TIBCO iWay® Service Manager

Pretty Good Privacy (PGP) Extension User's Guide

Version 8.0 and Higher
March 2021
DN3502302.0321

Contents

Chapter 1

Introducing Pretty Good Privacy (PGP)

This section provides an overview of the basic concepts involved in Pretty Good Privacy (PGP).

In this chapter:

PGP Overview

Pretty Good Privacy (PGP) is a software program that provides encryption, decryption, digital signature and key management services. PGP was developed by Philip R. Zimmermann starting in 1991. It was later formalized as a specification called OpenPGP as described in RFC4880. PGP is the original implementation of OpenPGP. The GNU Privacy Guard (GPG) is another implementation that avoids encumbered algorithms. In this document, the term PGP is often used instead of OpenPGP to simplify the discussion.

Cryptographic Services

In symmetric encryption, the same key is used to encrypt and decrypt. Symmetric algorithms are fast, but the distribution of the key can be cumbersome since it must be kept a secret. In asymmetric encryption, the public key and the private key form a key pair. The private key must be kept a secret but the public key can be distributed to anyone. Data encrypted by the public key can only be decrypted by the owner of the private key. Data encrypted by the private key can be decrypted by anyone since anyone can have the public key, but doing so authenticates its origin.

PGP encrypts using a symmetric algorithm with a new encryption key for each message. This session key is itself encrypted and sent together with the message.

With passphrase encryption, the session key is encrypted with a passphrase. The passphrase is simply a long password. Since the passphrase is known by the sender and the recipient, they can both decrypt the session key which is then used to decrypt the actual message. The difficulty with this approach is to distribute the passphrase to the recipient with some out of band mechanism and still keep it a secret.

With key pair encryption, the session key is encrypted with a public key. This eliminates the need for a shared secret because the public key can be known by anyone. The recipient uses the private key to decrypt the session key which is then used to decrypt the actual message. Since the private key is a secret known only by the recipient, only the recipient can decrypt the message.

PGP can sign messages using digital signatures. A signature confirms the identity of the sender and it confirms the message has not been altered during transmission. A digital signature is created as follows:

1. The message is reduced to a small block of data using a one-way hash function.
2. The hash is encrypted with the private key of the signer.
3. This signature is sent together with the original message.

The recipient can validate the signature with the following procedure:

1. The signature is decrypted using the public key of the signer to obtain the hash.
2. The hash is saved momentarily and recomputed from scratch over the original message.
3. The two hash values are compared.
4. If the values are equal, the signature is validated. Otherwise, the message was altered and the signature is invalid.

PGP almost always compresses the message to reduce its size. This has the advantage of reducing the redundancy within the message and therefore makes the encryption more effective.

PGP performs the operations in this order: the clear text message is signed first, the result is compressed and finally it is encrypted. It is possible to skip any of these steps.

The result of processing a clear message with PGP is a binary message. PGP offers another form called *armoring*, which produces an ASCII message similar to Base64. This is not cryptographically more secure, but may protect against inadvertent modifications of the message during transport through some media, such as email.

Key Management

The identity of a PGP user is given by a master key. This key usually has a very long expiration to keep the same identity for a long time. The master key is a key pair made up of the master public key and the master private key.

One or more User IDs can be bound to a master key. A User ID is made up of three parts in this format:

real name (*comment*) <*email address*>

For example:

John Smith (Doc Services) <john.smith@example.org>

It is possible to have multiple User IDs for situations where the user is known by different names in different circles. For example, another User ID can be created for the personal email address. This would help keep the personal communication separate from work.

A PGP user needs more than one key. For example, it is a good idea to use different keys for signature and encryption. It is also a good idea to change the encryption key regularly to reduce the risk of an attack.

PGP defines a subkey as a key bound to a master key. A subkey is an ordinary key that has been signed by the master key to indicate it belongs to the same identity. The advantage is a subkey can be revoked independently of the master key. For example, the master key is usually restricted to key creation and signing. A separate subkey is used just for encryption. This approach makes it possible to change the encryption key more frequently and still keep the user identity intact as the master key. It is also possible to have subkeys for signature but this is less common.

The master key together with all its subkeys is called a key ring. PGP often stores a key ring in a file which is also called a key ring. This is somewhat confusing, since a key ring file may contain multiple key rings, and is therefore more akin to a key ring collection. The public keys are stored in a public key ring file, and the private keys are stored in a secret key ring file. The public and the secret key rings are created when the first master key is created.

