Hm, because some other people tried to use cryptography in Talend, I will post this demonstration of AES(Rijndael) with 256-bit key in CBC mode. This code you could tailore into Talend subroutine and use on your own. Maybe something like this will be implemented in a way of Talend component and you will be able to encrypt data from the design point of view...hope it helps...
Ladislav

```
package org.archenroot.crypto.aes;
/*
 * @(#)EncryptString.java
 *
 * Summary:  Application for encrypt string variables.
 *  AES(Rijndael) with 256-bit key in CBC mode.
 *  This is just a demonstration piece of code.
 *
 * Copyright:  (C) 2009-2011
 *
 * Licence: MIT
 * Copyright (c) 2011 Ladislav Jech
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use,
 * copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following
 * conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * Requires: JDK 1.6+
 *
 * Created with: Eclipse Galileo 3.5
 *
 * Version History:
 *   1.1 2011-02-08
 * Initial version
 *   1.2 2011-12-01
 * Added JCE Unlimited Strength check
 */
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.ShortBufferException;
```

```
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.Charset;
import java.security.GeneralSecurityException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import static java.lang.System.out;
/**
 * Application for encrypt string using AES with 256-bit key in CBC mode.
 *
 * You need to unlock maximal cryptography level of Java Run-time Environment.
 * It's about replacing local_policy.jar and US_export_policy.jar which can
 * obtained at http://www.oracle.com/technetwork/java/javase/downloads/index.html.
 * Name of extension is: Java Cryptography Extension (JCE) Unlimited Strength
 * Jurisdiction Policy Files 6 (verze pro JRE 6.*).
 *
 * <p/>
 * This version use AES/CBC/PKCS5Padding b/
 * */
public class EncryptString {
 // ------------------------------ CONSTANTS ------------------------------
/**
 * Configuration of used algorithm.
 * Do not use less secure DES. Setting up another value can prevent of
 * using this software and will require to specify some additional
 * parameters for initial vector.
 */
private static final String ALGORITHM = "AES";
/**
 * Configuration of block mode. Do not use less secure ECB mode.
 */
private static final String BLOCK_MODE = "CBC";
/**
 * Configuration of "padding".
 */
private static final String PADDING = "PKCS5Padding";
/**
 * Encoding used for converting bytes <--> Strings.
 */
private static final Charset CHARSET = Charset.forName("UTF-8");
/**
 * 128-bit of salt. Not so secure, but stable bytes for use with AES-CBC mode.
 */
private static final IvParameterSpec CBC_SALT = new IvParameterSpec(
  new byte[] { 7, 34, 56, 78, 90, 87, 65, 43, 12, 34, 56, 78, -123,
    87, 65, 43 });
/**
 * 256-bit static key. Schould be replaced by generated one for better security.
 * But for demonstration purposes it's built-in.
```

```
 */
private static final SecretKeySpec STATIC_KEY = new SecretKeySpec(
  new byte[] { 75, 108, 105, 99, 32, 107, 32, 104, 101, 115, 108,
    117, 109, 32, 101, 120, 116, 101, 114, 110, 105, 99, 104,
    32, 115, 121, 115, 116, 101, 109, 117, 32 }, ALGORITHM);
/*
 * To make it more independent against external library (ArrayUtils), here are defined
 * an empty arrays.
 */
/** Empty array of bytes. */
private static final byte[] EMPTY_BYTE_ARRAY = new byte;
/** Mask for 0. bit of byte. */
private static final int BIT_0 = 1;
/** Mask for 1. bit of byte. */
private static final int BIT_1 = 0x02;
/** Mask for 2. bit of byte. */
private static final int BIT_2 = 0x04;
/** Mask for 3. bit of byte. */
private static final int BIT_3 = 0x08;
/** Mask for 4. bit of byte. */
private static final int BIT_4 = 0x10;
/** Mask for 5. bit of byte. */
private static final int BIT_5 = 0x20;
/** Mask for 6. bit of byte. */
private static final int BIT_6 = 0x40;
/** Mask for 7. bit of byte. */
private static final int BIT_7 = 0x80;
/** Mask array. */
private static final int[] BITS = { BIT_0, BIT_1, BIT_2, BIT_3, BIT_4,
  BIT_5, BIT_6, BIT_7 };
/**
 * Encrypts message in clear text and write into the XML file, seek for
 * beginning TAG <Heslo>, finishing then </Heslo>.
