Secure Messaging Extension
User’s Guide

Release 4.5.2

Monk Version
# Contents

## Chapter 1

**Introduction** 5

- Overview 5
  - Intended Reader 5
  - Components 5
- Introducing S/MIME 6
- Introducing Secure Messaging Extension 7
- Supported Operating Systems 10
- System Requirements 10
  - Environment Variable Settings 10

## Chapter 2

**Installation** 11

- Windows 11
  - Pre-installation 11
  - Installation Procedure 11
- UNIX 12
  - Pre-installation 12
  - Installation Procedure 12
- Files/Directories Created by the Installation 13

## Chapter 3

**Implementation** 14

- Sample Configuration 14
- Sample Implementation 15
- Sample Monk Scripts 15
- Certificate Formats 15
Chapter 1

Introduction

This document describes how to install and configure the Secure Messaging Extension.

1.1 Overview

The Secure Messaging Extension enables e*Gate to process Events utilizing the S/MIME (Secure Multipurpose Internet Mail Extensions) message format. The Secure Messaging Extension supports encryption, decryption and authentication of messages and is interoperable with any other client applications that support the S/MIME standard.

This adds the following features to a transaction:

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

1.1.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows NT and/or UNIX operations and administration; and to be thoroughly familiar with Windows-style GUI operations.

1.1.2 Components

The following components comprise Secure Messaging Extension:

- Secure Messaging Extension Monk function scripts
- Dynamically loaded libraries (DLLs)
- Monk collaborations that load and call their functions.

A complete list of the installed files appears in Table 1 on page 13.
1.2 Introducing S/MIME

Secure Multipurpose Internet Mail Extension (S/MIME) is a type of MIME message format that supports digital signatures and encryption of messages. The new form is based on the public-key encryption technology created by RSA Data Security, Inc. The RSA algorithm is based on the fact that there is no efficient way to factor very large numbers, making nearly it impossible to derive the private key based solely on the public key.

The public-key encryption system uses two keys: a public key known to everyone and a private key known only to the recipient of the message. For example, if JaneDoe wants to send a secure message to JohnDoe, JaneDoe uses JohnDoe’s public key to encrypt the message. JohnDoe then utilizes his own private key to decrypt it.

Only the public key can be used to encrypt messages, and only the corresponding private key can be used to decrypt them (and vice versa).

Similarly, the sender utilizes his/her private key to include a signature on the message, while the recipient uses the sender’s public key to verify the signature.

MIME offers a standardized way to represent and encode a wide variety of media types for transmission via the internet. Many e-Mail clients now support MIME, which enables sending and receiving of graphics, audio, and video files via the Internet. MIME also supports messages in character sets other than ASCII.

When using MIME, messages can contain the following data types:

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

With the implementation of S/MIME, the protocol is available that adds digital signatures and encryption to these messages. These messages consist of two parts: the header and the body. The header forms a collection of field/value pairs structured to provide information necessary for the transmission of the message. MIME defines how the body of a message is structured. This format permits the inclusion of the above mentioned datatypes in a standardized manner. S/MIME defines the security services, adding digital signatures and encryption, thus preventing forgery and interception.

1.3 Introducing Secure Messaging Extension

The Secure Messaging Extension provides security features, allowing the secure transmission of exchanges over public domains such as the Internet. Secure Messaging Extension adds the ability to use Public Key Infrastructure (PKI) technology to ensure the confidentiality of exchanges by digitally signing and encrypting messages as they are sent, and decrypting and authenticating messages when they are received.

The Secure Messaging Extension performs the encryption and decryption of messages using the S/MIME standard. The standard one-way hash algorithms ensure data integrity by verifying that no modifications are made to the message while in transit. The identity of the sender of a message is verified through the use of digital signatures, proving that the message actually originated from the entity who claims to have sent it.

The following flowcharts show the processing of the data from receipt to destination.
Figure 1 Inbound Signed/Encrypted Message

1. Receives an Inbound Message
2. **Encrypted Event?**
   - No: Proceed to next step.
   - Yes: Separate Block encrypted Event from PKI encrypted Session Key.

3. **Encryption required for this Event?**
   - No: Forward Event body to destination.
   - Yes: Raise exception.

4. **Signed Event?**
   - No: Forward Event body to destination.
   - Yes: Separate the digital signature and originator's Public Key from the body of the Event.

