IRONPORT®
SYSTEMS

IronPort Encryption Appliance 6.5
APPLICATION DEVELOPER'S REFERENCE MANUAL

COPYRIGHT

IRONPORT SYSTEMS® , INC.

IronPort Systems, Inc.
950 Elm Ave.
San Bruno, CA 94066

CONTACTING IRONPORT CUSTOMER SUPPORT

If you have purchased support directly from IronPort Systems, you can request support by phone, email, or online 24 hours a day, 7 days a week. During office hours (24 hours per day, Monday through Friday, excluding U.S. holidays), an engineer will contact you within an hour of your request. To report a critical issue that requires urgent assistance outside of our office hours, contact IronPort using the following information.

U.S. toll-free:1 (877) 641-IRON (4766)

International: www.ironport.com/support/contact_support.html

Support Portal: www.ironport.com/support

If you have purchased support through a reseller or other entity, contact the supplier for support of your IronPort products.

Table of Contents

CHAPTER 1

Message Personalization

This chapter describes how to set up email message personalization in the SMTP mail component of the IronPort Encryption appliance.

This chapter contains the following sections:

OVERVIEW

The IronPort Encryption appliance includes a set of template components for producing customized text messages, documents and statements. These mechanisms are primarily focused on plain text and HTML documents and can be extended to include other formats such as PDF.

Template documents are usually defined in text files on disk and can be plain text files, HTML files or XML files as required. Within the template document, certain fields are declared as being substitution variables. These are replaced with the corresponding data value when the template is read. The exact syntax for specifying these variables depends on which template engine is being used.

Each template engine takes two main inputs, one is a template document and the other is a map of variable names to variable values. When these are passed to the template engine, the variable substitution takes place and a fully personalized document is returned.

The data values are provided and set up by the component that calls the template engine. In the case of the mail server component, the Secure Envelope Sender sets up the variable map that is passed to the template engine. Each variable consists of a name and a value, for example:

Variable Name	Variable Value
firstname	John
Email	John.Doe@doecorp.com

As well as simple name/value pairs, some template engines support list values, also known as vectors. The exact syntax for handling list values depends on the template engine being used.

SETTING UP MESSAGE PERSONALIZATION

This section describes how to set up email message personalization in the SMTP mail component. The IronPort Encryption appliance supports the delivery of documents that are secured and delivered inside a Secure Envelope. The envelope is sent to the consumer as an HTML attachment to a multipart MIME email message. The main body part of the email contains an unencrypted message, which consists of either plain text or text and HTML. This message is typically used to explain the purpose and content of the entire package. The IronPort Encryption appliance includes the ability to personalize the content of a message.

The IronPort Encryption appliance can be used to deliver secure documents or secure email messages, in the latter case, the text of the sender's email message should be encrypted and included in the payload of the Secure Envelope. In this case the envelope payload now contains a text message and optionally a number of attachment files, all encrypted. It is possible to use a template to control the look and content of the text message included in the envelope payload.

From here on, the term 'email message text' will be used to refer to the unencrypted plain text message that is in the MIME body part of the email message. 'Envelope message text' will be used to refer to the optional, encrypted text message that is in the IronPort PXE message payload for secure email applications.

Message Personalization Configuration Parameters

Message Personalization is configured per application. For example, to access the message personalization configuration parameters for the Envelope application, you would click the Configuration tab and click Configuration > SMTP Adaptor > Applications > Registered Envelope - Enrolled > Message Personalization. Please refer to "Appendix A: Configuration Parameters" in the *IronPort Encryption Appliance Configuration Manual* for a list of parameters.

Configuring the Variable Map for Text Personalization

The IronPort Encryption appliance mail component allows a certain amount of flexibility in setting up variable names used for message personalization. Information is stored in the `variablemap.properties` file, located in `<`*`install_dir`*`>/conf`. Each application can define its own mapping file. Field names can be appended with certain prefixes which will result in pre-processing of the value of the field before being personalized.

The following prefixes are allowed:

- Secure - This will encrypt the value of the field. This is not applicable to all fields and only the fields added to the default variablemap.properties with this prefix are supported.
- HTMLEscape - This will HTML escape the value of the field. This is applicable to all "Message", "Secure" and "Response" fields.
- URLEscape - This will URL Encode the value of the field. This is applicable to all "Message", "Secure" and "Response" fields.

The following table lists the personalization mappings in the variablemap.properties file. If some of these values are not being used for message personalization, then they can be removed from this map file.

Field Name/Definition	Description
email=Recipient	Recipient's email address.
firstname=Message.Header.X-PostX-firstname	Recipient's first name. This field is used to demonstrate the use of X-Headers in the variable map properties file.
lastname=Message.Header.X-PostX-lastname	Recipients last name. This field is used to demonstrate passing an X-Headers into the variable map properties file.
subject=Message.Subject	Subject of the email.
toList=Message.To	List of all 'To' recipient's email addresses, separated by commas.
cc=Message.Cc	List of all 'Cc' recipient's email addresses, separated by commas.
from=Message.From	Email address of the sender.
fromPersonal=Message.From.Personal	Personal name from the From address.
date=Message.Header.Date	Date of the incoming email message.
messageSensitivity=Message.messageSensitivity	The sensitivity of the message. This determines the actions required to open the envelope (password requirements, etc.)
textmessage=Envelope.Message.Text	Message text
htmlmessage=Envelope.Message.Html	HTML message
toc-attachments=Envelope.Attachments	For Secure Direct applications, a list of the names of any attachments to the email message that are encrypted inside the envelope payload. This list is a Java Vector and should be used as such by templates.
replyEnabled=Response.ReplyEnabled	Indicates if Secure Reply is enabled.
replyAllEnabled=Response.ReplyAllEnabled	Indicates if Secure Reply All is enabled.
forwardEnabled=Response.ForwardEnabled	Indicates if Secure Forward is enabled.
replyWebService=Response.ReplyWebService	Indicates if reply for web services is enabled.

