WMB Encryption Decryption Nodes

Shravan Kumar Kudikala
IBM-India Software Labs
shravankk@in.ibm.com

Take Note!

Before using this report, be sure to read the general information under "Notices".

First Edition, July 2007
This edition applies to Version 1.0 of WMB Encryption Decryption Nodes and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

NOTICES	***---***	***04***
OVERVIEW AND CONFIGURATION	***---***	***05***
INSTALLING THE PLUG-IN NODE	***---***	***09***
EXAMPLE USAGE OF THE NODE	***---***	***10***

Notices

Trademarks and service marks

Overview and Configuration

Overview:

This Support pack provides two nodes to encrypt and decrypt the message body of amessage traversing in WMB. The plug-in nodes are developed in Java and hence they can be used on any platform supported by the WMB V6 brokers. However problems might be encountered on some platforms as the JRE and its dependant packages might behave differently. The nodes are tested on the following platforms Solaris, AIX, Windows XP and found to work normally.

Two nodes are provided in this support pack to encrypt and decrypt the messages. To encrypt a message using password based encryption concept (as per message exchangers' choice - for the selected algorithm) use PasswordBasedEncryption node. To decrypt the message using password based decryption concept use PasswordBasedDecryption node.

The Cryptographic Nodes are developed for the following reasons:

- All protocols in WMB do not support SSL, hence messages sent/received from/to WMB can be tampered.
- Numerous complex methods are available to encrypt and decrypt the messages. Which one to choose is a complex decision.
- Writing own cryptographic algorithm needs strong development skills and is time consuming expensive work.
- Using the existing industry standard packages is easy but again it has large volume of complex algorithms. One needs to have deep skills to understand and use them.
- In WMB there is no standard package or node that would allow users to encrypt and decrypt messages on the fly.

What does this support pack provide you?

- Separate nodes were developed for encryption and decryption of messages to increase clarity in the usage of the nodes.
- The nodes are provided with simple node properties (for ease of use) so that one can encrypt/decrypt messages without having any knowledge of cryptography.
- Messages sent outside the scope of WMB environment can now be encrypted using various algorithms ensuring the confidentiality of the message.
- Messages received from other sources by the WMB environment might be encrypted. Using these nodes the message can be decrypted and further processed safely.
- Currently Password Based Cryptography is provided in this supportpac and all messages are expected to be in BLOB format only.
- The nodes initialization / termination and exceptions are logged into Operating System log to notify the users.

What's planned in the future releases?

- Extend the cryptographic method from Password Based to Key Pair Based where the users can use the public/private keys to encrypt / decrypt the messages.
- Provide custom built-in symmetric cryptographic methods to secure the messages.
- Extend the nodes to use specific security provider like IBM, Bouncy castle, etc.
- Extend the nodes such that users can plug-in their own cryptographic methods to the existing nodes and use their own algorithms programmatically.
- Provide security to secure the user specified password on the node property tab.

- ❖ Provide an interface so that customers can use the plug-in package outside the scope of message broker to encrypt / decrypt the messages.

Currently the nodes are developed only for Password Based cryptography concepts. The default provider being IBM, shipped with the JRE 142 which comes with WMB V6. Additionally, customers can consider using other providers like Bouncy Castle.

The following Password Based algorithms were tested using the nodes provided in this supportpac. During the tests we used IBM and Bouncy Castle as the security providers. Some of the below mentioned algorithms might not work when Bouncy Castle is not used. The below data is only for awareness and users need not have depth knowledge about them.

Algorithm	**Padding Scheme**	**Bits**
PBEWithMD5AndDES	PKCS5 Scheme 1	64
PBEWithMD5AndRC2	PKCS5 Scheme 1	128
PBEWithSHA1AndDES	PKCS5 Scheme 1	64
PBEWithSHA1AndRC2	PKCS5 Scheme 1	128
PBEWithSHAAnd40BitRC4	PKCS12	40
PBEWithSHAAnd128BitRC4	PKCS12	128
PBEWithSHAAndTwofish-CBC	PKCS12	256
PBEWithSHAAnd40BitRC2-CBC	PKCS12	40
PBEWithSHAAnd128BitRC2-CBC	PKCS12	128
PBEWithSHAAnd2-KeyTripleDES-CBC	PKCS12	128
PBEWithSHAAnd3-KeyTripleDES-CBC	PKCS12	192

Configuring the nodes:

Apart from the node configuration described below, no special configuration is needed to run the basic functionality of the supportpac. Some basic algorithms work fine with default IBM provider. In order to have all algorithms working, it is recommended to have Bouncy Castle installed and configured. The installation and configuration is narrated below. More details about Bouncy Castle are available at http://www.bouncycastle.org/latest_releases.html.

