Bouncy Castle - .Net Implementation – Triple DES Algorithm

In this article, we will go through a simple example to demonstrate Encryption and Decryption techniques using Triple DES algorithms. We will use Bouncy Castle APIs for performing below steps.

Encryption steps -

1. **Select an input message** - Input message can be in the form of a string which the user enters, or a stream e.g. File stream, memory stream or Network stream. There can be two cases depending upon the input length –
   - a. The length is modulo 8 bytes i.e. it is possible to divide the input message into blocks of 8 bytes. In this case, there would be no padding needed.
   - b. If the length is not in multiples of 8, we would need padding to compete the last block. In this scenario, we should consider the PaddedBufferedBlockCipher class, which pads according to the selected industry standards.
2. **Generate a Triple DES key. (secret key)** - We will use the CipherKeyGenerator class from the Bouncy Castle apis. `CipherKeyGenerator` is the base class for symmetric key generation The Cipher Key Generator needs to know the strength of the key as well as the Cipher for which to generate the key. In the following examples, the ciphers are DESEDE with ECB mode and Triple DES with CBC Cipher as cipher mode.
3. **Create a Triple DES algorithm object and select the ECB mode** - Triple DES cipher can be created either directly instantiating the DESESE engine or using the CipherUtilities class which has a collection of various ciphers along with the mode and the Padding information.
4. **Initialize the Cipher** - We have to initialize the cipher with the symmetric key. Additionally, we need to specify if we are going to use the cipher for encryption or for decryption mechanism.
5. Encrypt the block.

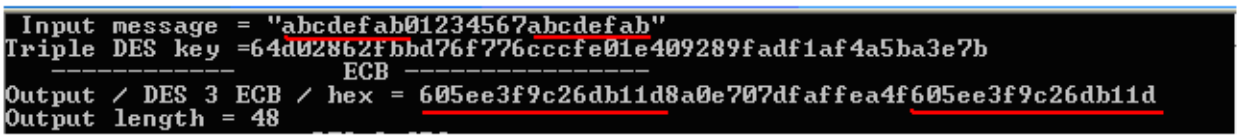
Code:

```
/// <summary>
/// generating a Triple DES key
/// </summary>
public void InitDES3Key()
{
        CipherKeyGenerator cipherKeyGenerator = new CipherKeyGenerator();
        cipherKeyGenerator.Init(new KeyGenerationParameters(new
SecureRandom(), 192));
        //192 specifies the size of key in bits i.e 24 bytes

        keyDES3 = cipherKeyGenerator.GenerateKey();
        BigInteger bigInteger = new BigInteger(keyDES3);
        Console.WriteLine(bigInteger.ToString(16));
}
```

```
/// <summary>
/// Encryption using DES3 algorithm in ECB mode
/// </summary>
/// <param name="message">Input message bytes</param>
/// <returns>Encrypted message bytes</returns>
public byte[] EncryptDES3(byte[] message)
{
        DesEdeEngine desedeEngine = new DesEdeEngine();
        BufferedBlockCipher bufferedCipher = new
BufferedBlockCipher(desedeEngine);

        // Create the KeyParameter for the DES3 key generated.
        KeyParameter keyparam =
ParameterUtilities.CreateKeyParameter("DESEDE", keyDES3);
        byte[] output = new
byte[bufferedCipher.GetOutputSize(message.Length)];
        bufferedCipher.Init(true, keyparam);
        output = bufferedCipher.DoFinal(message);
}
```

Let us analyze the input message and encrypted output generated.