Field Name/Definition	Description
replyURL=Response.URL	The Server Reply URL used for connecting to the server.
helpHostURL=Help.helpHostURL	URL for the help file.
secureFrom=Secure.From	Encrypted versions of the message properties.
secureReplyTo=Secure.Reply-To	Encrypted versions of the message properties.
secureToList=Secure.To	Encrypted versions of the message properties.
secureCcList=Secure.Cc	Encrypted versions of the message properties.
secureSubject=Secure.Subject	Encrypted versions of the message properties.
secureEmail=Secure.Recipient	Encrypted versions of the message properties.
encryptalgorithm=Secure.Algorithm	Encryption algorithm used for Secure Reply variables.
encryptiontoken=Secure.EncryptToken	Encryption token used for Secure Reply variables.
encryptionEnabled=Secure.EncryptionEnabled	Encryption enabled or not for Secure Reply.
secureKeyType=Secure.KeyType	Determines if Secure Reply characters are encrypted with a token or an encryption key.
accountId=Secure.AccountId	A token's account ID.
messageBarForgetMeURL=Secure.MessageBarFo rgetMeURL	URL that instructs the key server to remove the cookie.

TEMPLATE ENGINES

Overview

The IronPort Encryption appliance offers several template engine options for personalizing information in configured templates (such as messagebar.html). The options differ in the approach they use to personalize the data. More information on these different options is given below. The Velocity template engine is the most sophisticated of the lot and is the preferred option for most cases. The default options selected in the configuration need not be changed unless there is a specific need to prefer one option over the other.

Simple Template Engine

This template engine is suitable for simple text substitution but it cannot handle list values. The template document is a plain text file that contains the document to be personalized. Variable data fields are declared by enclosing them in `${…}`. Here is a sample template:

```
Dear ${FIRST_NAME},
Hello,
Thank you for registering. Please use your user ID and password to
login.

Thanks

${URL}
```

In this example, the field ${FIRST_NAME} is a substitution variable that will have the corresponding data value placed into it at execution time. If this field contains the value ‘Steve’ then the resultant document will look like this:

```
Dear Steve,
Hello,
Thank you for registering. Please use your user ID and password to
login.

Thanks

${URL}
```

Since the $ character is used to mark the beginning of a variable name, any occurrences of $ in the source document that are not variables must be escaped by entering them $$. So $10,000 would be written as `$$10,000.`

Velocity Template Engine

The Apache Jakarta Velocity template engine uses variables of the form ${…}, and it can also handle lists. Here is an example extract from an HTML template that uses a simple variable and a list variable:

```
<table border="1">
    <tr>
    <td>
    <table width="300" border="0" cellpadding="2" cellspacing="0">
        <tr>
        <td bgcolor="#CCCCCC">
        <pre>
${textmessage}
        </pre>
        </td>
        </tr>
        <tr>
        <td bgcolor="#003366"><p><span class="white">Attachments</
span></p></td>
        </tr>
#foreach ($attachment in ${attachments})
        <tr><td>&nbsp;</td></tr>
        <tr><td><a href="$attachment">$attachment</a></td></tr>
        <tr><td>&nbsp;</td></tr>
#end
    </table>
    </td>
    </tr>
</table>
```

Note that in this case the variable $attachment is local to the template and does not have its value defined by the caller.

More information about the Velocity template engine can be found in the documentation section of the Velocity web site:

`http://velocity.apache.org/`

XMLC Template Engine

The XMLC engine uses a different approach from the Simple and Velocity engines described above. XMLC is a compiler that takes a DOM document as input, either HTML or XML, and creates a Java class file that can be used to instantiate in memory a DOM representation of that document. For more information about DOM see `http://www.w3.org/DOM`. Variable data fields are identified using the id attribute of their DOM node, in the case of simple text in an HTML file, the text needs to be bracketed by a <span> tag like this <span id="*id*">some text</span>.

More information about the Enhydra XMLC DOM can be found at `http://www.xmlc.org`.

CHAPTER 2

MIME Envelope Builder

This chapter provides information about using the MIME Envelope Builder for packaging documents.

This chapter contains the following sections:

- "About the MIME Envelope Builder" on page 10
- "Compression Headers" on page 11
- "Keys and Encryption Headers" on page 12
- "X-PostX-Algorithm Header" on page 14
- "X-IronPort Headers for Incoming Mail" on page 16
- "X-IronPort Headers for Outgoing Mail" on page 16
- "Secure Envelope Headers" on page 17
- "Payload Headers" on page 20
- "Attachment Headers" on page 21

ABOUT THE MIME ENVELOPE BUILDER

The MIME Envelope Builder is controlled by a number of application-specific message headers ("X headers") as described in the following sections. Unless otherwise noted, if a header appears multiple times, the last occurrence is used.

COMPRESSION HEADERS

Documents included in the payload are grouped into secure documents and non-secure documents. Non-secure documents are defined as documents that do not contain customer-specific information and therefore do not need to be encrypted. The following two headers control whether or not each group is compressed:

X-PostX-Crypt-Compression

This header helps determine whether or not the secure documents are compressed. Valid values are:

Value	Definition
"1" or "compress"	Compress the secure documents
"0" or "none"	Do not compress the secure documents
"-1" or "conditional"	Compress the secure documents only if compression will actually make them smaller

X-PostX-Plain-Compression

This header helps determine whether or not the non-secure documents are compressed. Valid values are:

Value	Definition
"1" or "compress"	Compress the non-secure documents
"0" or "none"	Do not compress the non-secure documents
"-1" or "conditional"	Compress the non-secure documents only if compression will actually make them smaller

KEYS AND ENCRYPTION HEADERS

When encrypting the secure documents, the SHA-1 message digest of the user's key (for example, a password or PIN) is used to encrypt a random session key, the SHA-1 of which is used to encrypt the documents. The following headers are used to specify the user's key and the size of the key used to perform the actual encryption.