5. Hash Event body.

6. Retrieve Public Key from PKI Database.

7. Decrypt digital signature using Public Key.

8. **Decrypted digital signature = hashed Event body?**
   - No: Forward Event body to destination.
   - Yes: Retrieve Receiver's Private Key from PKI Database.

9. Decrypt Session Key using Private Key.

10. Decrypt Event using Session Key.

11. Forward Event body to destination.

12. Raise exception.
Figure 2  Outbound Signed/Encrypted Message

1. Receives Outbound Event/Message
2. Sign Event?
   - No
   - Yes → Hash Event
3. Retrieve Private Key from PKI Database
4. Encrypt hashed Event using originator’s Private Key to create a digital signature
5. Add digital signature & originator’s Public Key to Outbound Event
6. Encrypt Event?
   - No
   - Yes → Randomly generate Session Key
7. Use Session Key to block encrypt the Event
8. Retrieve Private Key from PKI Database
9. Encrypt Session Key with Partner’s Public Key Certificate
10. Add PKI encrypted Session Key to Block encrypted Event
11. Forward Outbound Event to destination
1.4 Supported Operating Systems

The Secure Messaging Extension (Java) is supported on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- HP-UX 11.0 and HP-UX 11i

1.5 System Requirements

To use the Secure Messaging Extension, you need the following:

- An e*Gate Participating Host, version 4.5.1 or higher.
- 9 MB free disk space on both the Participating and the Registry Hosts.

**Note:** Additional disk space will be required to process and queue the data that the Secure Messaging Extension processes; the amount necessary will vary based on the type and size of the data being processed, and any external applications performing the processing.

- A TCP/IP network or other network connection.
- A fast CPU, if secure message volume is expected to be high. (The public-key operations associated with encryption and signing are computationally expensive.)

1.5.1 Environment Variable Settings

The user must set the MONK_LIBRARY_PATH environment variable.

If using bash:

```bash
export MONK_LIBRARY_PATH=/home/someuser/egate/client/bin
```

or if using csh or tcsh:

```csh
setenv MONK_LIBRARY_PATH /home/someuser/egate/client/bin
```
Chapter 2

Installation

This chapter describes how to install the Secure Messaging Extension.

2.1 Windows

2.1.1 Pre-installation

- Exit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this extension.

2.1.2 Installation Procedure

To install the Secure Messaging Extension on a Windows system:

1. Log in as an Administrator on the workstation on which you want to install the extension.
2. Insert the installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive’s “Autorun” feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel’s Add/Remove Applications feature to launch the file setup.exe on the CD-ROM drive.
4. The InstallShield setup application will launch. Follow the on-screen instructions to install the extension.

Note: Be sure to install the extension files in the suggested “client” installation directory. The installation utility detects and suggests the appropriate installation directory. Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested “installation directory” setting.
2.2 UNIX

2.2.1 Pre-installation

- You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the Secure Messaging Extension on a UNIX system:

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type
   
   cd /cdrom

4. Start the installation script by typing:
   
   ./setup.sh

5. A menu of options will appear. Select the “install e*Way” option. Then, follow any additional on-screen directions.

Note: Be sure to install the extension files in the suggested “client” installation directory. The installation utility detects and suggests the appropriate installation directory. Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested “installation directory” setting.
Chapter 2
Installation

Section 2.3
Files/Directories Created by the Installation

2.3 Files/Directories Created by the Installation

The Secure Messaging Extension installation process will install the following files within the e*Gate directory tree. Files will be installed within the “egate\client” tree on the Participating Host and committed to the “default” schema on the Registry Host.

Table 1 Files created by the installation

<table>
<thead>
<tr>
<th>e*Gate Directory</th>
<th>File(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin\</td>
<td>stc_monksmime.dll</td>
</tr>
<tr>
<td>bin\AIX\</td>
<td>libsmt.a</td>
</tr>
<tr>
<td></td>
<td>libldapssl30.so</td>
</tr>
<tr>
<td>bin\HP-UX\</td>
<td>libsmt.sl</td>
</tr>
<tr>
<td>bin\Solaris\</td>
<td>libsmt.so</td>
</tr>
<tr>
<td>bin\Win32\</td>
<td>smt32.dll</td>
</tr>
<tr>
<td></td>
<td>nsldap32vll.dll</td>
</tr>
</tbody>
</table>
Implementation

This chapter includes information pertinent to implementing the Secure Messaging Extension in a production environment.