 *
 * @param cipher
 *            type of cipher used for encrypt.
 * @param key
 *            tajný klí? pou?itý pro ?ifrování souboru.
 * @param file
 *            file, do kterého se zapí?e za?ifrovaná zpráva.
 * @param clearMessage
 *            ?istý text zprávy ur?ené k ?ífrování.
 *
 * @throws InvalidKeyException
 *             kdy? je n?co ?patn? s klí?em. M??e být zp?sobeno tím, ?e JRE
 *             nepou?ívá knihovny pro vysoký stupe? ?ifrování.
 * @throws IOException
 *             kdy? je n?jaký problém se zápisem fileu, nebo obecn? s
 *             operacemi vstupu/výstupu.
 * @throws InvalidAlgorithmParameterException
 *             kdy? je problém se "solí", nebo-li s parametrem CBC_SALT.
 * @throws BadPaddingException
 *             kdy? je problém s "vycpávkou" ?ifry. Vycpávka zvy?uje
 *             zabezpe?ení p?i brute-force pokusech o prolomení ?ifry.
 * @throws IllegalBlockSizeException
 *             kdy? je problém s neplatnou velikostí bloku ?ifry.
```

```
 * @throws ShortBufferException
 *             kdy? je zásobník p?íli? malý pro zpracovávaná data.
 */
public static void writeEncrypted(  Cipher cipher,
         SecretKeySpec key,
         String file,
         String clearMessage)
throws
InvalidKeyException,
IOException,
InvalidAlgorithmParameterException,
IllegalBlockSizeException,
BadPaddingException,
ShortBufferException
{
 // Objects related to file works.
 FileReader fileReader = new FileReader(new File(file));
 BufferedReader inputBuffer = new BufferedReader(fileReader);
 File temporaryFile = new File("temp.xml");
 FileWriter fileWriter = new FileWriter(temporaryFile);
 BufferedWriter outputBuffer = new BufferedWriter(fileWriter);
 // Initiation of cipher.
 cipher.init(Cipher.ENCRYPT_MODE, key, CBC_SALT);
 byte[] clearMessageByte = clearMessage.getBytes(CHARSET);
 byte[] encryptedMessageByte = cipher.doFinal(clearMessageByte);
 // Encrypt message and transfer it to encoded format Base64.
 final String encodedBase64 = new String(encode(encryptedMessageByte),
   CHARSET);
 /*
  * Inactive part - idea is to make it work directly in RAM instead
  * of file, but for demonstration it is enough.
  * long delka = new File(file).length(); MappedByteBuffer in = new
  * FileInputStream(file).getChannel().map(
  * FileChannel.MapMode.READ_ONLY, 0, delka); int i = 0; while (i <
  * delka) in. System.out.print((char) in.get(i++));
  */
 // Write temporary file, which already consists of encrypted content.
 String row = null;
 while ((row = inputBuffer.readLine()) != null) {
  if (row.indexOf("</Heslo>") > 0) {
   outputBuffer.write("\t<Heslo>");
   outputBuffer.write(encodedBase64);
   outputBuffer.write("</Heslo>");
   outputBuffer.write("\n");
  } else {
   outputBuffer.write(row);
   outputBuffer.write("\n");
  }
 }
 // Explicitly clean memory.
 inputBuffer.close();
 fileReader.close();
 outputBuffer.close();
 fileWriter.close();
 // Write data from temporary to live file.