Input message has a block of bytes that is occurring twice. Since we have selected ECB mode (default mode), each block is encrypted separately with the cipher key. Therefore, the output message also, has repetition of encrypted block corresponding to the repeated input block



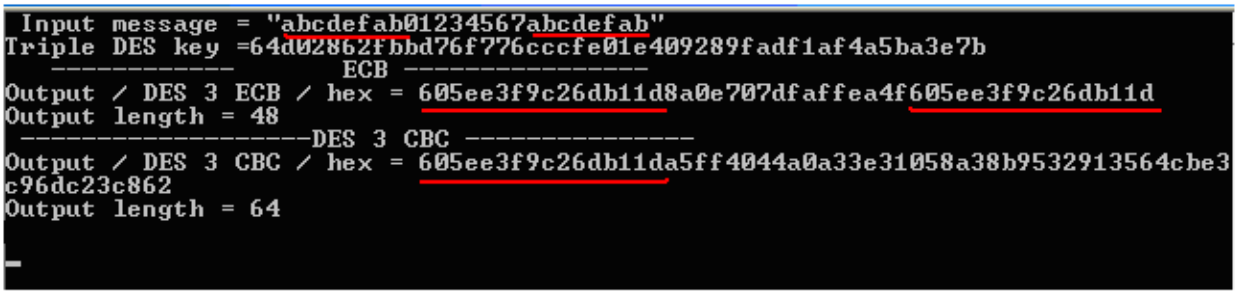
An attacker can probably recognize the pattern of repetition and thus have clue about the input message.

Lets us see the implementation using the CBC mode.

Code:

```
/// <summary>
/// Encryption using DES3 algorithm in CBC mode
/// </summary>
/// <param name="message">Input message bytes</param>
/// <returns>Encrypted message bytes</returns>
```

```
public byte[] EncryptDES3_CBC(byte[] message)
{
        DesEdeEngine desedeEngine = new DesEdeEngine();
        BufferedBlockCipher bufferedCipher = new PaddedBufferedBlockCipher(new
CbcBlockCipher(desedeEngine));
        KeyParameter keyparam =
ParameterUtilities.CreateKeyParameter("DESEDE", keyDES3);
        byte[] output = new
byte[bufferedCipher.GetOutputSize(message.Length)];
        bufferedCipher.Init(true, keyparam);
        output = bufferedCipher.DoFinal(message);
}
```



On analyzing the output generated by DES 3 CBC mode, we find that the first block is same as the first block of output by DES3 ECB mode, but the remaining blocks are never repeated. This is because the CBC mode uses the feedback mechanism and encryption each block depends upon the previous encrypted block.

Bouncy Castle - .Net Implementation – AES Algorithm

As mentioned previously, AES algorithms are stronger than DES and Triple DES algorithms. The below example implements an AES encryption logic using the CFB mode. Also, we pass an Initialization vector to the cipher, which is used for process the first cipher block.

Encryption Steps

1. Select an input message
2. Generate the AES key by specifying the strength needed for the key.
3. Instantiate the cipher object for AES scheme
4. Initialize the key parameter with Initialization Vector
5. Encrypt the message

Code:

```
/// <summary>
/// Encrypts Original Message using AES algorithm
/// </summary>
/// <param name="message">Original Message</param>
/// <param name="iVector">initialization Vector</param>
/// <returns>Encrypted Bytes</returns>
public byte[] AESEncryption(byte[] message, byte[] iVector)
{
        CipherKeyGenerator aesKey = new CipherKeyGenerator();
        aesKey.Init(new KeyGenerationParameters(new SecureRandom(), 128));
        byte[] m Key = aesKey.GenerateKey();
        KeyParameter aesKeyParam =
ParameterUtilities.CreateKeyParameter("AES", m_Key);

        // Setting up the Initialization Vector. IV is used for encrypting the
first block of input   message
        ParametersWithIV aesIVKeyParam = new ParametersWithIV(aesKeyParam,
iVector);

        // Create the cipher object for AES algorithm using CFB mode and No
Padding.
        IBufferedCipher cipher =
CipherUtilities.GetCipher("AES/CFB/NoPadding");
        cipher.Init(true, aesIVKeyParam);
        byte[] output = cipher.DoFinal(message);
}
```

Source: http://www.go4expert.com/articles/bouncy-castle-net-implementation-triple-t24829/