3.1 Sample Configuration

The sample on the CD demonstrates a simple S/MIME execution:

- Load S/MIME extension
- Define test for error
- Display the encryption and/or signature algorithms returned by the sign-encrypt and verify-decrypt functions
- Read sample files into memory
- Set preliminary variables and begins the session
- Import keys and certificates into cache
- Authenticate the signature of an inbound message
- Attempt to authenticate the signature of a corrupt inbound message
- Encrypt an outbound message
- Sign an outbound message
- Decrypt an inbound message
- Decrypt an inbound message that has been corrupted
- Decrypt and authenticate an inbound message
- Fail to decrypt and authenticate a corrupt inbound message
3.2 Sample Implementation

A sample Monk script for setting up Secure Messaging Extension is provided on the e*Gate installation CD-ROM in the directory samples/ewsmime

1 Copy all the files from the e*Gate installation CD-ROM samples directory samples/ewsmime to a temporary working directory on the system on which the Secure Messaging Extension is installed.

3.3 Sample Monk Scripts

The samples on the CD can be run using the stctrans command-line utility once they are copied onto a working directory. They do not require a complete e*Gate schema configuration to function, and are designed to illustrate the principles involved in creating your own custom Monk scripts. The library (dll) files to be loaded and the script to be tested must be in the load path (or, for simplicity’s sake, may be placed in the connected directory). See the Monk Developer’s Reference for more information about the load path.

The syntax of the stctrans utility is

stctrans monk_file.monk

Additional command-line flags are available; enter stctrans -h to display a list, or see the e*Gate Integrator System Administration and Operations Guide for more information.

3.4 Certificate Formats

The SMIME/C library accepts certificates in PKCS#7 format. DER encoded binary X.509 and Base64 encoded X.509 format certificates are also popular.

Windows 2000 and Internet Explorer provide a tool to transfer between formats. To change formats, perform the following:

1 On Windows 2000, double click the certificate file.
2 Select the Detail tab.

**Figure 4** Windows 2000 Detail Tab

3 Click on “copy to file” button.
4 Click Next. Choose the format.

For Internet Explorer:

1 Select Tools-->Internet option.
2 Choose Content Tab, click on Certificates.
3 Click on Import, to import your certificate.

4 Click on “Intermediate Certification Authorities” Tab to choose the certificate to import, and click on the “export” button.

5 Select the format, and save the file.
This chapter details the Secure Messaging Extension Functions. Prior to use it is necessary to load the extension into any Monk environment:

```
(define SMIMEH (load-interface "stc_monksmime.dll" "init_smimeext"))
```

SMIMEH is then a handle used for all subsequent method calls.

### 4.1 Secure Messaging Extension Functions

The current suite of Secure Messaging Extension functions are:

- **begin-session** on page 19
- **db-store** on page 21
- **set-param** on page 24
- **sign-encrypt** on page 26
- **verify-decrypt** on page 28
- **wipe-handle** on page 30

---

**begin-session**

**Syntax**

```
(SMIMEH "begin-session")
```

**Description**

*begin-session* prepares the SMIMEH handle to accept further instructions. It requires that “db-pathname” and “username” parameters be set prior to calling, and is itself required before calling any “db-store”, “sign-encrypt”, or “verify-decrypt” functions.

**Parameters**

None.

**Return Values**

- **vector**
  - Returns a two element vector:
db-query

Syntax

(SMIMEH "db-query" key-cert-type key-cert-name)

Description

db-query tests whether it is necessary to use “db-store” to add a key or certificate.

Parameters

<table>
<thead>
<tr>
<th>key-cert-type</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;signature-key&quot;</td>
<td>string</td>
<td>The binary contents of the string signature-key in PKCS#12 format.</td>
</tr>
<tr>
<td>&quot;decryption-key&quot;</td>
<td>string</td>
<td>The binary contents of the string decryption-key in PKCS#12 format.</td>
</tr>
<tr>
<td>&quot;CA-cert&quot;</td>
<td>string</td>
<td>The binary contents of the string CA-cert in PKCS#7 format.</td>
</tr>
<tr>
<td>&quot;signature-cert&quot;</td>
<td>string</td>
<td>The binary contents of the string signature-cert in PKCS#7 format.</td>
</tr>
</tbody>
</table>
Secure Messaging Extension User's Guide

Return Values

vector

Returns a two element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if found, otherwise (false).</td>
</tr>
<tr>
<td>1</td>
<td>string error message,</td>
<td>begin-session has not been called or failed.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_INVALID_PARAMETER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMT_E_CALL_FAILED</td>
<td>Internal error encountered, possibly due to missing or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corrupt</td>
</tr>
<tr>
<td></td>
<td>SMT_E_FILENOTFOUND</td>
<td>Entry does not exist in the data source.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_ENOUGH_MEMORY</td>
<td>Fail to allocate sufficient memory.</td>
</tr>
</tbody>
</table>

Additional Information

The function takes one of the parameters from the key-cert-type column.