It is possible to retrieve a key from a key ring file based on the User ID or one of its subparts (usually the email address). Since the User IDs are bound to a master key, the search will first identify the first key ring that matches in the key ring file. It will then return the first suitable key among the key ring, which could be the master key or one of its subkeys. This search is somewhat ambiguous but is often adequate.

To retrieve a (sub)key more precisely, it is possible to use the Key ID which is a string of 16 hexadecimal digits computed from a hash of the key. PGP also defines the Short Key ID as the right-most 8 hexadecimal digits of the Key ID. Short Key IDs are more convenient for humans at the expense of a somewhat higher risk of collision. For extremely precise identification of the key, PGP defines the fingerprint as a string of 40 hexadecimal digits. The chance of collision with a fingerprint is extremely small.

You will need to distribute your public key to let partners verify your signature and encrypt messages destined to you. PGP makes it easy to export your public key from your public key ring. The resulting document is called a public key, but it is more akin to a certificate since it contains the master public key, the public keys of the subkeys, and the User IDs. The public key can be uploaded to a public key server, posted on a web site, or sent directly to the partner. The partner will import this public key within his public key ring. Conversely, you will import your partner public keys in your own public key ring. This will add another master key to the public key ring. This one is different than your master key because there is no associated private master key in the secret key ring.

PGP defines a web of trust to certify a key really belongs to a user. A user can claim complete or partial trust in the identity of another user. This is a generalization of the strictly hierarchical model used in X509. The PGP extension has complete trust in all public keys in the public key ring. The contents of the public key ring must be carefully managed to reflect this assumption.

Supported PGP Components

The following iWay Service Manager (iSM) components are supported by the iWay Pretty Good Privacy (PGP) Extension:

- Services
  - PGP Encrypt Service (com.ibi.agents.PGPEncrypt). This service can sign and/or encrypt outgoing documents.
  - PGP Decrypt Service (com.ibi.agents.PGPDecryptAgent). This service can decrypt incoming documents and/or verify signatures.
- Preparser
  - PGP Decrypt Preparser (com.ibi.preparsers.PGPDecrypt). This preparser performs the same functionality as the PGP Decrypt Service (com.ibi.agents.PGPDecryptAgent).
- Preemitter
  - PGP Encrypt Preemitter (com.ibi.preemit.PGPEncrypt). This preemitter performs the same functionality as the PGP Encrypt Service (com.ibi.agents.PGPEncrypt).

Note: The iWay PGP Extension also provides the following iSM components for backwards compatibility. However, these have been superseded by the iSM components listed above.

- Services
  - PGP Sign Service (com.ibi.agents.XDPGPSignAgent). This service is deprecated. Use the PGP Encrypt Service (com.ibi.agents.PGPEncrypt) instead.
  - PGP Verify Sign Service (com.ibi.agents.XDPGPVerifySignAgent). This service is deprecated. Use the PGP Decrypt Service (com.ibi.agents.PGPDecryptAgent) instead.
- Preparser
  - PGP Verify Sign Preparser (com.ibi.preparsers.XDPGPVerifySignPreparser). This preparser is deprecated. Use the PGP Decrypt Preparser (com.ibi.preparsers.PGPDecrypt) instead.
- Preemitter
  - PGP Sign Preemitter (com.ibi.preemit.XDPGPSignPreEmitter). This preparser is deprecated. Use the PGP Encrypt Preemitter (com.ibi.preemit.PGPEncrypt) instead.

Configuring iWay Service Manager Components

The iWay Pretty Good Privacy (PGP) Extension provides specific services, preparsers, and preemitters that must be configured using iWay Service Manager (iSM). These components can then be added as inlets, routes, and outlets to build channels. For more information on configuring channel components, see the *iWay Service Manager User's Guide*.

Chapter 2

Configuring the iWay Pretty Good Privacy (PGP) Extension

This section describes how to configure the iWay Pretty Good Privacy (PGP) Extension.

In this chapter:

Configuring Bouncy Castle

The PGP specification lists the symmetric algorithms that can be used for encryption. The iWay PGP extension relies on the installed cryptographic providers to supply the necessary Cipher implementations. The IDEA implementation is missing by default because it is patented in many parts of the world and requires a license to run. The extended Bouncy Castle provider can be installed manually to add the IDEA implementation in the parts of the world that are not affected by this license. The user is responsible to confirm he complies with all the IDEA licensing terms.