X-PostX-KeySize

This header contains the size of the key, in bytes, to use. A value of 0 indicates to use the maximum size (currently 20). A positive integer specifies the size of the key in bytes. If this header isn't present, a default key size of 0 is used.

X-PostX-Key

This header contains the user's key. The SHA-1 of the user's key is used to encrypt a random string, which is used to encrypt the secure documents. The same key must be entered by the user (or constructed from information entered by the user) to successfully decrypt the random string and hence the entire payload.

X-PostX-SHAedKey

This header contains the base64 encoded SHA-1 message digest of the user's key. The user's key is used to encrypt a random string, which is used to encrypt the secure documents. The key used to generate the message digest must be entered by the user (or constructed from information entered by the user) to successfully decrypt the random string and hence the payload.

If both X-PostX-SHAedKey and X-PostX-Key are present, X-PostX-SHAedKey is used.

The following example shows the generation of an SHA-1 key in python:

```
    Python
>>> import base64,sha
>>> clear_passwd='postx'
>>> base64_sha_passwd=base64.encodestring(
    ...sha.new(clear_passwd).digest())
>>> print base64_sha_passwd
```

Output:

```
qx5yG4zSyp+x0z3wDVS9D/VK9mQ=
```

Multiple Passwords

If X-PostX-Key or X-PostX-SHAedKey appears multiple times, *each* of the passwords specified can be used to decrypt the resulting document. For example, if the following headers appear in the input message, then both the key `postx` and the key `seamus` can be used to decrypt the envelope:

```
X-PostX-Key: postx
X-PostX-Key: seamus
```

X-PostX-PRNG

This header is used to specify the pseudo-random number generator used to generate the salt used when encrypting the secure documents and the random key. Valid values can take three forms:

Value	Definition
algorithm	Just an algorithm is specified, the default size is used
algorithm,size	The header determines both the algorithm and the size
size	Only the size is specified

If the algorithm is specified, it must be one of the algorithms provided by the Java 2 Standard Edition (currently only "SHA1PRNG"). If the algorithm isn't specified, java.util.Random is used. If the size is specified, it must be 0 (indicating the default size should be used) or a positive integer.

Note — This is not the size of the key used for encryption; therefore; there are no export considerations.

X-PostX-Max-Bad-Passwords

This header specifies the maximum number of bad passwords that the envelope allows.

Note — This header is ignored for Registered Envelopes.

X-POSTX-ALGORITHM HEADER

This header specifies the name of a cryptographic algorithm. The header value has the following format:

```
<algorithm type>=< algorithm name>[; < algorithm
   type>=< algorithm name>]...
```

The valid algorithm types and algorithm names are shown in the table below. The algorithm name in bold is the default.

Algorithm Type	Description	Algorithm Names
SaltGen	random salt generation	**Random,** SHA1PRNG
IVGen	initialization vector generation	**Random,** SHA1PRNG
SessionKeyGen	session key generation	**Random,** SHA1PRNG
SessionKeyEnc	session key encryption	**ARC4,** AES, RSA[1]
PayloadEnc	payload encryption	**ARC4,** AES
SessionKeyVer	session key verification	**CRC-32,** SHA-1
PayloadVer	payload verification	**CRC-32,** SHA-1
KeyServerKeyHash	key hashing between an envelope and the key server	plain, **SHA-1**
Note: RSA session key encryption is for special applications and requires configuration beyond just specifying the algorithm.		

The types are case-insensitive, but the names are case-sensitive. Whitespace surrounding the types and names is ignored. Each X-PostX-Algorithm header is used.

X-PostX-Identity

Used for Registered Envelopes when using external authentication providers such as Kerberos. The value for this would be the username to be authenticated against. For example, ‘jsmith’.

X-PostX-AccountId

Specifies the account to use for the message.

X-PostX-Message-Sensitivity

Specifies the sensitivity of the message. The header's value should be between 0 and 50, inclusive.

X-PostX-SkipRules

If present, no rule checking is performed for the message.

X-PostX-Token

This header contains the alias of the token to use when sending the message.

X-IRONPORT HEADERS FOR INCOMING MAIL

The IronPort Encryption appliance scans for the following headers in incoming mail. It uses these headers to direct mail to the appropriate application for encryption.

X-IronPort-Encrypt=SMIME

This header can be used to direct the message to the S/MIME application.

X-IronPort-Encrypt=PGP

This header can be used to direct the message to the PGP application.

X-IronPort-Encrypt=SecureMailbox

This header can be used to direct the message to the Secure Mailbox (formerly WebSafe) application.

X-IronPort-Encrypt=RegEnvelope

This header can be used to direct the message to the Registered Envelope application.

X-IRONPORT HEADERS FOR OUTGOING MAIL

The IronPort Encryption appliance inserts the following headers in outgoing mail.

X-IronPort-Encrypt=Success

The appliance inserts this header if the message was encrypted without errors.

X-IronPort-Encrypt=Failure

The appliance inserts this header if an error occurred during encryption.

WARNING: The `X-IronPort-Encrypt=Failure` header is particularly important if you configure the appliance to use an IronPort Email security appliance as the mail gateway. If the IronPort Encryption appliance adds the `X-IronPort-Encrypt=Failure` header to a message, then the IronPort Email Security appliance *must* be configured to recognize the header and handle the error. Otherwise, highly sensitive messages intended for encryption might inadvertently be sent unencrypted.

