How to encrypt and decrypt a file by using Visual C#

For a Microsoft Visual Basic .NET version of this article, see 301070 .

This article refers to the following Microsoft .NET Framework Class Library namespaces:

- **System.IO**
- **System.Security**
- **System.Security.Cryptography**

Note This article does not apply to the Microsoft .NET Framework 2.0.

Summary

This article describes how to use the cryptography classes that are provided by the Microsoft .NET Framework to encrypt a text file to an unreadable state, and then to decrypt that text file back to its original format.

Requirements

The following list outlines the recommended hardware, software, network infrastructure, and service packs that you must have:

- Microsoft Windows 2000 Professional, Windows 2000 Server, Windows 2000 Advanced Server, Windows NT 4.0 Server or Microsoft Windows XP Professional

- Microsoft Visual Studio 2005 or Microsoft Visual Studio .NET

Encryption and decryption

The **System.Security.Cryptographic** namespace in the Microsoft .NET Framework provides a variety of tools to help you with encryption and with decryption. The **CryptoStream** class is one of the many classes that is provided. The **CryptoStream** class is designed to encrypt or to decrypt content as it is streamed out to a file.

Encrypt a file

To encrypt a file, follow these steps:

1. Start Visual Studio 2005 or Visual Studio .NET.

2. Click **Visual C#** under **Projects**, and then click **Console Application**under **Templates**. Visual C# .NET creates a **Static** class for you, together with an empty **Main()** procedure.

3. Use the **using** statement (as indicated in the sample code that follows) on the following namespaces:
   - **System**

   - **System.Security**

- **System.Security.Cryptography**
- **System.Text**
- **System.IO**

so that you do not have to qualify declarations from these namespaces later in your code. You must use these statements before any other declarations.

```
using System;
using System.IO;
using System.Security;
using System.Security.Cryptog
raphy;
using System.Runtime.InteropS
ervices;
using System.Text;
```

4. Generate a secret key to encrypt and to decrypt the data. The **DESCryptoServiceProvider** is based on a symmetric encryption algorithm. The symmetric encryption requires a key and an initialization vector (IV) to encrypt the data. To decrypt the data, you must have the same key and the same IV. You must also use the same encryption algorithm. You can generate the keys by using either of the following methods:
   - **Method 1** You can prompt the user for a password. Then, use the password as the key and the IV.
   - **Method 2** When you create a new instance of the symmetric cryptographic classes, a new key and IV are automatically created for the session. Use the key and IV that are generated by the managed symmetric cryptographic classes to encrypt and to decrypt the file.

     For more information about how to generate and distribute keys, see the Microsoft .NET Framework SDK Documentation, or see the following Microsoft Developer Network (MSDN) Web site:

5. Add the following function to generate a new key for a session (as noted in Method 2 of step 4):

```
//  Call this function to rem
ove the key from memory after
use for security.
[System.Runtime.InteropServic
es.DllImport("KERNEL32.DLL",
EntryPoint="RtlZeroMemory")]
public static extern bool Zer
oMemory(ref string Destinatio
n, int Length);

// Function to Generate a 64
bits Key.
static string GenerateKey()
{
// Create an instance of Syme
tric Algorithm. Key and IV is
generated automatically.
DESCryptoServiceProvider desC
rypto =(DESCryptoServiceProvi
der)DESCryptoServiceProvider.
Create();

// Use the Automatically gene
rated key for Encryption.
return ASCIIEncoding.ASCII.Ge
tString(desCrypto.Key);
}
```

6. Create a method in your class that is named **EncryptFile**. The **EncryptFile** class must have the following three parameters:
   - *sInputFilename*
   - *sOutputFilename*
   - *sKey* (The secret key that is used to encrypt and decrypt the file.)

```
static void EncryptFile(strin
```

```
g sInputFilename,
string sOutputFilename,
string sKey)
```

7. In the **EncryptFile** procedure, create an input **FileStream** object and an output **FileStream** object. These objects can be read from and written to the target files.

```
FileStream fsInput = new File
Stream(sInputFilename,
FileMode.Open,
FileAccess.Read);

FileStream fsEncrypted = new
FileStream(sOutputFilename,
FileMode.Create,
FileAccess.Write);
```

8. Declare an instance of the **DESCryptoServiceProvider** class. This represents the actual encryption and the actual decryption technology that is used on the files. At this point, you can create a different provider if you prefer to use RSAsecutiry or another cryptographic technique.

```
DESCryptoServiceProvider DES
= new DESCryptoServiceProvide
r();
```

9. The cryptographic provider must be provided with your secret key as an array of bytes. The **System.Text** namespace provides a function that is named **GetBytes()**. As part of its encoding features, the **GetBytes()** function takes a string, and then returns an array of bytes. The size of the key is different for each cryptographic technique. For example, Data Encryption Standard (DES) takes a 64-bit key that is equal to 8 bytes or to 8 characters.