The way we have installed/configured Bouncy Castle during our tests is narrated here. The same can be followed by the customers.

1. Download the signed jar file "bcprov-jdk14-136.jar" and copy it to ext directory of WMB JRE. (<WMB-Install-Dir>\jre\lib\ext).
2. Add Bouncy Castle as the last provider in the java.security file located under security folder of WMB JRE. (<WMB-Install-Dir>\jre\lib\security) as shown below :

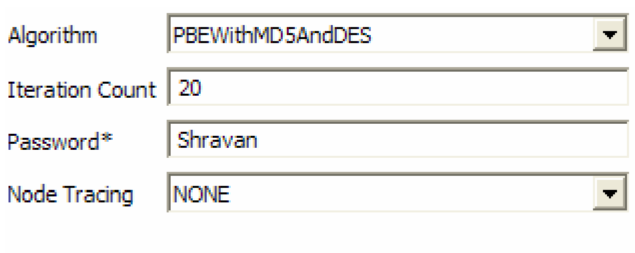
```
security.provider.1=com.ibm.jsse.IBMJSSEProvider
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=org.bouncycastle.jce.provider.BouncyCastleProv
ider
```

When large messages are used to encrypt or decrypt there can be failures. To overcome it customers need to request the "Unlimited strength jurisdiction policy file" from the JRE provider. The provider will provide two jar files (a) local_policy.jar and (b) US_export_policy.jar. Replace similar jar files available under "<WMB-Install-Dir>\jre\lib\security" with the jar files provided by the JRE provider.

Note: An encrypted message can be larger than the original message. When using WMQ as the protocol customers need to configure the queue size properly as the resulting message can be large.

All the algorithms that passed with the above configuration are listed under the node property "Algorithm" for both the nodes. We have limited our testing to the max message size of 52 MB in our test cases.

For both the nodes, PasswordBasedEncryption and PasswordBasedDecryption node, the properties are common which are explained below. Put an instance of the node on message flow editor. Right click on the node and select "Properties" option to configure the node properties. The following image shows the Basic properties of the nodes.



Algorithm	PBEWithMD5AndDES
Iteration Count	20
Password*	Shravan
Node Tracing	NONE

- ✓ *Property* **Algorithm**: From the drop down list select an algorithm that which you would like to use for encryption/decryption of the message. If no value is chosen, the default algorithm chosen is "PBEWithMD5AndDES". Use the same algorithm to encrypt/decrypt the message on the nodes. For ex:- If you are using "PBEWithMD5AndRC2" algorithm for encryption then same algorithm must be used for decryption of the same message.

- ✓ *Property* **Iteration Count**: Give iteration count number. It is recommended to have iteration count greater than 10. If no value is specified the node takes it as 30. Use the same value to encrypt / decrypt the message on the nodes. For ex:- If you are using 20 for encryption then the same 20 value must be used for decryption of the same message.

- ✓ *Property* **Password**: This is a mandatory field. Enter a password value. Use the same value to encrypt/decrypt the message on the nodes. For ex:- If you are using a value like "Shravan" for encryption then same value must be used for decryption of the same message.

- ✓ *Property* **Node Tracing**: Choose the appropriate level to trace the activity of the node. By default this is "NONE". It is encouraged to set NONE on production systems as the log refreshing mechanism isn't present in this node. The node appends the information into a single log file, irrespective of number of instances of the flows that use this node. The log file "CryptographicNodesLogData.txt" can be located in <WMB-install-dir>\bin on windows and on Unix machines it can be located in the directory from where the broker start command is issued. The same file is used by both the nodes. The Node Tracing value set here will have the following significance.