 File f = new File(file);
```

```
if (f.delete()) {
 FileReader fileReader1 = new FileReader(new File(temporaryFile
   .getAbsoluteFile().toString()));
 BufferedReader inputBuffer1 = new BufferedReader(fileReader1);
 File outputFile = new File(file);
 FileWriter fileWriter1 = new FileWriter(outputFile);
 BufferedWriter outputBuffer1 = new BufferedWriter(
   fileWriter1);
 String outputRow = null;
 while ((outputRow = inputBuffer1.readLine()) != null) {
  outputBuffer1.write(outputRow);
  outputBuffer1.write("\n");
 }
 inputBuffer1.close();
 fileReader1.close();
 outputBuffer1.close();
 fileWriter1.close();
}
temporaryFile.deleteOnExit();
temporaryFile = null;
f = null;
/*
 * Check encrypting mechanims in a way that encrypted and encoded message
 * is again decoded and decrypted and printed to standard output. If the message
 * is different on input&output then it symptom of corrupted cryptography settings.
 */
cipher.init(Cipher.DECRYPT_MODE, key, CBC_SALT);
byte[] original = cipher.doFinal(decode(encodedBase64
   .getBytes("UTF-8")));
 System.out
   .println("Decoded end decrypted message for check: "
     + new String(original, CHARSET));
}
// ------------------------------ POMOCNÉ METODY ------------------------------
/**
 * Converts bytes to String of hexadecimal values.
 *
 * @param buf
 *            input array of bytes.
 * @return String of hexadecimal values of input buffer
 */
public static String jakoHexa(byte buf[]) {
 StringBuffer strbuf = new StringBuffer(buf.length * 2);
 int i;
 for (i = 0; i < buf.length; i++) {
  if (((int) buf & 0xff) < 0x10)
   strbuf.append("0");
  strbuf.append(Long.toString((int) buf & 0xff, 16));
 }
 return strbuf.toString();
}
/**
 * Converts array of clear binary data to array of ASCII 0 and 1 characters
 * for each bit of argument.
 *
 * @param raw
```

```
     *            raw binary data
     * @return array of byte bajtových data 0 a 1 for each bit of argument
     * @see org.apache.commons.codec.BinaryEncoder#encode(byte[])
     */
    public static byte[] doASCIIBajtu(byte[] raw) {
     if (isEmpty(raw)) {
      return EMPTY_BYTE_ARRAY;
     }
     // get length/8 * bytes with 3-bit jump to left from length
     byte[] l_ascii = new byte;
     /*
      * Lower index jj by 8 because of not recompute block in case on multiplication
      */
     for (int ii = 0, jj = l_ascii.length - 1; ii < raw.length; ii++, jj -= 8) {
      for (int bits = 0; bits < BITS.length; ++bits) {
       if ((raw & BITS) == 0) {
        l_ascii = '0';
       } else {
        l_ascii = '1';
       }
      }
     }
     return l_ascii;
    }
    /**
     * Decodes array of bytes where each each byte is like ASCII '0' nebo '1'.
     *
     * @param ascii
     *            each byte is like ASCII '0' nebo '1'
     * @return clear coded binary where each bit is like byte from byte array argument
     */
    public static byte[] fromASCII(byte[] ascii) {
     if (isEmpty(ascii)) {
      return EMPTY_BYTE_ARRAY;
     }
     // get length/8 * bytes with 3-bit jump to right from length
     byte[] l_raw = new byte;
     /*
      * Lower index jj by 8 because of not recompute block in case on multiplication
      */
     for (int ii = 0, jj = ascii.length - 1; ii < l_raw.length; ii++, jj -= 8) {
      for (int bits = 0; bits < BITS.length; ++bits) {
       if (ascii == '1') {
        l_raw |= BITS;
       }
      }
     }
     return l_raw;
    }
    /**
     * Converts array of binary data to array of ASCII 0 a 1characters.
     *
     * @param binary
     *            clear binary data for convert
     * @return array of bytes ASCII of 0 a 1 values, each byt for 1 bit of argument
     * @see org.apache.commons.codec.BinaryEncoder#encode(byte[])
```

```
 */
public static byte[] encode(byte[] binary) {
 return doASCIIBajtu(binary);
}
/**
 * Decodes array of bytes where each byte represents ASCII '0' or '1'.