   If you do not provide a key, the provider

randomly generates one. This successfully encrypts the file, but there is no way to decrypt the file. Note that you must also provide the initialization vector (IV). This value is used as part of the encryption. Like the key, the IV is randomly generated if you do not provide the value. Because the values must be the same for the encryption and the decryption, you must not permit random generation of these values.

```
DES.Key = ASCIIEncoding.ASCII
.GetBytes(sKey);
DES.IV = ASCIIEncoding.ASCII.
GetBytes(sKey);
```

10. Create an instance of the **CryptoStream** class by using the cryptographic provider to obtain an encrypting object (**CreateEncryptor**) and the existing output **FileStream** object as a part of the constructor.

```
ICryptoTransform desencrypt =
DES.CreateEncryptor();
CryptoStream cryptostream = n
ew CryptoStream(fsEncrypted,
desencrypt,
CryptoStreamMode.Write);
```

11. Read in the input file, and then write out to the output file. Pass through the **CryptoStream** object where the file is encrypted by using the key that you provided.

```
byte[] bytearrayinput = new b
yte[fsInput.Length - 1];
fsInput.Read(bytearrayinput,
0, bytearrayinput.Length);
cryptostream.Write(bytearrayi
nput, 0, bytearrayinput.Lengt
h);
```

Decrypt a file

To decrypt a file, follow these steps:

1. Create a method, and then name it DecryptFile. The decryption process is similar to the encryption process, however, the **DecryptFile** procedure has two key differences from the **EncryptFile** procedure.
   - **CreateDecryptor** is used instead of **CreateEncryptor** to create the **CryptoStream** object, that specifies how the object can be used.

   - When the decrypted text is written to the destination file, the **CryptoStream** object is now the source instead of the destination stream.

```
static void DecryptFile(strin
g sInputFilename,
                string sOutpu
tFilename,
                string sKey)
{
DESCryptoServiceProvider DES
= new DESCryptoServiceProvide
r();
//A 64 bit key and IV is requ
ired for this provider.
//Set secret key For DES algo
rithm.
DES.Key = ASCIIEncoding.ASCII
.GetBytes(sKey);
//Set initialization vector.
DES.IV = ASCIIEncoding.ASCII.
GetBytes(sKey);

//Create a file stream to rea
d the encrypted file back.
FileStream fsread = new FileS
tream(sInputFilename,

FileMode.Open,

FileAccess.Read);
//Create a DES decryptor from
the DES instance.
ICryptoTransform desdecrypt =
```

```
DES.CreateDecryptor();
//Create crypto stream set to
read and do a
//DES decryption transform on
incoming bytes.
CryptoStream cryptostreamDecr
= new CryptoStream(fsread,

desdecrypt,

CryptoStreamMode.Read);
//Print the contents of the d
ecrypted file.
StreamWriter fsDecrypted = ne
w StreamWriter(sOutputFilenam
e);
fsDecrypted.Write(new StreamR
eader(cryptostreamDecr).ReadT
oEnd());
fsDecrypted.Flush();
fsDecrypted.Close();
}
```

2. Add the following lines to the **Main()** procedure to call both **EncryptFile** and **DecryptFile**:

```
static void Main()
{
      // Must be 64 bits, 8 b
ytes.
      // Distribute this key
to the user who will decrypt
this file.
      string sSecretKey;

      // Get the key for the
file to encrypt.
      sSecretKey = GenerateKe
y();

      // For additional secur
ity pin the key.
      GCHandle gch = GCHandle
.Alloc( sSecretKey,GCHandleTy
pe.Pinned );

      // Encrypt the file.
```

```
EncryptFile(@"C:\MyData.txt",
         @"C:\Encrypted.txt",
         sSecretKey);

      // Decrypt the file.

DecryptFile(@"C:\Encrypted.tx
t",
         @"C:\Decrypted.txt",
         sSecretKey);

      // Remove the key from
memory.

ZeroMemory(gch.AddrOfPinnedOb
ject(), sSecretKey.Length * 2
);
      gch.Free();
}
```

3. Save the file. Run your application. Make sure that the path that is used for the input file name points to an existing file.

Test the procedure

Test this code with a text (.txt) file to confirm that the code encrypted and decrypted the file correctly. Make sure that you decrypt the file to a new file (as in the **Main()** procedure in this article) instead of to the original file. Examine the decrypted file, and then compare it to the original file.