  NONE: No logging is done.
  ERROR: Log only if an error occurs in the node during message processing.
  INFO : Log basic node operations and ERROR(s) if any.
  DEBUG : Log very detailed information of the node, including every function it enters and exists along with ERROR(s).

Connecting the terminals:

The plug-in nodes route each message it processes successfully to the out terminal. If this fails, the message is routed to the failure terminal; you can connect nodes to this terminal to handle further processing. If you have not connected the failure terminal, the message is backed out to the input node.

Terminals and properties:

The plug-in node terminals are described in the following **Table - 1**.

Terminal	Description
In	The input terminal from which the message is routed to the node.
Out	The output terminal to which the message is routed if it is successfully processed.
Failure	The output terminal to which the message is routed if an error occurred. If this is not connected then the message is backed out to the input node.

Table – 1

Installing the plug-in node

❖ SupportPac contents

Download the supportpac and unzip into a temporary directory

The following directories will be present:

\WMB Encryption Decryption Nodes\V6Runtime\ - Contains files meant for WMB V6 runtime.
\WMB Encryption Decryption Nodes\V6Workbench\ - Contains files meant for WMB V6 Toolkit
\WMB Encryption Decryption Nodes\ia9w.pdf - Current user manual that you are reading

❖ Prerequisites

This Support Pac provides a plug-in node to be used with the IBM WebSphere Message Broker Version 6. If any changes are to be made to the plug-in node, then contact the author.

❖ Supported Platforms

This SupportPac has been developed and tested on Microsoft Windows XP, AIX and Sun Solaris environments. But this plug-in node can be run on any environment that is supported by the WMB Broker V6.

