

Table of Contents

INTRODUCTION:	2
TARGET AUDIENCE AND PURPOSE OF THE DOCUMENT:	2
SIGNIFICANCE OF TAG 90 AND TAG 91:	2
TAG 90:	2
TAG 91:	2
TRIPLEDES IMPLEMENTATION:	3
RECOMMENDATION:	4
JAVA IMPLEMENTATION:	5
C# IMPLEMENTATION:	6
MANAGED C++ IMPLEMENTATION:	7

Introduction:

TripleDES is a standard encryption algorithm used to encrypt the sensitive information during the data transmission across the network. In FIX Gateway, TripleDES (CBC Mode with PKCS 5 Padding) is implemented to encrypt connecting entity's password and new password information during the logon request processing.

Target audience and purpose of the document:

This document serves as a handy guide to interact with FIX gateway specifically involving message that uses encryption. The code snippet included in this document is just for reference and the hosting exchange is not liable for accuracy of the same. It is recommended that this document should be read in conformation with the Open Interface Document.

ISVs are expected to verify their application in test environment before implementing encryption in production environment.

Significance of Tag 90 And Tag 91:

The values in tag 90 and tag 91 (both tags are the part of standard message header) are used to communicate the password information in an encrypted form during the logon processing on the FIX gateway.

Tag 90:

The value of Tag 90 signifies the length of ensuing encrypted information in Tag 91. The value of Tag 90 will always be unencrypted. It demarcates the boundary of encrypted password information.

Tag 91:

The value of Tag 91 is an actual encrypted password/new password information. This information is to be encrypted using TripleDES algorithm. Delving further in the content of value of Tag 91, it is to be noted that after encrypting the required information, ASCII value of each character of encrypted information is to be represented in its hexadecimal form with the fix length of 2 with leading zero.

Example:

For an encrypted information [a x ½ P z W ¿], its representation will be [61 78 AB 50 7A 57 A8].

TripleDES Implementation:

To encrypt any information in TripleDES, it requires having one Key and the other IV factor. For simplicity, we can refer to Key as Key1 and IV factor as Key2. The length of Key (Key1) should be 24 (fixed) and the length of IV factor (Key2) should be 8 (fixed). Please note that both the keys are case sensitive.

The format for the Key (Key1) will be as follows:

[<IV Factor (Key2)><Next 16 characters as published by the Exchange>]

IV Factor (Key2) will be the current password. In case when password length is less than 8 characters then the required no of '|' will be appended to make it of length 8.

Example:

If the current password is [**abc.123**] then the IV Factor (Key2) value will be [**abc.123|**].

If the current password is [**abc.1234**] then expected IV factor is [**abc.1234**].

The encryption key (Key1) and IV factor (Key2) as described above should be used to encrypt the password/new password information.

Example:

Case 1: Login with an existing password

16 characters published by exchange is [**~!@#\$\$%^&*={};<>?**]

Current Password is [**abc.123**]

Key1 will be [**abc.123~!@#\$\$%^&*={};<>?**]

Key2 will be [**abc.123|**]

Information to be encrypted will be actual password i.e. [**abc.123**]

(Here, actual password [**abc.123**] is considered not the [**abc.123|**])

Case 2: New password to be set

16 characters published by exchange is [**~!@#\$\$%^&*={};<>?**]

Current Password is [**abc.1234**]

New password is [**xyz.6757**]

Key1 will be [**abc.1234~!@#\$\$%^&*={};<>?**]

Key2 will be [**abc.1234**]

Information to be encrypted will be the actual password i.e. [**abc.1234,xyz.6757**]

Recommendation:

It is recommended that the 16 characters as published by the Exchange should be saved in the UI such that the user will have to type it only once during the first logon. Rest of the time it will be read directly from the saved location. It is also recommended that the 16 characters as published by the Exchange should not be masked viz. with no asterisk or any other representation.