SECURE ENVELOPE HEADERS

The following headers contain information about the envelope itself.

X-PostX-EnvelopeFileEncoding

This header contains the name of the character encoding to use for the envelope attachment to the outgoing e-mail message. The value of this header, if present, is placed in the charset attribute of the Content-Type header of the MIME part containing the envelope.

X-PostX-Line-Endings

This header determines the line endings used for the envelope. Valid values are:

Value	Definition
"d" or "DOS"	carriage return, linefeed ("\r\n")
"m", "Mac", or "Macintosh"	carriage return ("\r")
"u" or "Unix"	linefeed ("\n")

If this header is not present, the default is DOS line endings. Except for special purpose applications, the default should be used.

X-PostX-Prefix

Each envelope is constructed from four or five parts, which are contained in files. The files are:

File Name	Description
prolog.html	This file contains what will be the first part of the final envelope. It should start the HTML document, open the <HEAD> section (but not close it), contain any style sheets, and open (but not close) a <SCRIPT> tag.
lib/*.js	This directory contains the pieces of the IronPort Encryption appliance JavaScript library which is used to decode and decrypt the payload and to control the applet (if the applet is required). Each file must be pure JavaScript.
js or cust.js	This file contains JavaScript specific to the application the envelope embodies. It must also be pure JavaScript (and JavaScript data).
text.js	This optional file contains a JavaScript object containing user-visible text.

File Name	Description
epilog.html	This file contains the final part of the envelope. It should close the <SCRIPT> tag and the <HEAD> section opened in prolog.html, contain the entire <BODY> section, and finally close the <HTML> document.

When the final envelope is constructed, it consists of the contents of prolog.html, any necessary files from lib, and the contents of js (or cust.js), text.js (if present) and epilog.html.

The value of the X-PostX-Prefix header is the path prefix to the files. If the value of the header ends with a slash, it should be a path to a directory containing the parts of the envelope. If the value of this header does not end in a slash, a period and the name of each part is appended to the prefix.

Example

If the value of the X-PostX-Prefix header is "../../samples/default_envelope/" then the files are:

```
../../samples/default_envelope/prolog.html
../../samples/default_envelope/lib/*.js
../../samples/default_envelope/js
../../samples/default_envelope/epilog.html
```

If the value of X-PostX-Prefix header is "../../samples/default_envelope" (note no trailing slash) then the files are:

```
../../samples/default_envelope.prolog.html
../../samples/default_envelope.lib/*.js
../../samples/default_envelope.js
../../samples/default_envelope.epilog.html
```

If this header isn't present, a locally configured default is used.

X-PostX-Envelope-Profile-Name

This header contains the name of the envelope profile to use when generating the Secure Envelope. Each Secure Envelope appearing in the Configuration tree (Configuration > SMTP Adaptor > Envelopes) generates an envelope profile. This header determines which of those envelopes will be used.

X-PostX-ExpirationDate

This header contains the date upon which a message expires. This date uses the following formats:

Value	Definition
+<days>	Relative number of days
+<hh:mm:ss>	Relative number of hours:minutes:seconds
<date>,<format>	Absolute date specified with a certain format. The format symbols must match those supported by java.util.SimpleDateFormat
<Milliseconds>	Absolute date specified as milliseconds since Jan 1, 1970

X-PostX-ReadNotificationDate

This header contains the date upon which a notification is sent if an email remains unread. This date uses the following formats:

Value	Definition
+<days>	Relative number of days
+<hh:mm:ss>	Relative number of hours:minutes:seconds
<date>,<format>	Absolute date specified with a certain format. The format symbols must match those supported by java.util.SimpleDateFormat
<Milliseconds>	Absolute date specified as milliseconds since Jan 1, 1970

Disposition-Notification-To

If present when building a registered envelope, a return receipt will be sent when the envelope is open. For now, the value of the header is ignored.

PAYLOAD HEADERS

The final headers specify the actual documents to be included in the envelope. The first headers appear in the input message itself, the remaining headers appear in each of the attachments to the input message.

X-PostX-Name

This header contains the name of the payload. This is currently only used to suggest a directory name when the user saves the payload and the payload contains multiple documents. If this header isn't present, the name of the "main" document—the document with 2 or 8 in the flags—is used (see "X-PostX-Flags" on page 21).

X-PostX-Add-Message-Bar

A flag indicating whether to generate a message bar and add it to the payload.

X-PostX-No-Message-Bar

A flag indicating no message bar should be added to the payload, regardless of the encryption server configuration.

X-PostX-AttachmentName

This header contains the file name of the attachment to include in the email.

X-PostX-AttachmentEncoding

This header contains the file encoding of the attachment to include in the email.

X-PostX-BodyFile

This header contains the email message file.

X-PostX-BodyFileEncoding

This header contains the encoding of the email message file.

X-PostX-BodyFileFields

This header contains the parameter values for the message file variables.

X-PostX-MessageTextCharset

This header pertains to message personalization.

X-PostX-MessageFileEncoding

This header pertains to message personalization.

X-PostX-MessageEncoding

This header pertains to message personalization.

ATTACHMENT HEADERS

The remaining headers appear on the attachment to the input message. Each attachment contains one of the documents to be placed in the payload. The body of the attachment contains the document's data. The headers examined are:

X-PostX-Flags

This header contains the flags, represented as an integer, for the document. The flags should be a sum of the following values:

Flags	Description
1	This document is secure (that is, it will be encrypted). If this flag isn't set, the document isn't encrypted.
2	The document should be written to the window containing the envelope.
4	The document should be written to disk.
8	The document should be launched (the browser told to open in a new window a file containing a URL pointing to this document). If flag 8 is used then flag 4 must also be used.
16	This value is reserved for future use and should not be included.
32	This flag is deprecated and is ignored whether or not it is set.