❖ Installing the Cryptographic nodes

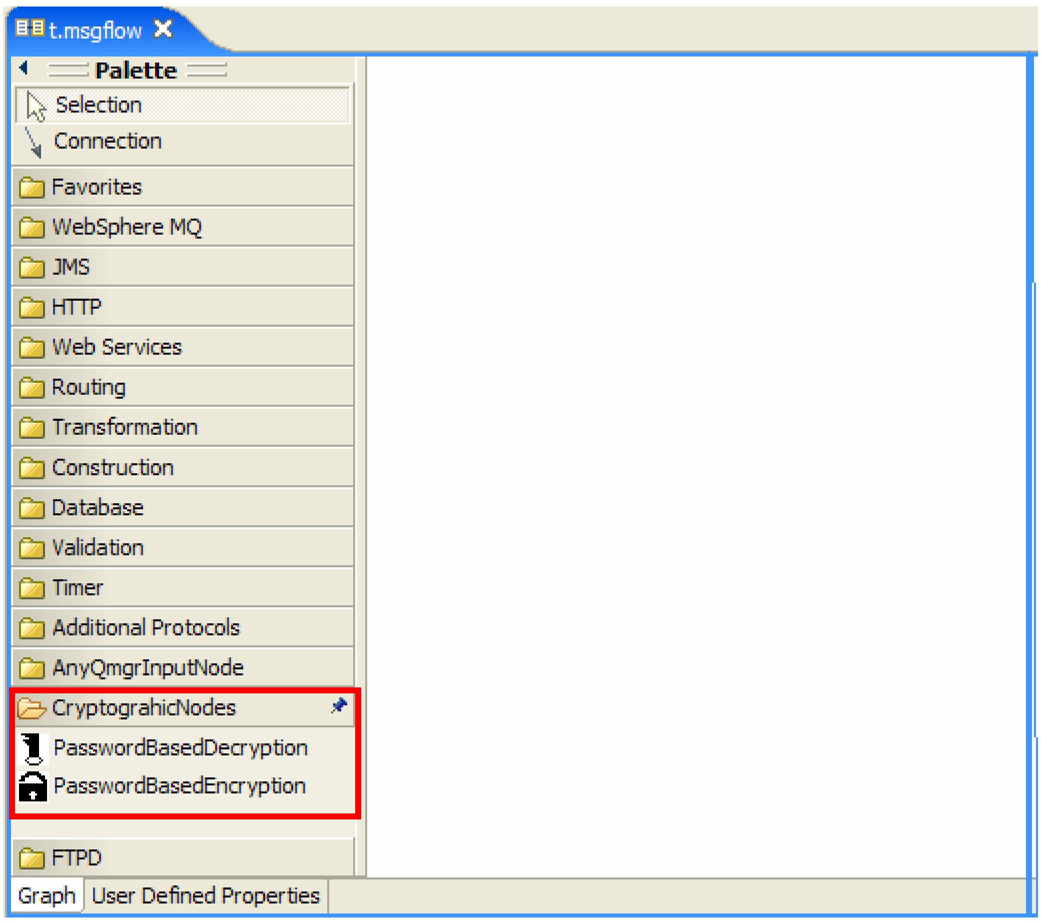
From the extracted files :->

- copy '\WMB Encryption Decryption Nodes \V6Runtime\CryptographicNodes.jar' file to '<WMB-Install–Directory>\jplugin'.
- Copy '\WMB Encryption Decryption Nodes \V6Workbench\com.ibm.broker.xternal.plugins.cryptographicnodes' directory to '<WMB-Toolkit-Install-Directory>\evtoolkit\eclipse\plugins' directory.
- Install "Unlimited strength jurisdiction policy" and "BounceCastle" files as described in "Configuring the nodes" section.
- Restart message broker (use mqsistart command).
- Restart the toolkit with "–clean" option (use wmbt.exe -clean)

Example usage of the nodes

This section demonstrates the usage of the nodes supplied in this supportpac.

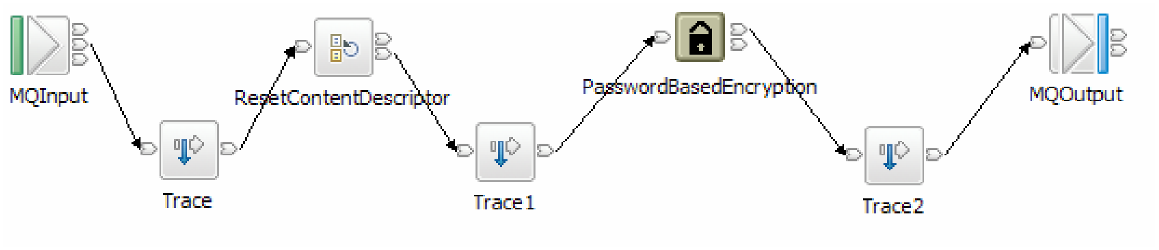
Install the plug-in as described above. After the successful installation of the node, you would see the two nodes (PasswordBasedEncryption and PasswordBasedDecryption) in the Message Flow Editor palette, as shown below:



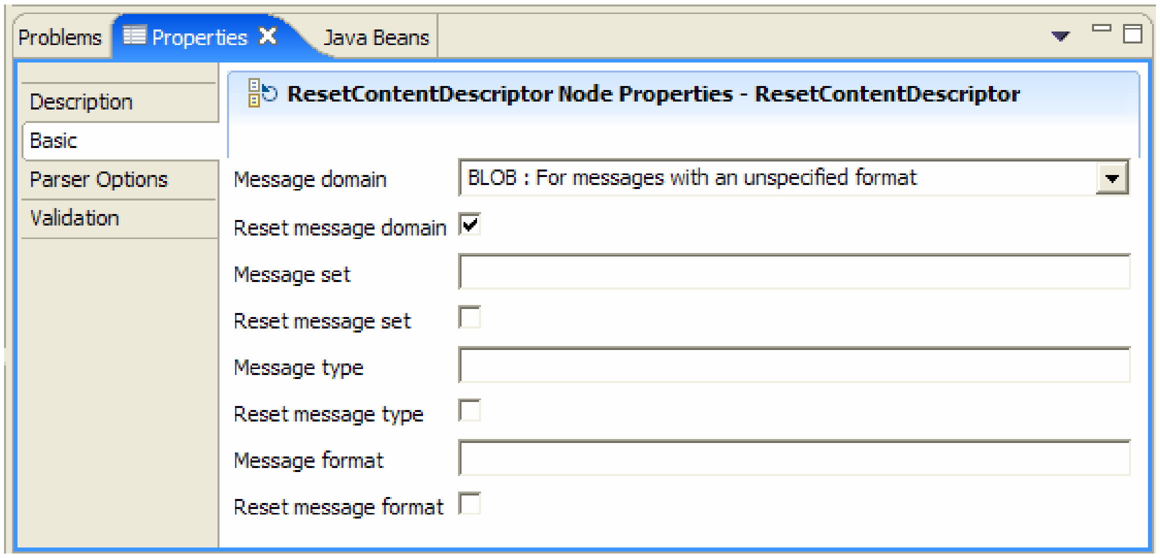
Choose the PasswordBasedEncryption node to encrypt the message body and PasswordBasedDecryption node to decrypt the message body. The headers are untouched and only the message body is encrypted / decrypted. The nodes expect only BLOB message data so use ResetContentDescriptor node available under the “Transformation” category to change the message domain.