(db-query key-cert-type key-cert-name)

See Also

begin-session on page 19 must be called before the query.

db-store

Syntax

(SMIMEH "db-store" key-or-cert-type key-or-cert-name key-or-cert-data trust-model)

Description

db-store loads X.509 certificates and keys into the cache for future use.
Parameters

<table>
<thead>
<tr>
<th>key-or-cert-type</th>
<th>Value Type</th>
<th>key-or-cert-data</th>
</tr>
</thead>
<tbody>
<tr>
<td>“signature-key”</td>
<td>string</td>
<td>The binary contents of the string signature-key in PKCS#12 format.</td>
</tr>
<tr>
<td>“decryption-key”</td>
<td>string</td>
<td>The binary contents of the string decryption-key in PKCS#12 format.</td>
</tr>
<tr>
<td>“CA-cert”</td>
<td>string</td>
<td>The binary contents of the string CA-cert in PKCS#7 format.</td>
</tr>
<tr>
<td>“signature-cert”</td>
<td>string</td>
<td>The binary contents of the string signature-cert in PKCS#7 format.</td>
</tr>
<tr>
<td>“encryption-cert”</td>
<td>string</td>
<td>The binary contents of the string encryption-cert in PKCS#7 format.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key-or-cert-name</td>
<td>string</td>
<td>The exact display name of the key or certificate being loaded.</td>
</tr>
<tr>
<td>key-or-cert-data</td>
<td>string</td>
<td>The binary representation of the key or certificate to store.</td>
</tr>
<tr>
<td>trust-model</td>
<td>string</td>
<td>Required when selecting signature-cert or encryption-cert. Valid options are CA-trust or direct-trust.</td>
</tr>
</tbody>
</table>

Return Value

vector

Returns a two element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if successful, otherwise (false).</td>
</tr>
</tbody>
</table>
### Secure Messaging Extension User’s Guide

**Chapter 4**

**Secure Messaging Extension Functions**

#### Section 4.1

**Throws**

None.

**Location**

stc_monksmime.dll

**Additional Information**

The function takes one of the parameters from the key-cert-type column.

(db-store key-or-cert-type key-or-cert-name key-or-cert-data trust-model)

**See Also**

- **begin-session** on page 19 must be called before using the **db-store** api.

---

**end-session**

**Syntax**

```
(SMIMEH "end-session")
```

**Description**

**end-session** closes the SMIMEH handle created by **begin-session**.

**Parameters**

None.
Return Values

vector
Returns a two element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
</tr>
<tr>
<td>1</td>
<td>string error message, if applicable.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_INVALID_PARAMETER</td>
</tr>
<tr>
<td></td>
<td>begin-session has not been activated or failed.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_CALL_FAILED</td>
</tr>
<tr>
<td></td>
<td>Function unsuccessful. An I/O error occurred during the persistent saving of the user's security preferences. Check disk space and file access permissions.</td>
</tr>
</tbody>
</table>

Throws
None.

Location
stc_monksmime.dll

See Also

begin-session on page 19 must be successfully activated. end-session must be called to release the resource for this session.

set-param

Syntax

(SMIMEH "set-param" param-name param-value)

Description

set-param sets the parameter names and values to be used by SMIME for sending and receiving data.

