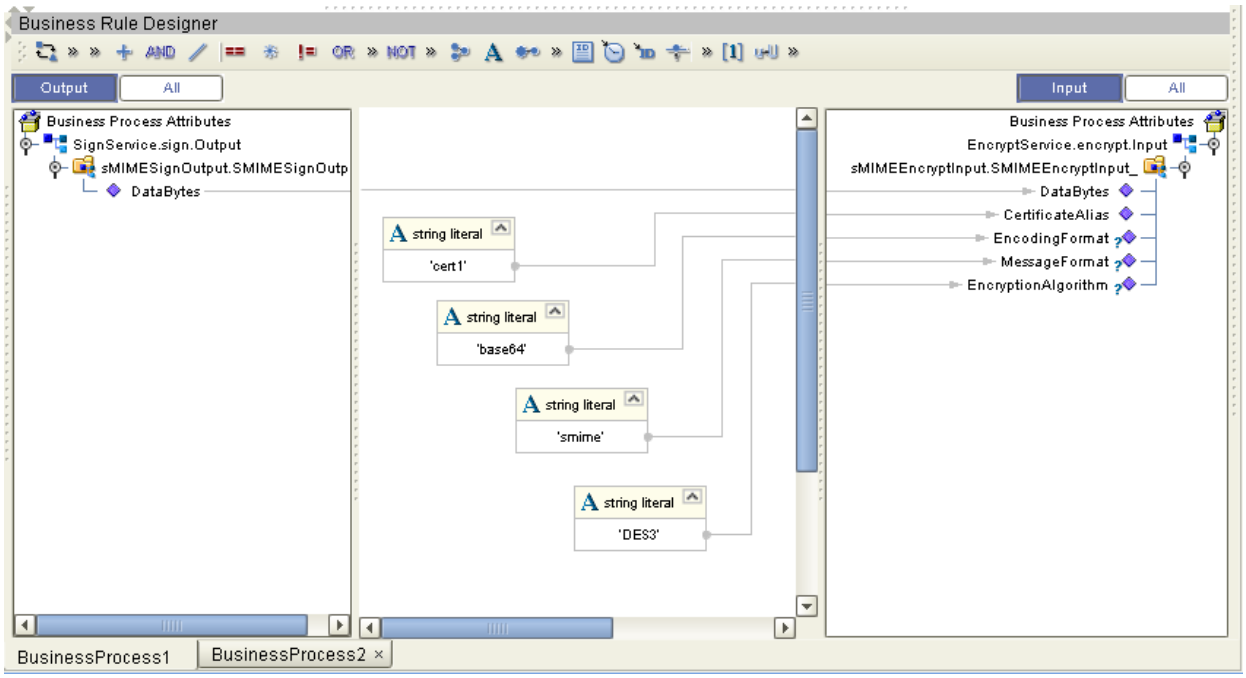


Encrypting the Data

To encrypt the data, the SMIMEEncryptionInput OTD accepts the signed data, along with the following input string literals:

- The public certificate alias (cert 1)
- The message format (smime)
- The encoding format (base64)
- The Encryption algorithm (DES3)