If the X-PostX-Flags header isn't specified, a default of 3 (secure, and write to envelope window) is used for text and HTML files, and a default of 13 (secure, write to disk, launch in new window) is used for binary files.

X-PostX-HTML-Description

This header contains an HTML description of the document. The HTML description is currently not used.

X-PostX-Text-Description

This header contains a text description of the document. The text description is currently not used.

Content-Type

This header is not specific to the IronPort Encryption appliance; rather, it is the standard MIME header. It is used to determine the type of document. Valid values are:

File Name	Description
text/plain	The document is plain text.

File Name	Description
text/html	The document is HTML.
multipart/*	The attachments contained within this attachment are recursively parsed.
message/rfc822	The attachment contained within this attachment is recursively parsed.

All other types result in a binary document.

CHAPTER 3

Password Schemas

This chapter contains the following sections:

OVERVIEW

This chapter provides a number of example messages that show the various password forms. These examples use mime_builder.py script to send the messages.

mime_builder.py is a command line Python script that can be used to construct and send test e-mail messages to an encryption server. It supports a number of options for formatting the header, body, and attachments to the e-mail and controlling how the e-mail is sent. In addition to this chapter, mime_builder.py contains built in documentation. You can display a brief summary of each of mime_builder.py's options using the -? option:

```
mime_builder.py -?
```

More detailed usage information can be displayed using the --usage option:

```
mime_builder.py --usage
```

A number of examples can be displayed using the --examples option:

```
mime_builder.py --examples
```

Each of the examples below contains a command line and output. The first line of each example command is shown starting with the '$ ' command prompt, and each subsequent line with the continuation prompt '> ', to differentiate them from output. The prompt aren't entered, and may be different than those shown here, depending upon your operating system and configuration.

Each command is also broken over multiple lines with new lines escaped by a backslash. The commands may be entered this way on Unix-like systems (Linux, Solaris, Cygwin, etc.). On Windows, the command must be entered on a single line with the backslashes removed.

Each of the examples includes the –w option, which instructs mime_builder.py to write the e-mail message to standard output in addition to sending it to the specified SMTP server. The –w option may be omitted to just send the message, or replaced with –W to just display the message.

The boundary attribute in each Content-Type: header in the sample output is generated randomly and will be different each time mime_builder.py is run.

CLEAR PASSWORDS

To send passwords in the clear to the encryption server, add the following X-Header:

```
X-PostX-key: <key>
```

mime_builder.py does this automatically with the key argument.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -w \
> postx mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
From: root@localhost
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.3856.1110908744.108.1"

--10.10.1.225.12206.3856.1110908744.108.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
    name="mime_builder.py"

[base64 encoded lines omitted]

--10.10.1.225.12206.3856.1110908744.108.1--
```

SHA-1 HASHED PASSWORDS

To send SHA encrypted passwords to the encryption server, add the following X-Header:

```
X-PostX-SHAedKey: <base64 encoded hashed key>
```

mime_builder.py does this automatically when you include the –S option.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" –S -w \
> postx mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
From: root@localhost
To: root@localhost
X-PostX-Secure: true
X-PostX-SHAedKey: qx5yG4zSyp+x0z3wDVS9D/VK9mQ=
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.2820.1110918369.219.1"

-- 10.10.1.225.12206.2820.1110918369.219.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
    name="mime_builder.py"

[base64 encoded lines omitted]

-- 10.10.1.225.12206.2820.1110918369.219.1--
```

USER ID AND PASSWORD

Creating an envelope that requests a user id and password does not require any additional X-PostX headers, but it does require a different envelope. This envelope is covered in the *MIME Envelope Builder* chapter.

To build the user ID and password, you must concatenate the user ID and password together and use the separator specified in the envelope. The example below uses the following values:

```
user id = engine
password = postx
separator = ^
```

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -w \
> engine^postx mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
From: root@localhost
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: engine^postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    oundary=" 10.10.1.225.12206.2820.1110918369.219.1"

-- 10.10.1.225.12206.2820.1110918369.219.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
     name="mime_builder.py"


[base64 encoded lines omitted]

-- 10.10.1.225.12206.2820.1110918369.219.1--
```

MULTIPLE PASSWORDS PER ENVELOPE

Allowing an envelope to accept multiple passwords requires each password to appear in an X-PostX-Key or X-PostX-SHAedKey header. Passwords sent in the clear should be placed in X-PostX-Key headers and SHA-1 hashed passwords should be placed in X-PostX-SHAedKey headers.

Multiple Clear Passwords

The following example adds three clear passwords. Any one of the three passwords may be used to open the resulting envelope. Note that mime_builder.py splits the key specified on the command line on commas and adds each resulting key.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -w \
> engine,postx,envelope mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
From: root@localhost
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: engine
X-PostX-Key: postx
X-PostX-Key: envelope
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary=" 10.10.1.225.12206.1152.1110918421.307.1"

-- 10.10.1.225.12206.1152.1110918421.307.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
 name="mime_builder.py"


[base64 encoded lines omitted]

-- 10.10.1.225.12206.1152.1110918421.307.1--
```

Multiple SHA-1 Hashed Passwords

The following example adds two SHA-1 hashed passwords. Either of the passwords may be used to open the resulting envelope.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -S -w \
> engine,postx mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
From: root@localhost
To: root@localhost
X-PostX-Secure: true
X-PostX-SHAed-Key: xkAzZy9cqYTf6WKQlnGWHoAS8MU=
X-PostX-SHAed-Key: qx5yG4zSyp+x0z3wDVS9D/VK9mQ=
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary=" 10.10.1.225.12206.1152.1110918421.307.1"