Parameters

<table>
<thead>
<tr>
<th>param-name</th>
<th>Value Type</th>
<th>param-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;db-pathname&quot;</td>
<td>string</td>
<td>The directory in which to store the S/MIME certificate and key cache. The value should not contain illegal characters like: *, ?, ”, &lt;, &lt;, and</td>
</tr>
<tr>
<td>&quot;username&quot;</td>
<td>string</td>
<td>The name of the certificate to be used for signing. The value should not contain illegal characters like: *, ?, ”, &lt;, &lt;, and</td>
</tr>
<tr>
<td>param-name</td>
<td>Value Type</td>
<td>param-value</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>“key-passphrase”</td>
<td>string</td>
<td>The passphrase (password or PIN) used to protect S/MIME key cache. The password must be at least 8 characters in length and no more than 64 characters.</td>
</tr>
<tr>
<td>“encryption-alg”</td>
<td>string</td>
<td>The algorithm-string used for encryption. The possible choices are: “DES_EDE3_CBC”, “RC2_128”, “DES_CBC”, or “RC2_40”.</td>
</tr>
<tr>
<td>“signature-alg”</td>
<td>string</td>
<td>The algorithm-string used to attach the signature. The possible choices are: “RSA_MD5” or “RSA_SHA1”.</td>
</tr>
<tr>
<td>“signature-type”</td>
<td>string</td>
<td>Specifies the signature type. The possible choices are “detached” or “inline”.</td>
</tr>
<tr>
<td>“output-encoding”</td>
<td>string</td>
<td>Specifies the encoding type for the input/output. The possible choices are “base64” (default) or “binary”.</td>
</tr>
<tr>
<td>“recipient-list”</td>
<td>string</td>
<td>A vector of certificate names for one or more recipient identifiers, determines which certificates to use for encryption.</td>
</tr>
<tr>
<td>“sender”</td>
<td>string</td>
<td>Specifies the sender identifier, and determines which certificate to use for decryption.</td>
</tr>
<tr>
<td>“message-type”</td>
<td>string</td>
<td>Specifies the message format for input/output. The possible choices are “smime2” or “pkcs7” (default).</td>
</tr>
<tr>
<td>pkcs12-passphrase</td>
<td>string</td>
<td>The passphrase associated with a signature-key or decryption-key. Can be any length and must be ASCII characters. Must be set when the passphrase is required for a particular signature-key or decryption-key (PKCS12). It should be set just before db-store</td>
</tr>
</tbody>
</table>
Secure Messaging Extension Functions

Chapter 4

Section 4.1

Return Value

vector

Returns a two element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if successful, otherwise (false).</td>
</tr>
<tr>
<td>1</td>
<td>string error message, if applicable. SMT_E_INVALID_PARAMETER</td>
<td>Message object is corrupt.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_ENOUGH_MEMORY</td>
<td>Memory allocation failed.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_FOUND</td>
<td>Properties for message object can not be found.</td>
</tr>
</tbody>
</table>

Throws

None.

Additional Information

The function takes one of the parameters from the param-name column.

(set-param param-name param-value)

Location

stc_monksmime.dll

See Also

Before calling begin-session on page 19, username and db-pathname must be set using set-param.

sign-encrypt

Syntax

(SMIMEH "sign-encrypt" outbound-message)

Description

sign-encrypt signs and/or encrypts outbound-messages. The result is the ciphertext (or signed only) message, stored in vector-ref 2.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outbound message</td>
<td>string</td>
<td>The outgoing message.</td>
</tr>
</tbody>
</table>
Return Value

**vector**

Returns a four element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if successful, otherwise (false).</td>
</tr>
<tr>
<td>1</td>
<td>string error message or output message if no error.</td>
<td>Returns a four element vector:</td>
</tr>
<tr>
<td></td>
<td>SMT_E_INVALID_PARAMETER</td>
<td>begin-session has not been successfully activated or message object is corrupt.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_FOUND</td>
<td>Function did not find properties for this message object.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_ENOUGH_MEMORY</td>
<td>Fail to allocate memory.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_CALL_FAILED</td>
<td>Function could not create an internal object.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_BUSY</td>
<td>Other thread or process are using same resource.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_INVALID_RECIPS</td>
<td>One or more recipients has an untrusted encryption certificate.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NO_KEYS</td>
<td>No private key for authenticating a signature or no certificate.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_FOUND</td>
<td>The recipients for the encrypted message are not specified or properties for the message object can not be found.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_ENOUGH_MEMORY</td>
<td>Memory allocation failed.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_FILENOTFOUND</td>
<td>No recipient exists for encryption.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_CALL_FAILED</td>
<td>Internal call failed or passphrase of signing operation is incorrect.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_UNSUPPORTED_MIME_TYPE</td>
<td>Invalid combinations of settings for MIME tagging. The valid combinations are:</td>
</tr>
<tr>
<td></td>
<td>1 message-type=&quot;smime2&quot; signature-type=&quot;detached&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 message-type=&quot;smime2&quot; signature-type=&quot;inline&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 message-type=&quot;smime2&quot; signature-type=&quot;detached&quot; encoding-type=&quot;binary&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 message-type=&quot;smime2&quot; signature-type=&quot;detached&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 message-type=&quot;pkcs7&quot; signature-type=&quot;detached&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 message-type=&quot;pkcs7&quot; signature-type=&quot;inline&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Throws

None.