Procedure: How to Configure Bouncy Castle

1. Navigate to the following directory:

   <INSTALLDIR>\lib

2. Locate the file that matches bcprov-<*VERSION*>.jar and make a note of this version.
3. Download the bcprov-ext-<*VERSION*>.jar from the Bouncy Castle website.

   The versions of the bcprov-<*VERSION*>.jar and bcprov-ext-<*VERSION*>.jar files must match.

   The Bouncy Castle website usually provides only the latest version, so you may need to download the bcprov-ext-<*VERSION*>.jar file from another website.
4. Create a backup of the bcprov-<*VERSION*>.jar file in another directory.
5. Remove the bcprov-<*VERSION*>.jar file and replace it with the bcprov-ext-<*VERSION*>.jar file.
6. Verify whether the Bouncy Castle is also installed in the following directories:

   %JAVA_HOME%\lib\ext

and

```
%JAVA_HOME%\jre\lib\ext
```

If so, then replace bcprov-<*VERSION*>.jar with bcprov-ext-<*VERSION*>.jar in these locations also, since the lib\ext directory takes precedence.

Configuring Pretty Good Privacy (PGP)

The following sections describe how to create the key rings, list the keys, and import/export public keys using the GNU Privacy Guard (GnuPG) utility. If you are using another PGP implementation, then consult the accompanying documentation to learn how to accomplish the same tasks.

Procedure: How to Create Key Rings

The iWay Pretty Good Privacy (PGP) Extension requires access to the public key ring and the secret key ring. To create key rings using the GNU Privacy Guard (GnuPG) utility:

1. Download the GnuPG utility from the following website:

   http://www.gnupg.org/download/

   For Windows, either the full or light version of Gpg4win is acceptable, since only the command line interface is required.

2. Install the GnuPG utility.

3. Open a command prompt and navigate to the directory where the GnuPG utility is installed. For example:

   ```
   cd C:\Program Files (x86)\GNU\GnuPG
   ```

4. Generate a master key pair by executing the following command:

   ```
   gpg2 --gen-key
   ```

5. Select *RSA (sign only)* as the key type, and follow the prompts.

   You will be prompted to enter the key length, expiration, user real name, user email address, and an optional comment.

6. Enter the passphrase, which should be a long password.

   Make note of the passphrase, since your master key becomes unusable without it.

7. Edit the master key with the following command.

   ```
   gpg2 --edit-key user@host
   ```

where:

user@host

Is the actual user email address previously entered.

8. Start adding an encryption subkey using the following command:

```
addkey
```

9. Enter the passphrase to access the master private key.
10. Select *RSA (encrypt only)*, enter the key length and expiration.
11. Save the new subkey and exit the GNU Privacy Guard (GnuPG) utility using the following command:

```
save
```

Procedure: How to List Keys

- To list the public keys, enter the following command:

```
gpg2 --list-keys
```

The short key IDs and the user IDs are displayed.

- To list the private keys, enter the following command.

```
gpg2 --list-secret-keys
```

The short key IDs and the user IDs are displayed.

- To view the fingerprint of the public keys, enter the following command:

```
gpg2 --fingerprint --fingerprint
```

The second identical option is required to view the fingerprints of the subkeys.

- To view the fingerprint of the private master keys, enter the following command:

```
gpg2 --list-secret-keys --fingerprint
```

Remember, the fingerprint of a private (sub)key is the same as the fingerprint of the corresponding public (sub)key.

- To view the long key ID of the public keys, enter the following command:

```
gpg2 --list-keys --with-colon
```

The output of this command is intended to be machine readable. The long key ID is the field containing 16 hexadecimal digits.

- To view the long key ID of the private keys, enter the following command:

```
gpg2 --list-secret-keys --with-colon
```

Procedure: How to Export a Public Key

To export your public key to send to a partner, enter the following command:

```
gpg2 --armor --output pub.asc --export user@host
```

where:

user@host

Is the actual user email address in the User ID.

The output is in the pub.asc file. This file must be distributed to the communication partners.

Procedure: How to Import a Public Key

To import the public key of a partner, enter the following command:

```
gpg2 --import filepath
```

where:

filepath

Is the actual file path.