-- 10.10.1.225.12206.1152.1110918421.307.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
    name="mime_builder.py"

[base64 encoded lines omitted]

-- 10.10.1.225.12206.1152.1110918421.307.1--
```

CHAPTER 4

Payload Types

This chapter contains examples showing how to send email messages containing different types of payloads using the mime_builder.py command line tool. For a browser-based tool to construct and send email, please see the *IronPort Encryption Appliance Operations Manual*.

This chapter contains the following sections:

OVERVIEW

This chapter provides a number of example messages that show the various payload types. These examples use mime_builder.py script to send the messages.

mime_builder.py is a command line Python script that can be used to construct and send test email messages to an encryption server. It supports a number of options for formatting the header, body, and attachments to the email and controlling how the email is sent. In addition to this chapter, mime_builder.py contains built in documentation. You can display a brief summary of each of mime_builder.py's options using the -? option:

```
mime_builder.py -?
```

More detailed usage information can be displayed using the --usage option:

```
mime_builder.py --usage
```

A number of examples can be displayed using the --examples option:

```
mime_builder.py --examples
```

Each of the examples below contains a command line and output. The first line of each example command is shown starting with the '$ ' command prompt, and each subsequent line with the continuation prompt '> ', to differentiate them from output. The prompt aren't entered, and may be different than those shown here, depending upon your operating system and configuration.

Each command is also broken over multiple lines with new lines escaped by a backslash. The commands may be entered this way on Unix-like systems (Linux, Solaris, Cygwin, etc.). On Windows, the command must be entered on a single line with the backslashes removed.

Each of the examples includes the –w option, which instructs mime_builder.py to write the email message to standard output in addition to sending it to the specified SMTP server. The –w option may be omitted to just send the message, or replaced with –W to just display the message.

The boundary attribute in each Content-Type: header in the sample output is generated randomly and will be different each time mime_builder.py is run.

SENDING DIFFERENT TYPES OF PAYLOADS

This section contains examples showing how to send emails containing different types of payloads using the mime_builder.py command line tool.

Text Payload

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -tt -w \
> postx mime_builder.py
```

Sample Output

```
Subject: Test Mail from Enterprise
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.3856.1110908744.108.1"


--10.10.1.225.12206.3856.1110908744.108.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;
    name="mime_builder.py"

[base64 encoded lines omitted]

--10.10.1.225.12206.3856.1110908744.108.1--
```

If a payload is not specified, you will be prompted for input. Enter <ctrl-d> to end input on Unix-like systems, or <ctrl-z><return> on Windows.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -tt -w postx
Reading payload from stdin
hello world
<ctrl-d>
```

Sample Output

```
Subject: Test Mail from Enterprise
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.476.1110909521.329.1"


--10.10.1.225.12206.476.1110909521.329.1
Content-Transfer-Encoding: base64
Content-Type: text/plain;

aGVsbG8gd29ybGQK

--10.10.1.225.12206.476.1110909521.329.1—-
```

HTML Payload

Example

```
$ mime_builder.py -froot@localhost \
  > -m"localhost,root@localhost" -th -w postx
  Reading payload from stdin
  <html><body><b>hello world</b></body></html>
  <ctrl-d>
```

Sample Output

```
Subject: Test Mail from Enterprise
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.3808.1110910895.287.1"


--10.10.1.225.12206.3808.1110910895.287.1
Content-Transfer-Encoding: base64
Content-Type: text/html;

PGh0bWw+PGJvZHk+PGI+aGVsbG8gd29ybGQ8L2I+PC9ib2R5PjwvaHRtbD4K

--10.10.1.225.12206.3808.1110910895.287.1—-
```

Binary Payload

Binary payloads require that the recipient either have Java available and download a one-time Java applet. Alternatively, you can make "open online" available.

Example

```
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" -tb -w postx logo.gif
```

Sample Output

```
Subject: Test Mail from Enterprise
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.3500.1110911268.236.1"


--10.10.1.225.12206.3500.1110911268.236.1
Content-Transfer-Encoding: base64
Content-Type: application/octet-stream;
  name="logo.gif"

[base64 encoded lines omitted]

--10.10.1.225.12206.3500.1110911268.236.1—-
```

Multiple Attachment Payload

The individual parts of a multiple attachment payload must have an X-PostX-Flags header that determines the disposition of the part. mime_builder.py uses a directory containing a __toc__ file to generate the multiple attachments and the flags for each part.

Multiple attachment payloads also require that the recipient either have Java available and download a one-time Java applet. Alternatively, you can make "open online" available.

Example

```
$ mkdir test
$ cd test
$ touch test.gif
$ touch test.html
$ cat >__toc__
[DEFAULT]
payloadname = Multi Attachment
flags = FLAG_WRITE
[test.html]
flags = FLAG_WRITE | FLAG_SECURE | FLAG_LAUNCH
<ctrl-d>
$ cd ..
$ mime_builder.py -froot@localhost \
> -m"localhost,root@localhost" –w postx test/
```

Sample Output

```
Subject: Test Mail from Enterprise
To: root@localhost
X-PostX-Secure: true
X-PostX-Key: postx
X-PostX-Obfuscate: 0
X-PostX-Line-Endings: d
X-PostX-Name: Multi Attachment
Content-Type: multipart/mixed;
    boundary="10.10.1.225.12206.2724.1110911955.303.1"


--10.10.1.225.12206.2724.1110911955.303.1
Content-Transfer-Encoding: base64
X-PostX-Flags: 4
Content-Type: application/octet-stream;
  name="test.gif"


--10.10.1.225.12206.2724.1110911955.303.1
Content-Transfer-Encoding: base64
X-PostX-Flags: 13
Content-Type: text/html;
  name="test.html"