Location

stc_monksmime.dll

Additional Information

The current version of the RSA Security libraries that the Secure Messaging Extension uses, requires that a private key be located in the cache before allowing any encryption to occur. The username for encryption must match with the private key’s value for username.

The db-path must be set to:

./tmp/smimecache or monk/smime/tmp/smimecache

See Also

verify-decrypt on page 28, values for “output-encoding”, “message-type” and “signature-type” must be matched with values used for sign-encrypt if they have been assigned.

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>string indicating encryption algorithm used; #f (false) if error or no encryption used.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>string indicating signature algorithm used; #f (false) if error or no signature used.</td>
<td></td>
</tr>
</tbody>
</table>

**verify-decrypt**

Syntax

(MIMEH "verify-decrypt" inbound-message)

Description

verify-decrypt decrypts and/or verifies signature authenticity of the inbound-message. The result is the plaintext message, (upon successful authentication), stored in vector-ref 2.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inbound-message</td>
<td>string</td>
<td>The inbound message previously signed and/or encrypted.</td>
</tr>
</tbody>
</table>

Return Value

vector

Returns a four element vector:
## Secure Messaging Extension Functions

### Chapter 4 Section 4.1

**Throws**

None.

**Location**

stc_monksmime.dll

**See Also**

See `sign-encrypt` on page 26 values for “output-encoding”, “message-type” and “signature_type” must be matched with values used for `verify-decrypt` if they have been assigned.

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if successful, otherwise (false).</td>
</tr>
<tr>
<td>1</td>
<td>string error message or output message if no error.</td>
<td>begin-session has not been successfully activated or message object is corrupt.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_INVALID_PARAMETER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMT_E_NOT_ENOUGH_MEMORY</td>
<td>Fail to allocate memory.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_CALL_FAILED</td>
<td>Function could not create an internal object.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_BUSY</td>
<td>Other thread or process using the same resource.</td>
</tr>
<tr>
<td></td>
<td>SMT_S_FALSE</td>
<td>Function successfully verified the message, but either the address book entry was not added or the certificate trust data store was not updated.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_CORRUPT</td>
<td>The message is corrupt. Ensure the values for “output-encoding” and “message-type” are the same as specified during encryption/signing.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_ERROR</td>
<td>Fail to verify signature because the message has been tampered with.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_NO_KEYS</td>
<td>No private key to decrypt the message.</td>
</tr>
<tr>
<td></td>
<td>SMT_E_FILENOTFOUND</td>
<td>Function could not locate necessary information for the user, possibly as a result of the data store being corrupt.</td>
</tr>
<tr>
<td>2</td>
<td>string indicating encryption algorithm used; #f (false) if error or no encryption used.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>string indicating signature algorithm used; #f (false) if error or no signature used.</td>
<td></td>
</tr>
</tbody>
</table>
wipe-handle

Syntax

(SMIMEH "wipe-handle")

Description

_wipe-handle_ deletes temporary variables stored in the object handle.

Parameters

None.

Return Value

vector

Returns a two element vector:

<table>
<thead>
<tr>
<th>Element</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>#t or #f</td>
<td>(true) if successful, otherwise (false).</td>
</tr>
</tbody>
</table>
| 1       | string error message, if applicable.  
|         | SMT_E_INVALID_PARAMETER         | _begin-session_ has not been successfully 
|         |                                   | activated or message object is corrupt.         |

Throws

None.

Location

stc_monksmime.dll
Index

B
Base64 15
begin-session 19

C
Certificate Formats 15
components 5

D
db-query 20
db-store 21
DER 15

E
end-session 23

F
files/directories created by installation 13
functions
   begin-session 19
db-query 20
db-store 21
end-session 23
set-param 24
sign-encrypt 26
verify-decrypt 28
wipe-handle 30

I
intended reader 5
introducing S/MIME 6

S
S/MIME 6
sample configuration 14
Secure Messaging Extension functions 19
set-param 24
sign-encrypt 26

SMIMEH 19
system requirements 10

U
UNIX 12
   installation procedure 12
   pre-installation 12

V
verify-decrypt 28

W
Windows NT 11
   installation procedure 11
   pre-installation 11
   wipe-handle 30

X
X.509 15