Configuring the Java Runtime Environment

This section describes how to configure the Java Runtime Environment (JRE) for use with the iWay PGP Extension.

Jurisdiction Policy Files

By default, the JCE Cryptography polices are limited, which do not allow certain encryption algorithms to function properly. Ensure that the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files are installed. They can be downloaded from *http://www.oracle.com* at the same location as the JRE.

Configuring iSM Components for the iWay Pretty Good Privacy (PGP) Extension

This section describes how to configure the following iSM Components for the iWay Pretty Good Privacy (PGP) Extension:

- PGP Encryption and Signature Service (com.ibi.agents.PGPEncrypt)

- PGP Decryption and Verification Service (com.ibi.agents.PGPDecryptAgent)
- PGP Decryption and Verification Preparser (com.ibi.preparsers.PGPDecrypt)
- PGP Encryption and Signature Preemitter (com.ibi.preemit.PGPEncrypt)

Configuring the PGP Encryption and Signature Service

Syntax:

```
com.ibi.agents.PGPEncrypt
```

Description:

This service is used to encrypt and/or sign a document using Pretty Good Privacy. PGP is specified in RFC4880. This section assumes you are familiar with PGP.

PGP encrypts using a symmetric algorithm using a new session key for each message. The session key is itself encrypted and bundled with the cipher text. The session key can be encrypted in one of two methods: passphrase or key pair. The passphrase method hashes a long password to produce the key encryption key. The key pair method uses the public key of the recipient as the key encryption key. When using the key pair method, the recipient public key must be stored in a public key ring to be retrieved.

PGP signs messages using the private key of the signer. The private key must be stored in a secret key ring to be retrieved.

PGP performs the operations in this order: the clear text message is signed first, the result is compressed and finally it is encrypted. It is possible to skip any of these steps. When none of the operations are selected, the result is still a binary PGP message made up of a literal data packet.

Parameters:

The following tables describe parameters for the PGP Encryption and Signature Service. Each table is followed by a discussion of the specific parameter group.

General Parameters	
Armor	Determines whether the binary message will be armored for transfer in ASCII.
Literal Filename	The file name to appear in the Literal Data Packet. The receiving program may use this name when storing the data to disk. The special value _CONSOLE means the data is unusually sensitive.

The result of processing a document with PGP is a binary message. PGP defines a format called *armoring* that encodes the binary information in ASCII similar to Base64. The message is not more secure since it encodes exactly the same binary information. This format is useful to protect the data during transport when the media could inadvertently alter it, for example in the body of an email message.

PGP wraps the original message in a Literal Data Packet. This packet contains an optional field that can hold a filename. This can be used by the recipient as a hint when storing the data to disk. The special value _CONSOLE means the data is *for your eyes only* and should not be permanently saved. This is just a convention and PGP does not enforce it.

Signature Parameters	
Sign	Whether to sign the data in the PGP message.
Signature Hash Algorithm	Algorithm to use for the signature digest.
Secret Key Ring	Location of the secret key ring containing the signature key.
Signature Key User ID	The User ID of the private key used for signing. The format can be *=name* for the exact User ID, *user@host* to match only the email address, or <*user@host*> optionally surrounded by ignored comments to match only the value within angle brackets against the email address. This parameter also determines which User ID(s) will be listed in Signer's User ID subpackets.
Signature Key ID	The Key ID of the private key used for signing. The format can be 8 hex digits for the short key ID, 16 hex digits for the long key ID, or 40 hex digits for the fingerprint. Spaces are allowed, but are not counted in the length. This parameter can be used in place or together with the Signature Key User ID to select a more specific key.
Signature Key Passphrase	Case-sensitive passphrase to unlock the Signature Private Key.

Signature Parameters	
Signature Expiration	The number of seconds after the signature creation time that the signature expires. The format is: `[xxh][xxm]xx[s]` If absent or has a value of zero, the signature never expires.

The Sign parameter determines whether the clear text document will be signed. When the document is unsigned, the remaining parameters of this group are ignored.

The Signature Hash Algorithm parameter specifies the algorithm used to compute the hash. The signature algorithm itself is determined by the type of the signature key. For example, an RSA signature key will create an RSA signature, similarly, a DSA signature key will produce a DSA signature.