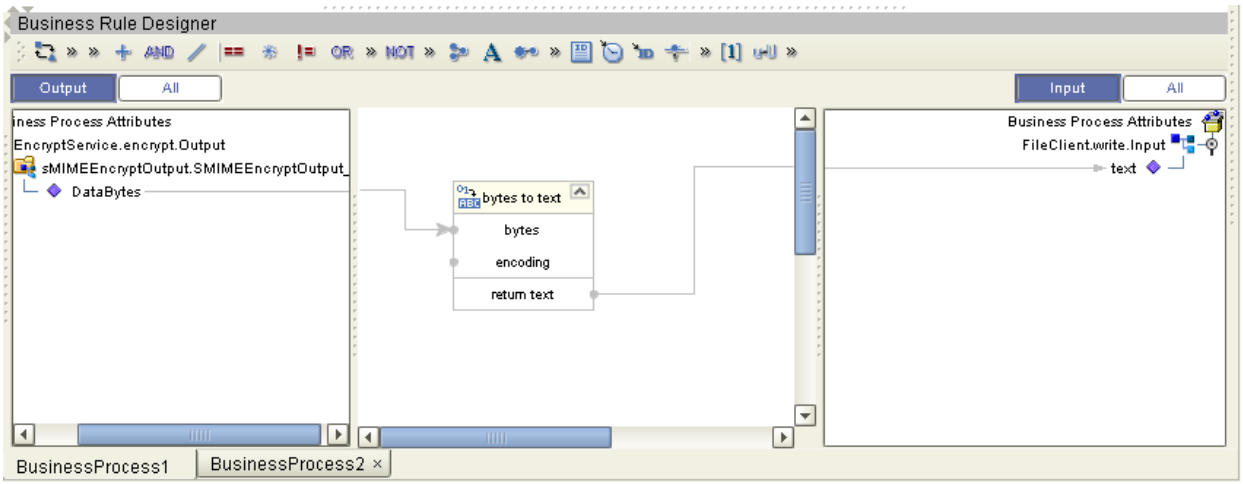
Figure 40 Encrypting the Data



Write Data to an Input File

An input file is required to create objects shareable between the business processes.

Figure 41 Writing Data to a FileClient.write.Input OTD

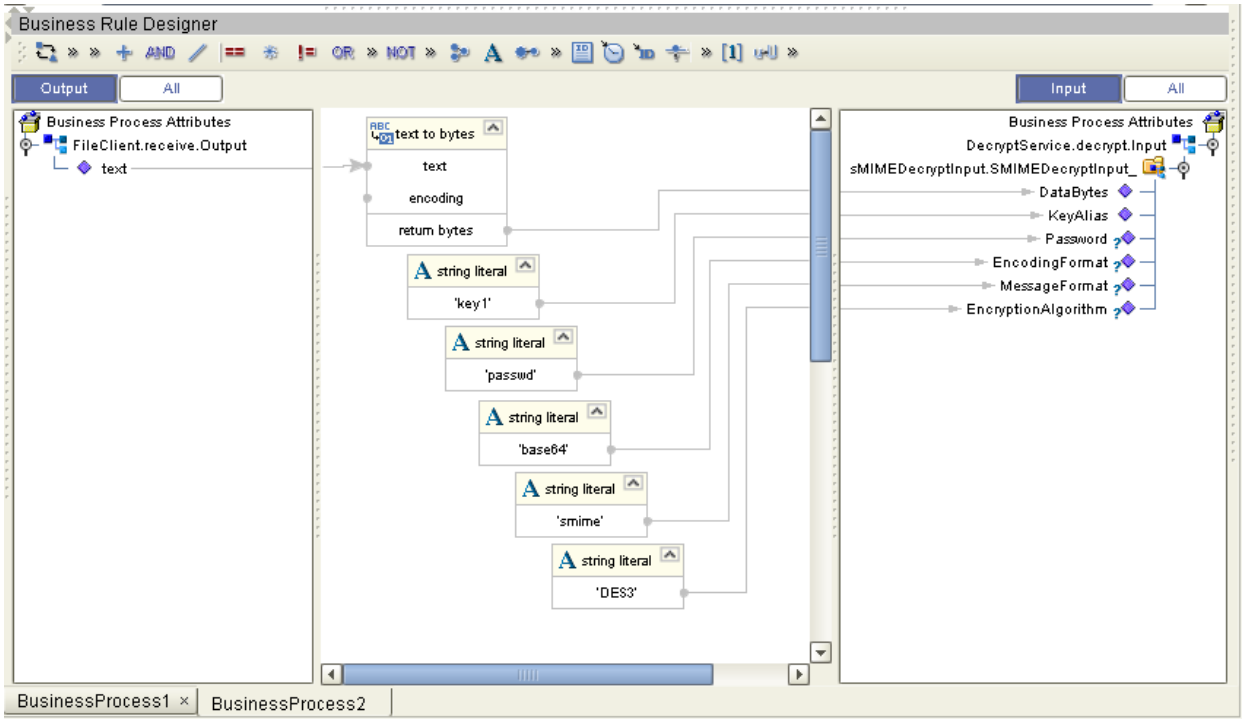


Gathers and Decrypts Data

To decrypt data, the SMIMEDecryptInput OTD accepts data from the FileClient.write.Input OTD along with the following input string literals:

- The private key alias (key 1)
- The password of the key (passwd)
- The message format (smime)
- The encoding format (base64)
- The Encryption algorithm (DES3)