Demonstrating how to encrypt the data :

In this example (please see below figure), the input message is of XML domain and is received from WMQ using MQInput node. As the PasswordBasedEncryption node expects only BLOB message body, ResetContentDescriptor node is used to convert the message domain from XML to BLOB. The trace nodes are used only for diagnostic purpose. A screen shot of the message flow is shown below.



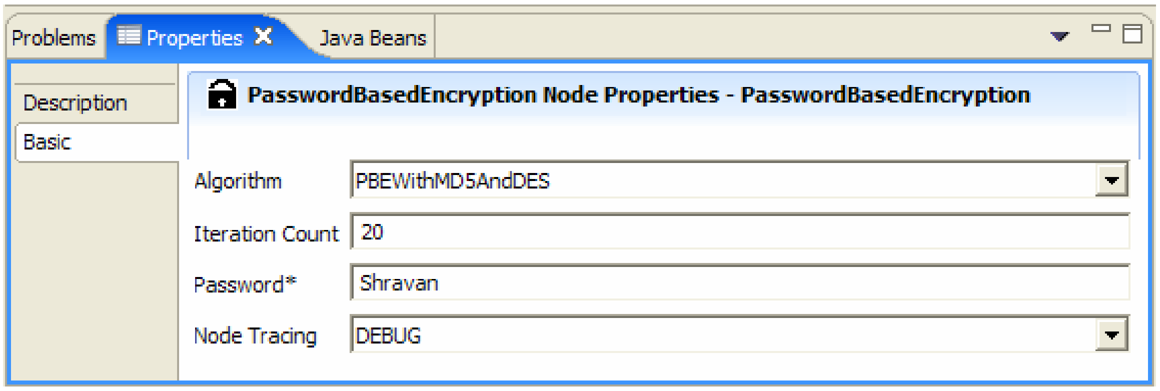
The properties of ‘ResetContentDescriptor’ node are set as shown below.



ResetContentDescriptor Node Properties - ResetContentDescriptor

Message domain	BLOB : For messages with an unspecified format
Reset message domain	✓
Message set	
Reset message set	☐
Message type	
Reset message type	☐
Message format	
Reset message format	☐

This converts the XML message domain to BLOB message domain. The properties of PasswordBasedEncryption node are set as shown below.



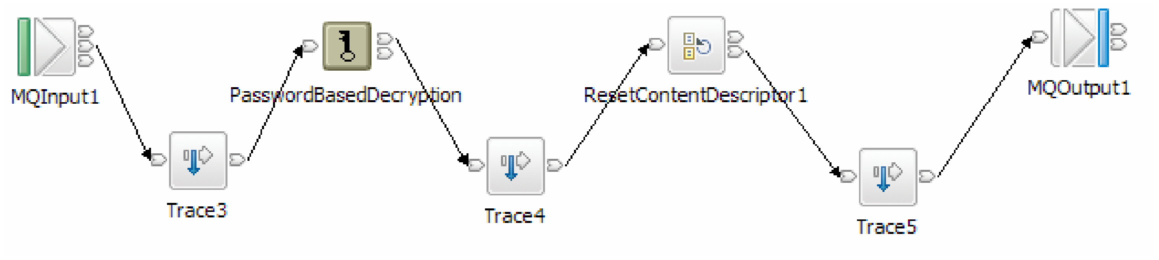
PasswordBasedEncryption Node Properties - PasswordBasedEncryption

Algorithm	PBEWithMD5AndDES
Iteration Count	20
Password*	Shravan
Node Tracing	DEBUG