Complete code listing

```
using System;
using System.IO;
using System.Security;
using System.Security.Cryptograph
y;
using System.Runtime.InteropServi
ces;
using System.Text;

namespace CSEncryptDecrypt
```

```
{
   class Class1
   {
      //  Call this function to r
emove the key from memory after u
se for security

[System.Runtime.InteropServices.D
llImport("KERNEL32.DLL", EntryPoi
nt="RtlZeroMemory")]
      public static extern bool Z
eroMemory(IntPtr Destination, int
Length);

      // Function to Generate a 6
4 bits Key.
      static string GenerateKey()
      {
         // Create an instance of
Symetric Algorithm. Key and IV is
generated automatically.
         DESCryptoServiceProvider
desCrypto =(DESCryptoServicePrivi
der)DESCryptoServiceProvider.Crea
te();

         // Use the Automatically
generated key for Encryption.
         return ASCIIEncoding.ASC
II.GetString(desCrypto.Key);
      }

      static void EncryptFile(str
ing sInputFilename,
         string sOutputFilename,
         string sKey)
      {
         FileStream fsInput = new
FileStream(sInputFilename,
            FileMode.Open,
            FileAccess.Read);

         FileStream fsEncrypted =
new FileStream(sOutputFilename,
            FileMode.Create,
            FileAccess.Write);
         DESCryptoServiceProvider
DES = new DESCryptoServiceProvide
r();
         DES.Key = ASCIIEncoding.
ASCII.GetBytes(sKey);
```

```
         DES.IV = ASCIIEncoding.A
SCII.GetBytes(sKey);
         ICryptoTransform desencr
ypt = DES.CreateEncryptor();
         CryptoStream cryptostrea
m = new CryptoStream(fsEncrypted,
            desencrypt,

CryptoStreamMode.Write);

         byte[] bytearrayinput =
new byte[fsInput.Length];

fsInput.Read(bytearrayinput, 0, b
ytearrayinput.Length);

cryptostream.Write(bytearrayinput
, 0, bytearrayinput.Length);
         cryptostream.Close();
         fsInput.Close();
         fsEncrypted.Close();
      }

      static void DecryptFile(str
ing sInputFilename,
         string sOutputFilename,
         string sKey)
      {
         DESCryptoServiceProvider
DES = new DESCryptoServiceProvide
r();
         //A 64 bit key and IV is
required for this provider.
         //Set secret key For DES
algorithm.
         DES.Key = ASCIIEncoding.
ASCII.GetBytes(sKey);
         //Set initialization vec
tor.
         DES.IV = ASCIIEncoding.A
SCII.GetBytes(sKey);

         //Create a file stream t
o read the encrypted file back.
         FileStream fsread = new
FileStream(sInputFilename,
            FileMode.Open,
            FileAccess.Read);
         //Create a DES decryptor
from the DES instance.
         ICryptoTransform desdecr
```

```
ypt = DES.CreateDecryptor();
         //Create crypto stream s
et to read and do a
         //DES decryption transfo
rm on incoming bytes.
         CryptoStream cryptostrea
mDecr = new CryptoStream(fsread,
            desdecrypt,

CryptoStreamMode.Read);
         //Print the contents of
the decrypted file.
         StreamWriter fsDecrypted
= new StreamWriter(sOutputFilenam
e);
         fsDecrypted.Write(new St
reamReader(cryptostreamDecr).Read
ToEnd());
         fsDecrypted.Flush();
         fsDecrypted.Close();
      }

      static void Main()
      {
         // Must be 64 bits, 8 by
tes.
         // Distribute this key t
o the user who will decrypt this
file.
         string sSecretKey;

         // Get the Key for the f
ile to Encrypt.
         sSecretKey = GenerateKey
();

         // For additional securi
ty Pin the key.
         GCHandle gch = GCHandle.
Alloc( sSecretKey,GCHandleType.Pi
nned );

         // Encrypt the file.

EncryptFile(@"C:\MyData.txt",
            @"C:\Encrypted.txt",
            sSecretKey);

         // Decrypt the file.

DecryptFile(@"C:\Encrypted.txt",
```

```
                @"C:\Decrypted.txt",
                sSecretKey);

            // Remove the Key from m
emory.

ZeroMemory(gch.AddrOfPinnedObject
(), sSecretKey.Length * 2);
            gch.Free();
        }
    }
}
```

References

For more information about cryptography, and about using the cryptographic features of .NET, see the following MSDN Web site:

System.Security.Cryptography namespace

Properties

Article ID: 307010 - Last Review: Jun 26, 2017 - Revision: 2

Applies to
Microsoft Visual C# 2005, Microsoft Visual C# .NET 2003 Standard Edition, Microsoft Visual C# .NET 2002 Standard Edition

Account support
Safety & Security Center
Report a support scam
Supported products list
Download Security Essentials
Contact Microsoft Support
Microsoft Lifecycle Policy
Malicious Software Removal Tool
Locate Microsoft addresses worldwide