Code Snippet For Reference

Following are the code snippets of TripleDES implementation in different languages:

Java Implementation:

```
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class CTripleDESImplementation
{
    private String key;        //Key
    private String initializationVector;    //IV Factor

    //Constructor
    public CTripleDESImplementation(String key, String
                                   initializationVector)
    {
        this.key = key;
        this.initializationVector = initializationVector;
    }

    //Encryption method. Paramter being the string to be
    //encrypted.
    public byte[] encrypt(String plainText) throws Exception
    {
        byte[] plaintext = plainText.getBytes();

        byte[] tdesKeyData = key.getBytes();
        byte[] myIV = initializationVector.getBytes();

        //Instantiating Cipher class instace
        Cipher c3des =
            Cipher.getInstance("DESede/CBC/PKCS5Padding");

        SecretKeySpec myKey = new SecretKeySpec(tdesKeyData,
                                                "DESede");
        IvParameterSpec ivspec = new IvParameterSpec(myIV);

        //ENCRYPT_MODE
        c3des.init(Cipher.ENCRYPT_MODE, myKey, ivspec);
        byte[] cipherText = c3des.doFinal(plaintext);

        //Convert the received byte array in hexadecimal
        //representation. Character range will be from 0 to 255.

        return cipherText; // return hexadecimal representation of
                           //ciphertext.
    }
}
```

C# Implementation:

```
using System;
using System.Diagnostics;
using System.Security.Cryptography;
using System.Text;

public class CTripleDESImplementation
{
    private byte[] EncryptionKey;    //Key
    public byte[] EncryptionKeyStr
    {
        get { return EncryptionKey; }
        set { EncryptionKey = value; }
    }

    private byte[] IV;    //IV Factor
    public byte[] IVStr
    {
        get { return IV; }
        set { IV = value; }
    }

    //Encryption method. Parameter being string to be encrypted,
    public byte[] Encrypt(byte[] textToEncrypt)
    {
        TripleDESCryptoServiceProvider tdes = new
            TripleDESCryptoServiceProvider();

        tdes.Key = EncryptionKey;
        tdes.IV = IV;

        byte[] encryptedBuffer =
            tdes.CreateEncryptor().TransformFinalBlock(textToEncrypt, 0,
                textToEncrypt.Length);

        //Convert the received byte array in hexadecimal representation
        //Character range should be 0 to 255.

        return encryptedBuffer;// return hexadecimal representation of
            //encryptedBuffer.
    }
}
```

Managed C++ Implementation:

```
#pragma once
using namespace System;
using namespace System::Security::Cryptography;
using namespace System::Text;
using namespace System::Globalization;
using namespace System::Diagnostics;
using namespace System::Runtime::InteropServices;

public ref class CTripleDESImplementation
{
private:
    array<Byte>^ m_cEncryptionKey;    //Key
    array<Byte>^ m_cIV;              //IV Factor

public:
    array<Byte>^ GetEncryptionKey()
    {
        return m_cEncryptionKey;
    }

    void SetEncryptionKey(String^ pcEncryptionKey)
    {
        m_cEncryptionKey
        = Encoding::ASCII->GetBytes(pcEncryptionKey);
    }

    array<Byte>^ GetIV()
    {
        return m_cIV;
    }

    void SetIV(String^ pcIV)
    {
        m_cIV = Encoding::ASCII->GetBytes(pcIV);
    }

    //Constructor
    CTripleDESImplementation(String^ pcEncryptionKey,
                             String^ pcIV)
    {
        SetEncryptionKey(pcEncryptionKey);
        SetIV(pcIV);
    }

    //Encrypt method. Parameter being the string to be encrypted.
    array<Byte>^ Encrypt(array<Byte>^ cPlainText)
    {
        TripleDESCryptoServiceProvider^ tdes = gcnew
            TripleDESCryptoServiceProvider();

        tdes->Key      = m_cEncryptionKey;
        tdes->IV       = m_cIV;
    }
}
```

```
array<Byte>^ CipherText =
tdes->CreateEncryptor()->TransformFinalBlock
    (cPlainText, 0, cPlainText->Length);

//Convert the received byte array in hexadecimal
//representation
//Character range should be 0 to 255.

return CipherText; //return hexadecimal representation of
                    //CipherText
}
};
```