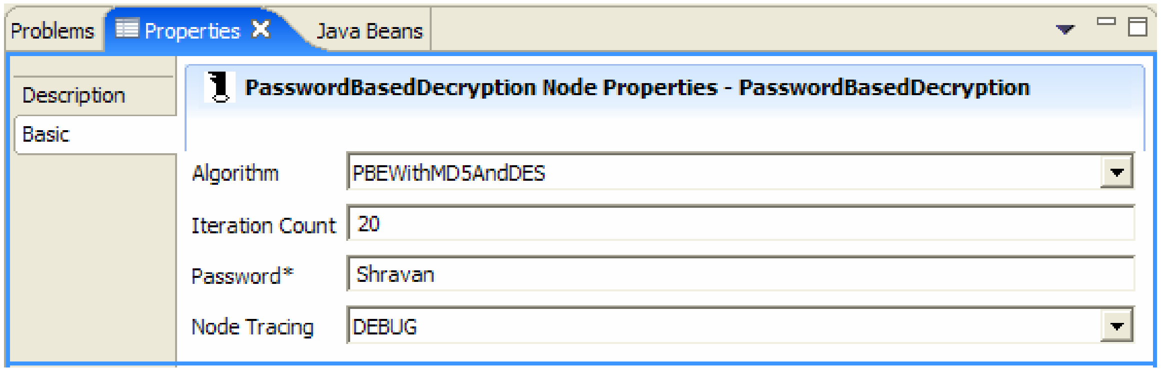
As per the example, the node uses the password based encryption algorithm “PBEWithMD5AndDES’ with iteration count 20 and password as “Shravan”. On the decryption node the same properties should and must be used to decrypt the same message. It is not possible to encrypt with one set of properties and decrypt with another set of properties. Once the message is encrypted at this node, the message which is in BLOB domain is put on the WMQ queue using MQOutput node.

Demonstrating how to decrypt the data :

In this example (please see below figure), the input message is of BLOB domain and is received from WMQ using MQInput node. The message we receive should be encrypted message. So, in this case for demonstration purpose, we will have the above flow output queue as the input queue of this flow. As the PasswordBasedDecryption node returns the decrypted message body in BLOB domain, use the ResetContentDescriptor node to convert the message domain to XML. The message can be encrypted by user application using password based encryption algorithms and can be decrypted using the PasswordBasedDecryption node, provided the node properties are chosen appropriately. The algorithm, iteration count and the password with which the message is encrypted, the same values should be specified on the node properties to decrypt the message. It is not possible to encrypt with one set of properties and decrypt with another set of properties. The trace nodes in this case are used only for diagnostic purpose. A screen shot of the message flow is shown below.



The properties of PasswordBasedEncryption node are set as shown below. The node expects the BLOB message in encrypted format and it decrypts the message and returns the BLOB message.