Figure 42 Decrypting Data

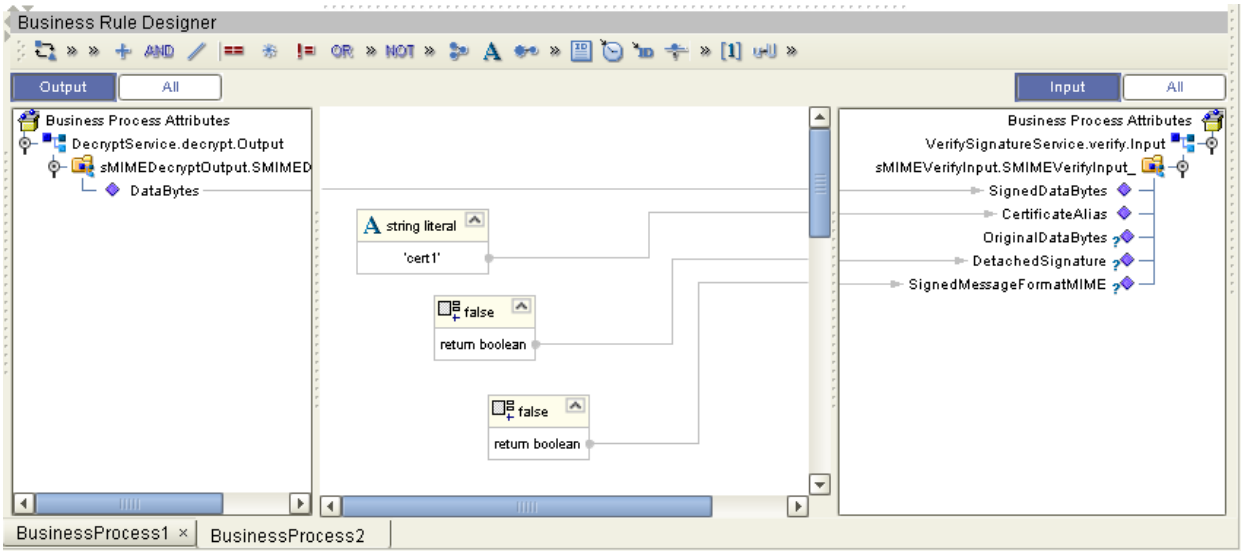


Verify the Signature

To verify the signature, the SMIMEVerifyInput OTD accepts the decrypted data, along with the following input string literals:

- The sender's public certificate

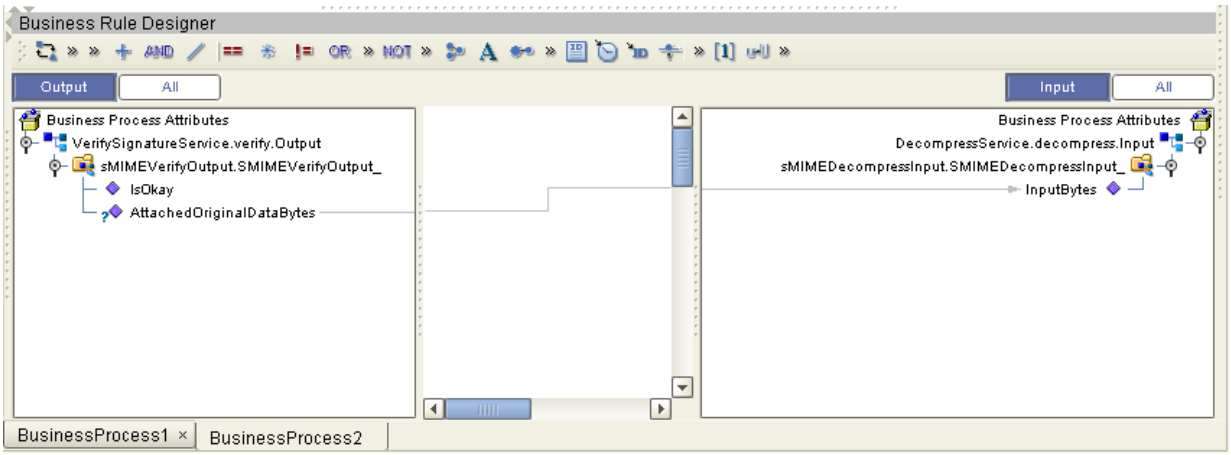
Figure 43 Verifying the Signature



Decompress the Data

To decompress the data, the SMIMEDecompressInput OTD accepts the verified data.