 *
 * @param ascii
 *           each byte represents ASCII '0' or '1'
 * @return clean coded binary data where each bit is related to one byte in input
 *         array of bytes argument
 * @see org.apache.commons.codec.Decoder#decode(Object)
 */
public static byte[] decode(byte[] ascii) {
 return fromASCII(ascii);
}
/**
 * Returns <code>true</code> if input array is <code>null</code> or
 * empty (length 0)
 *
 * @param array
 *            input array
 * @return <code>true</code> if input array is <code>null</code> or
 *         empty (length 0)
 */
private static boolean isEmpty(byte[] array) {
 return array == null || array.length == 0;
}
```

```
public static void testEnvironment()
throws
NoSuchAlgorithmException,
NoSuchPaddingException,
GeneralSecurityException
{

 // Testing 64-bit block
 byte[] data = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };
 // Creates 128-bit key from raw bytes
 SecretKey key128 = new SecretKeySpec(new byte[] { 0x00, 0x01, 0x02,
   0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
   0x0d, 0x0e, 0x0f }, ALGORITHM);
 // Create ciphre and try to encrypt data block with defined key
 Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + BLOCK_MODE + "/"
   + PADDING);
 cipher.init(Cipher.ENCRYPT_MODE, key128, CBC_SALT);
 cipher.doFinal(data);
 System.out.println("128-bit test result: O.K.");
 // Creates 192-bit key from raw bytes
 SecretKey key192 = new SecretKeySpec(new byte[] { 0x00, 0x01, 0x02,
   0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
   0x0d, 0x0e, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16,
   0x17 }, ALGORITHM);
 // Try to encrypt data block with new key
 cipher.init(Cipher.ENCRYPT_MODE, key192, CBC_SALT);
 cipher.doFinal(data);
 System.out.println("192-bit test result: O.K.");
```

```
 // Creates 256-bit key from raw bytes
 SecretKey key256 = new SecretKeySpec(new byte[] { 0x00, 0x01, 0x02,
   0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
   0x0d, 0x0e, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16,
   0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },
   ALGORITHM);
 // Try to encrypt data block with new key
 cipher.init(Cipher.ENCRYPT_MODE, key256, CBC_SALT);
 cipher.doFinal(data);
 System.out.println("256-bit test result: O.K.");
 System.out.println("Testing JCE restrinctions finished.");
 cipher = null;
}
// --------------------------- main() method ---------------------------
/**
 * Application for encrypt message using AES with 256-bit key in CBC mode.
 *
 * @param args
 *             Input parametric XML file. Message for encrypt.
 *
 * @throws NoSuchAlgorithmException
 *             if AES is not supported.
 * @throws NoSuchPaddingException
 *             if PKCS5 padding is not supported.
 * @throws InvalidKeyException
 *             if there is something bad with key.
 * @throws IOException
 *             if there is issue with file works.
 * @throws java.security.InvalidAlgorithmParameterException
 *             if there is issue with CBC_SALT(salting).
 */
public static void main(String[] args)
throws
InvalidAlgorithmParameterException,
InvalidKeyException,
IOException,
NoSuchAlgorithmException,
InvalidAlgorithmParameterException,
NoSuchPaddingException,
Exception
{
 // For test purposes
 // String file = "Test.xml";
 // String message = "Some string to encrypt;
 // If no arguments just write out help.
 if (args.length == 0) {
  testEnvironment();
  return;
 } else if (args.length == 2) {
  // Test JCE
  testEnvironment();
  // Initiation of encryption algorithm.
  out.println("Initiation of ciphre AES in CBC mode with \"padding\" PKCS5Padding");
  Cipher cipher = Cipher.getInstance(ALGORITHM + "/" + BLOCK_MODE
    + "/" + PADDING);
  out.println("Initiation of ciphre O.K.");
```

```
    // Initiation of key.
    SecretKeySpec key = STATIC_KEY;
    out.println("Initiation of static key O.K.");
    out.println("Start encrypting >>> ...");
    // Attach and init variables of input parameters.
    String file = args.toString();
    String message = args.toString();
    // For testing.
    // String file = "/home/zangetsu/file.xml";
    // String message = "AES cifrovani v praxy";
    // Write input message to input file in encrypted and encoded form.
    writeEncrypted(cipher, key, file, message);
    out.println("End of encrypting >>> O.K.");
    cipher = null;
   } else {
    // Bad number or types of arguments.
   }
  }
 }
```