--10.10.1.225.12206.2724.1110911955.303.1--
```

CHAPTER 5

Secure Response

The chapter describes how the Secure Response application works and explains the *server* portion of the mechanism for Secure Reply messages.

This chapter contains the following sections:

INTRODUCTION

Secure Response allows a recipient of a Secure Envelope to reply to the sender in a manner as secure as the original message without using any special software. The recipient uses a standard SSL- and JavaScript-enabled web browser to send the Secure Reply message.

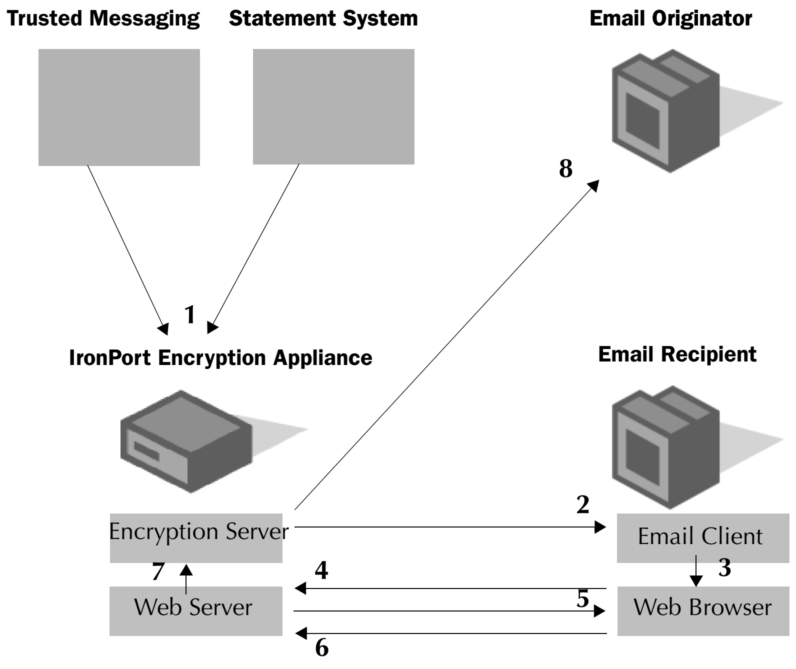
A specially constructed link included in the payload placed in a Secure Envelope provides the ability to send a Secure Reply. The link calls a JavaScript function that constructs an HTML form that is posted to an SSL-secured web page that allows the recipient to enter the reply. The reply email is then sent to the encryption server to secure the email before sending it to the original sender (or other recipient specified in the original payload). The envelope containing the reply may also allow a Secure Reply to the initial reply. Optionally, the Secure Reply message can be sent in the clear, without encryption. That way, you can enable reply email messages sent within the organization to be unencrypted.

The following Secure Response options are available:

- **Secure Reply -** Enables the Reply button on the envelope contents. The reply is sent to the sender only.
- **Secure Reply to All -** Enables the Reply All button on the envelope. The reply is sent to the sender and all of the recipients in the To and CC lists.
- **Secure Forward -** Enables the Secure Forward button on the envelope which allows the recipient to forward the message to other users. Note: Attachments are not currently forwarded.

Note — For information on configuring SSL, please see the *IronPort Encryption Appliance Configuration Manual*.

The following diagram gives an overview of the steps taken in receiving a Secure Envelope and responding to it with a Secure Reply message.



The steps in the process are:

1. The data required for generating a secure email is sent to the encryption server.
2. The encryption server constructs an email message with a Secure Envelope that contains the payload as an attachment. The encryption server is configured so that, when the payload is decrypted, it enables the recipient to send a Secure Reply. The email message is sent to the recipient.
3. The recipient opens the Secure Envelope in a web browser and enters a password to decrypt and display the payload.
4. The recipient clicks the link to send a Secure Reply. The link calls a JavaScript function that opens a new browser window and sends an initial POST to the Secure Reply web application on the web server.
5. Based on the configuration file, the web application generates an HTML form with some of the fields completed using values specified in the payload. The web application then sends the form back to the recipient's web browser.
6. The recipient enters the reply and submits the form, which creates an HTTP POST back to the Secure Response web application. The web application examines the submitted

information. It regenerates the form if there are errors with the submitted information, and it repeats steps 5 and 6 as necessary. When the submitted information is free of errors, the web application constructs an email message using the entered data and any attached files.

7. The message is sent to the encryption server for delivery. Depending on the configuration, the server encrypts the message in a Secure Envelope. Secure Response can be configured to send the reply message unencrypted.
8. The reply message is delivered to the sender of the original email using the address specified in the original message.

Note — When encrypted messages are sent using Secure Response, the originator might reply to a Secure Reply message. In that case, the originator may receive a Secure Envelope with a payload that allows Secure Reply messages. An email exchange with multiple messages and replies can occur with all the messages protected in Secure Envelopes.

SECURE RESPONSE BEHAVIOR

Secure Response behavior is determined by three elements: the Secure Reply configuration as set in the encryption server, the HTML form values passed by the JavaScript when it initially posts to the page, and the certificate associated with the SSL server.

SECURE RESPONSE FORM

The Secure Response form is invoked by clicking on the Reply, Reply All, or Forward buttons on the payload. A form is displayed that allows you to respond to or forward the message and manage multiple attachments.

HTML Form Values

Communication between the Secure Reply web application and the user is via an HTTP POST. The initial POST may establish a value for any of the fields, some of the field values are derived from the configuration file entries, and user entered values provide the remaining fields.

to

The To field contains the destination of the Secure Reply. The To field can only be edited if you are responding using the Forward button.

bcc

This field allows entry of space, comma, or semi-colon separated addresses to be placed in the Bcc: header of the reply. This field is only available if the Offer BCC parameter is checked within the encryption server. If Freeze Bcc is checked, this field is shown as non-editable.

cc

This field allows entry of space, comma, or semi-colon separated addresses to be placed in the Cc: header of the reply. This field is only available if the Offer CC parameter is checked within the encryption server. If Freeze CC is checked, this field is shown as non-editable.

from

The from field contains the email address of the recipient opening the Secure Envelope. This field is not editable.

subject

The subject field contains the subject of the email message containing the Secure Envelope containing the Secure Reply. The subject is always displayed but is only editable if the Freeze Subject parameter is not checked within the encryption server.

sendsecure

This field specifies whether the reply email should be sent as a secure email or as a plain email. Absence of this field will result in the decision of securing the email being made on the domain list matching.

SSL/TLS CERTIFICATE

In the standard configuration, the Secure Reply is served using the HTTPS protocol on an SSL-secured connection. Such a connection requires an X.509 certificate. The certificate must be stored in the JKS file called keystore in the conf directory. The certificate should be stored under the alias “pxessl”, the algorithm must be RSA, and the keystore and key passwords must be “pxessl”.

The default keystore file contains a sample certificate suitable for testing, but a real certificate should be placed in the keystore file before deployment.

CHAPTER 6

Controlling the WebSafe Application

This chapter contains the following sections:

OPTIONAL STANDARD PARAMETERS PER INCOMING EMAIL

WebSafe enables you to edit certain global parameters in SMTP headers on a per message basis.

SMTP headers have the following format:

<header>:<value>

The parameters are specified as SMTP headers to the incoming message and are as follows:

Parameter	**Definition**
X-*PostX*WS-AppName	This specifies the application name for the incoming email and is stored along with the email.
X-PostXWS-ExpiryDays	This specifies the content expiry duration in number of days for this email and overrides the global value "ExpiryDurat*ionInDays".*
X-PostXWS- RRNotifyDays	This specifies the return receipt notification expiry duration in number of days for this email and overrides the global value "NotifyNoRRInDays".
Disposition-Notification-To	This is a standard SMTP header recognized by WebSafe, which indicates the need for a return receipt for this email.

OPTIONAL CUSTOM PARAMETERS PER INCOMING EMAIL

WebSafe enables you to use custom fields and values per message, which are stored in the database. These parameters are specified as SMTP headers in the incoming message and are of the format <header>:<value>, as follows:

```
X-PostXWS-<FieldName>:<FieldValue>
```

For example: X-PostXWS-CaseId:00000123321

Time-Sensitive Emails: Monitoring Aging and Missing Return Receipts

The encryption server hosts a "WebSafeService" Mbean that monitors for aged content data and marks it for deletion from the database. The emails marked for deletion are not shown to the recipient. It also monitors for return receipts received and if the return receipt has not been received within the specified "NotifyNoRRInDays" it notifies the sender appropriately.

The default version of this service runs only once a day and not continuously.

You can configure the Mbean by going to the Configuration tab and clicking Scheduling > Scheduled Tasks > WebSafe Service.

Templates for Compose

WebSafe supports templates to be used when composing a new email. The default compose button uses a template XML file "plain.xml" stored at the "Compose Template Path" location specified in WebSafe Client Configuration. The XML template allows an enterprise to change its default compose screen behavior and/or add new compose screens easily. The templates can specify each newly composed email to:

- Have certain pre-filled values for fields
- Not show certain parameters like cc or attachments
- Make certain fields read-only
- Specify custom parameters
- List allowable or restricted attachments
- Specify a maximum limit for attachment size, and so forth.

These templates are defined in XML format and an example is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: plain.xml,v 1.1 2002/10/14 18:17:16 ashish Exp $ -->

<WebSafe-compose version="$Revision: 1.1 $">
    <application>Direct Deposit Account</application>
    <form>
      <offer-to>true</offer-to>
      <offer-cc readonly="true">true</offer-cc>
         <offer-bcc>false</offer-bcc>
      <offer-subject>true</offer-subject>
      <offer-attachment>true</offer-attachment>
      <custom name="Account" readonly="true"></custom>
      <custom name="CaseId"></custom>
      <custom name="BranchId"></custom>
    </form>
    <defaults>
      <to>
        <value id="1">ashish@postx.com</value>
                <value id="2">shankar@postx.com</value>
                <value id="3">kumar@postx.com</value>
      </to>
      <cc>
        <value id="1">ashish@postx.com</value>
      </cc>
        <subject>
          <value id="1">Account Balance</value>
          <value id="2">Resend Cheques</value>
        </subject>
        <CaseId>
          <value id="1">Case001</value>
          <value id="2">Case002</value>
          <value id="3">Case003</value>
        </CaseId>
        <Account>
          <value id="1">Acct001</value>
        </Account>
        <BranchId>
          <value id="1">Br001</value>
       </BranchId>
    <attachments>
      <allowed>*.htm,*.txt,*.pdf</allowed>
      <not-allowed></not-allowed>
      <max-size-allowed>5000000</max-size-allowed>
    </attachments>
  <domains>
    <restrict>enterprisedomain.com</restrict>
  </domains>
  <sender-confirm>true</sender-confirm>
 </defaults>
</WebSafe-compose>
```

WEBSAFE HEADER FOR SENDING NOTIFICATIONS

By default, the `X-PostX-NoSecure: true` header is used by the encryption server to identify email messages that are to be sent out to the users in the standard non-encrypted format. The HasHeader PostX-NoSecure matcher is located in the Check for Send Clear rule in the root ruleset, and it processes the message in the SMTP Delivery application for remote delivery in an unencrypted format. WebSafe typically uses this header to send out email notifications to external email addresses, informing them that a new message has arrived.

Index