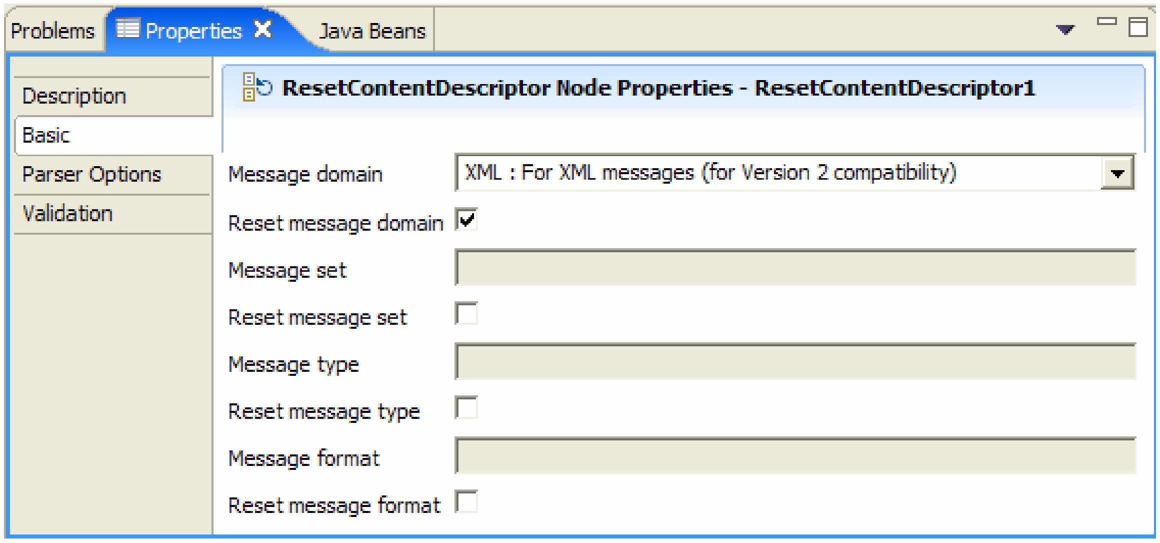


PasswordBasedDecryption Node Properties - PasswordBasedDecryption

Algorithm	PBEWithMD5AndDES
Iteration Count	20
Password*	Shravan
Node Tracing	DEBUG

As per the example, the node uses the password based decryption algorithm "PBEWithMD5AndDES' with iteration count 20 and password as "Shravan". Once the message is decrypted at this node, the message which is in BLOB domain is sent to the next node. Here we used the 'ResetContentDescriptor node to change the BLOB domain to XML domain as the original message which is encrypted is of XML domain.

The properties of 'ResetContentDescriptor' node are set as shown below. This converts the BLOB message domain to XML message domain.



ResetContentDescriptor Node Properties - ResetContentDescriptor1

Message domain: XML : For XML messages (for Version 2 compatibility)
Reset message domain: ✓
Message set:
Reset message set: ☐
Message type:
Reset message type: ☐
Message format:
Reset message format: ☐

The XML message returned from this node is sent to the WMQ Queue using the MQOutput Node. At this stage the input message sent to the first flow should be same as the output received on the last flow as the process of encryption is done in first flow and decryption is done in the second flow.

Using in client applications

If you are planning to do only one operation, either encryption or decryption in message broker and the other operation else where in your java client application then you need to use the plug-in jar file. For ex:

1. You have encrypted the data using the WebSphere Message Broker and your application has received the encrypted message. In order to decrypt the message in your application use the following method.

```
byte output [] =
    new PasswordBasedImplementation().decrypt(
    algorithm,
    password,
    iterationCount,
    encryptedMessageByteArray);
```

   where algorithm, password and iterationCount are java String objects which indicate the parameters used for encrypting the data in WebSphere Message Broker.

2. You want to encrypt the data using your application and send the encrypted message to WebSphere Message Broker so that it can be decrypted using the plug-in node. In order to encrypt the message in your application use the following method.

```
byte output [] =
    new PasswordBasedImplementation().encrypt(
    algorithm,
    password,
    iterationCount,
    inputMessageByteArray);
```

where algorithm, password and iterationCount are java String objects which indicate the parameters to be used for decrypting the data in WebSphere Message Broker.

3. You need to have the supplied plug-in jar file "CryptographicNodes.jar" in the classpath.
4. You need to import the following class in your java application.
   com.ibm.broker.xternal.plugins.cryptographicnodes.PasswordBasedImplementation;

A simple java client application is shown here to demonstrate how to use the encryption and decryption.

```
/*Java Source Code – testClient.java */

import com.ibm.broker.xternal.plugins.cryptographicnodes.PasswordBasedImplementation;

public class testClient
{
    public static void main(String[] args)
    {
            String inputMessage = "Hello Cryptography";
            String algorithm = "PBEWithSHAAnd40BitRC4";
            int iterationCount = 750;
            String password = "Shravan K Kudikala";
            try
            {
                    System.out.println("Message to be encrypted \""+inputMessage+"\"");
                    byte input[] = inputMessage.getBytes();
                    byte cypher[]= new
PasswordBasedImplementation().encrypt(algorithm,password,iterationCount,input);
                    System.out.println("Encrypted message \""+new String(cypher)+"\"");
                    byte output[] = new
PasswordBasedImplementation().decrypt(algorithm,password,iterationCount,cypher);
                   System.out.println("Message after decryption \""+new String(output)+"\"");
            }catch(Exception e)
            {
                    e.printStackTrace();
            }
    }
}
```