The Signature Key User ID and the Signature Key ID specify which signature key will be retrieved from the Secret Key Ring. The User ID is possibly ambiguous but is convenient for humans. The Key ID should be used when the signer key must be determined more precisely. Both parameters are optional. The signer key must satisfy both criteria when both parameters are used. When both parameters are absent, the default signature key is chosen. The signer key must not be revoked and it must not be an encryption only key.

The Signature Key Passphrase is required to unlock the signature private key. This is because the private key is encrypted within the secret key ring.

The Signature Expiration is optional. It determines how long the signature will remain valid. The format is:

`[xxh][xxm]xx[s]`

For example, 1h30m means the signature will be valid for 90 minutes. If the value is absent or has a value of zero, the signature never expires.

Compression Parameters	
Compress	Whether to compress the data in the PGP message.
Compression Algorithm	Algorithm to use to compress the data in the PGP message.

The Compress parameter determines whether the data will be compressed. This will be the signed data when signature is on, or the clear text message when unsigned. PGP almost always compresses the message. It might be necessary to turn off compression to work around a problem with the software of a partner. The Compression Algorithm parameter specifies which algorithm will be used to compress the message. The default is ZIP.

Encryption Parameters	
Encrypt	Whether to encrypt the data in the PGP message.
Symmetric Algorithm	Symmetric algorithm to be used for encryption.
Encryption Method	Determines how the session key is encrypted, use Passphrase for Password-Based Encryption (PBE), use Keypair to encrypt with a public key.
Encryption Passphrase	Case sensitive passphrase when using the Passphrase encryption method. The passphrase will be used to generate a key encryption key to encrypt the session key.
Public Key Ring	Location of the public key ring containing the encryption public key when using the Keypair encryption method.
Encryption Key User ID	User ID of the recipient public key when using the Keypair encryption method. The format can be *=name* for the exact User ID, *user@host* to match only the email address, or *<user@host>* optionally surrounded by ignored comments to match only the value within angle brackets against the email address. For backwards compatibility, this parameter defaults to the value of the Encryption Passphrase parameter.
Encryption Key ID	The Key ID of the recipient public key when using the Keypair encryption method. The format can be 8 hex digits for the short key ID, 16 hex digits for the long key ID, or 40 hex digits for the fingerprint. Spaces are allowed, but are not counted in the length. This parameter can be used in place or together with the Encryption Key User ID to select a more specific key.

The Encrypt parameter determines whether the message is encrypted. When the message is not encrypted, the remaining parameters of this group are ignored.

The Symmetric Algorithm parameter specifies the algorithm to be used for encryption. This implicitly determines the size of the session key. The DES algorithm is not recommended since it is now considered too weak.

The Encryption Method determines how the session key is encrypted. Choose between Passphrase or Keypair.

With the Passphrase method, the Encryption Passphrase is used to generate a key encryption key to encrypt the session key. The Public Key Ring, Encryption Key User ID, and Encyption Key ID are ignored.

With the Keypair method, the recipient public key is used as the key encryption key. The Encryption Key User ID and the Encryption Key ID specify which public key will be retrieved from the Public Key Ring. The User ID is possibly ambiguous but is convenient for humans. The Key ID should be used when the recipient key must be determined more precisely. Both parameters are optional. The recipient key must satisfy both criteria when both parameters are used. When both parameters are absent, the default encryption key is chosen. The selected key must not be revoked and it must support encryption. The Encryption Passphrase and the Encryption Key User ID parameters used to be a single parameter. For backwards compatibility, the Encryption Key ID defaults to the value of the Encryption Passphrase parameter. Except for this, the Encryption Passphrase parameter is ignored with the Keypair method.

Edges

The following table lists and describes the edges that are returned by the PGP Encryption and Signature Service (com.ibi.agents.PGPEncrypt).

Edge Name	Description
success	The message was successfully packaged with PGP.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_security	The message could not be packaged.

Configuring the PGP Decryption and Verification Service

Syntax:

```
com.ibi.agents.PGPDecryptAgent
```

Description:

This service decrypts and verifies the signature of a message packaged with Pretty Good Privacy (PGP). When successful, this service returns the original clear text message. PGP is specified in RFC4880. This section assumes you are familiar with PGP.

The service accepts messages constructed in various ways. The first input message might be encrypted and unsigned. The second message might be unencrypted and signed, and so on. The structure of the message determines how it is processed. Special Registers (SREGs) describe the structure that was actually found. It is also possible to insist that input messages must be signed.