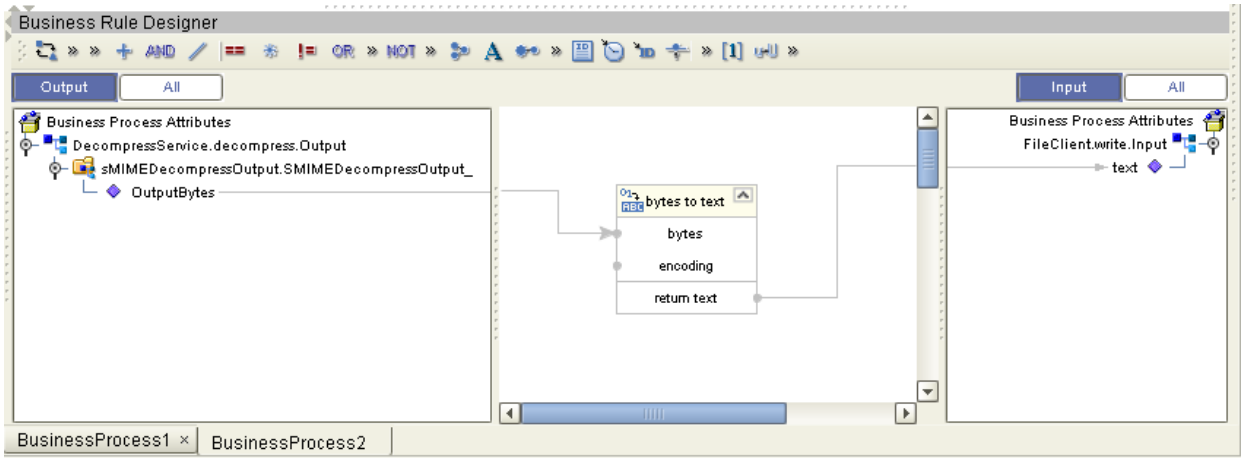
Figure 44 Decompress the Data



Write Data to a Text File

In the final step, data is converted from bytes to text, then sent to the FileClient.write.Input OTD.

Figure 45 Write Data to a Text File



6.5 Using the Sample Project in eGate

This section describes how to use the **SME_JCE_Project** with eGate. This section does not provide an explanation of how to *create* a Project. For these instructions, you should refer to the eGate Integrator Tutorial.

Before running a sample eGate Project, you must:

- Import the sample Project (see **Locating and Importing the Sample Projects** on page 47)
- Configure the File eWays
- Configure the JMS Clients
- Create an Environment for the sample Project
- Create a Deployment Profile

6.5.1 Working with the Sample Project in eGate

The eGate sample Project **SME_JCE_Project** contains two Connectivity Maps (CMap1 and CMap2). The first Connectivity Map contains a process that compress, encrypts, and signs sample data, while the second Connectivity Map contains a process that decrypts, decompresses, and verifies the data.

The Connectivity Maps for these samples appear as follows:

Figure 46 Connectivity Map - Encryption

Figure 47 Connectivity Map - Decryption

6.5.2 Configuring the File eWays

The sample uses an inbound and an outbound File eWay. To configure the sample Projects, use the following information.

1. Double-click the **Inbound File eWay**, select **Inbound File eWay** in the Templates dialog box and click **OK**.
2. The **Parameters** dialog box opens to the Inbound File eWay configuration. Modify the configuration for your system, including the settings for the **Inbound** File eWay in Table 8, and click **OK**. The configuration settings are saved for the eWay.

Table 8 Inbound File eWay Settings

Inbound eWay Connection Parameters	
Directory	C:/temp
Input file name	Input*.txt

3. In the same way, modify the **Outbound** File eWay configuration for your system, including the settings in Table 9, and click **OK**.

Table 9 Outbound File eWay Settings

Outbound eWay Connection Parameters	
Directory	C:/temp
Output file name	output%.txt

Chapter 5

Using SME Java Methods

The SME exposes various Java methods to add extra functionality, making it easier to set, and get information in the SME OTDs. For a complete list of the Java methods within the classes listed below, refer to the **Javadoc**.

- SMEUtil
- SMIMECompressor
- SMIMEDecompressor
- SMIMEDecryptor
- SMIMEEncryptor
- SMIMESignatureVerifier
- SMIMESigner

You can find the **Javadoc** in the **SMEWebServicesDocs.sar** file. For complete instructions, see the *ICAN Installation Guide*.

Index

S

T

U

W

X