Parameters:

The following tables describe the parameters for the PGP Decryption and Verification Service. Each table is followed by a discussion of that parameter group.

Decryption Parameters	
Decryption Key Passphrase	Case sensitive passphrase to decrypt the session key. This parameter is optional, and will be used when the session key has been encrypted with a passphrase.
Secret Key Ring	Location of the secret key ring containing the private keys to decrypt the session keys. This parameter is optional, and will be used when the session key has been encrypted with a public key.
Private Key Passphrase	Case sensitive passphrase to unlock the Decryption Private Key within the Secret Key Ring. This parameter is optional, and will be used when the session key has been encrypted with a public key.

These parameters come into play when the input message is encrypted. The Decryption Key Passphrase is used to decrypt the session key if it was encrypted with a Passphrase. If the session key was encrypted with a public key, then the Secret Key Ring is searched for a private key matching the Key ID indicated in the message. The Private Key Passphrase is needed to unlock the private key. This is because the private key is encrypted within the secret key ring.

Signature Verification Parameters	
Signature Required	If set, incoming messages require a valid signature.

Signature Verification Parameters	
Public Key Ring	Location of the public key ring containing the signature public keys.
Acceptable Hash Algorithms	Space separated list of hash names that can be used in PGP signatures, other hashes will cause a validation failure before being evaluated. The intention is to forbid weak algorithms defined in the PGP specification. The default is SHA1 SHA224 SHA256 SHA384 SHA512. Other hash names include MD5, RIPEMD160, DOUBLE_SHA, MD2, TIGER_192, and HAVAL_5_160.

When the Signature Required parameter is set and the input message is unsigned, an error is reported and the fail_unsigned edge is followed.

The remaining parameters come into play when the input message is signed. The Public Key Ring is searched for the signer public key matching the Signer Key ID indicated in the message. This public key is used to read the signature in the message.

The Acceptable Hash Algorithms parameter can be used to reject signatures produced with weak hash algorithms. The value of the parameter is a space-separated list of acceptable hash algorithm names. The default is SHA1 SHA224 SHA256 SHA384 SHA512. When the signature in the input message is based on a hash algorithm that does not appear in the list, an error is produced and the fail_security edge is followed. The other possible hash names are: MD5, RIPEMD160, DOUBLE_SHA, MD2, TIGER_192, HAVAL_5_160. Some of these hash names are not necessarily weak, but they are not mentioned in RFC4880 and therefore are not standard in OpenPGP.

Edges

The following table lists and describes the edges that are returned by the PGP Decryption and Verification Service (com.ibi.agents.PGPDecryptAgent).

Edge Name	Description
success	The message was successfully unpackaged with PGP.
fail_parse	An iFL or XPath expression could not be evaluated.

Edge Name	Description
fail_security	The message could not be unpackaged or the signature hash algorithm is not acceptable.
fail_unsigned	The message was successfully unpacked but was unsigned and the Signature Required parameter is on.

Special Registers (SREGs)

The following table describes the Special Registers (SREGs) that are assigned upon successful unpackaging of the PGP message.

Special Register Name	Description
pgp_encrypted	A value of *true* is returned if the PGP message was encrypted. Otherwise, a value of *false* is returned.
pgp_signed	A value of *true* is returned if the PGP message was signed. Otherwise, a value of *false* is returned.
pgp_signer	The user ID of the first signer of the first signature.
pgp_filename	The file name attribute of the literal data packet.

Configuring the PGP Decryption and Verification Preparser

Syntax:

```
com.ibi.preparsers.PGPDecrypt
```

Description:

This preparser implements the same functionality as the PGP Decryption and Verification Service (com.ibi.agents.PGPDecryptAgent). For more information, see *Configuring the PGP Decryption and Verification Service* on page 19.

The preparser causes an error if the returned edge would be anything other than *success*.

Configuring the PGP Encryption and Signature Preemitter

Syntax:

```
com.ibi.preemit.PGPEncrypt
```

Description:

This preemitter implements exactly the same functionality as the PGP Encryption and Signature service (com.ibi.agents.PGPEncrypt). For more information, see *Configuring the PGP Encryption and Signature Service* on page 15.

The preemitter causes an error if the returned edge would be anything other than *success*.

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.