Encrypting data

Article • 09/15/2021 • 6 minutes to read • 17 contributors

In this article

Symmetric encryption and asymmetric encryption are performed using different processes. Symmetric encryption is performed on streams and is therefore useful to encrypt large amounts of data. Asymmetric encryption is performed on a small number of bytes and is therefore useful only for small amounts of data.

Symmetric encryption

The managed symmetric cryptography classes are used with a special stream class called a CryptoStream that encrypts data read into the stream. The **CryptoStream** class is initialized with a managed stream class, a class that implements the ICryptoTransform interface (created from a class that implements a cryptographic algorithm), and a CryptoStreamMode enumeration that describes the type of access permitted to the **CryptoStream**. The **CryptoStream** class can be initialized using any class that derives from the Stream class, including FileStream, MemoryStream, and NetworkStream. Using these classes, you can perform symmetric encryption on a variety of stream objects.

The following example illustrates how to create a new instance of the default implementation class for the Aes algorithm. The instance is used to perform encryption on a **CryptoStream** class. In this example, the **CryptoStream** is initialized with a stream object called `fileStream` that can be any type of managed stream. The **CreateEncryptor** method from the **Aes** class is passed the key and IV that are used for encryption. In this case, the default key and IV generated from `aes` are used.

C# Copy

```
Aes aes = Aes.Create();
CryptoStream cryptStream = new CryptoStream(
    fileStream, aes.CreateEncryptor(key, iv), CryptoStreamMode.Write);
```

After this code is executed, any data written to the **CryptoStream** object is encrypted using the AES algorithm.

The following example shows the entire process of creating a stream, encrypting the stream, writing to the stream, and closing the stream. This example creates a file stream that is encrypted using the **CryptoStream** class and the **Aes** class. Generated IV is written to beginning of FileStream, so it can be read and used for decryption. Then a message is written to the encrypted stream with the StreamWriter class. While the same key can be used multiple times to encrypt and decrypt data, it is recommended to generate a new random IV each time. This way the encrypted data is always different, even when plain text is the same.

C# Copy

```
using System;
using System.IO;
using System.Security.Cryptography;

try
{
    using (FileStream fileStream = new("TestData.txt", FileMode.OpenOrCreate))
    {
        using (Aes aes = Aes.Create())
        {
            byte[] key =
            {
                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16
            };
            aes.Key = key;

            byte[] iv = aes.IV;
            fileStream.Write(iv, 0, iv.Length);

            using (CryptoStream cryptoStream = new(
                fileStream,
                aes.CreateEncryptor(),
                CryptoStreamMode.Write))
            {
                using (StreamWriter encryptWriter = new(cryptoStream))
                {
                    encryptWriter.WriteLine("Hello World!");
                }
            }
        }
```

```
    }

    Console.WriteLine("The file was encrypted.");
}
catch (Exception ex)
{
    Console.WriteLine($"The encryption failed. {ex}");
}
```

The code encrypts the stream using the AES symmetric algorithm, and writes IV and then encrypted "Hello World!" to the stream. If the code is successful, it creates an encrypted file named *TestData.txt* and displays the following text to the console:

Console Copy

```
The text was encrypted.
```

You can decrypt the file by using the symmetric decryption example in Decrypting Data. That example and this example specify the same key.

However, if an exception is raised, the code displays the following text to the console:

Console Copy

```
The encryption failed.
```

Asymmetric encryption

Asymmetric algorithms are usually used to encrypt small amounts of data such as the encryption of a symmetric key and IV. Typically, an individual performing asymmetric encryption uses the public key generated by another party. The RSA class is provided by .NET for this purpose.

The following example uses public key information to encrypt a symmetric key and IV. Two byte arrays are initialized that represent the public key of a third party. An RSAParameters object is initialized to these values. Next, the **RSAParameters** object (along with the public key it represents) is imported into an **RSA** instance using the RSA.ImportParameters method. Finally, the private key and IV created by an Aes class are encrypted. This example requires systems to have 128-bit encryption installed.

C# Copy

```
using System;
using System.Security.Cryptography;

class Class1
{
    static void Main()
    {
        //Initialize the byte arrays to the public key information.
        byte[] modulus =
        {
            214,46,220,83,160,73,40,39,201,155,19,202,3,11,191,178,56,
            74,90,36,248,103,18,144,170,163,145,87,54,61,34,220,222,
            207,137,149,173,14,92,120,206,222,158,28,40,24,30,16,175,
            108,128,35,230,118,40,121,113,125,216,130,11,24,90,48,194,
            240,105,44,76,34,57,249,228,125,80,38,9,136,29,117,207,139,
            168,181,85,137,126,10,126,242,120,247,121,8,100,12,201,171,
            38,226,193,180,190,117,177,87,143,242,213,11,44,180,113,93,
            106,99,179,68,175,211,164,116,64,148,226,254,172,147
        };

        byte[] exponent = { 1, 0, 1 };

        //Create values to store encrypted symmetric keys.
        byte[] encryptedSymmetricKey;
        byte[] encryptedSymmetricIV;

        //Create a new instance of the RSA class.
        RSA rsa = RSA.Create();

        //Create a new instance of the RSAParameters structure.
        RSAParameters rsaKeyInfo = new RSAParameters();

        //Set rsaKeyInfo to the public key values.
        rsaKeyInfo.Modulus = modulus;
        rsaKeyInfo.Exponent = exponent;

        //Import key parameters into rsa.
        rsa.ImportParameters(rsaKeyInfo);

        //Create a new instance of the default Aes implementation class.
        Aes aes = Aes.Create();
```

```
        //Encrypt the symmetric key and IV.
        encryptedSymmetricKey = rsa.Encrypt(aes.Key, RSAEncryptionPadding.Pkcs1);
        encryptedSymmetricIV = rsa.Encrypt(aes.IV, RSAEncryptionPadding.Pkcs1);
    }
}
```

See also

- Generating keys for encryption and decryption
- Decrypting data
- Cryptographic services
- Cross-platform cryptography
- Timing vulnerabilities with CBC-mode symmetric decryption using padding
- ASP.NET Core data protection

Recommended content

Decrypting data

Learn how to decrypt data in .NET, using a symmetric algorithm or an asymmetric algorithm.

Generating Keys for Encryption and Decryption

Understand how to create and manage symmetric and asymmetric keys for encryption and decryption in .NET.

Aes Class (System.Security.Cryptography)

Represents the abstract base class from which all implementations of the Advanced Encryption Standard (AES) must inherit.

RijndaelManaged Class (System.Security.Cryptography)

Accesses the managed version of the Rijndael algorithm. This class cannot be inherited.

Walkthrough: Create a Cryptographic Application

Walk through the creation of a cryptographic application. Learn how to encrypt and decrypt content in a Windows Forms application.

Rfc2898DeriveBytes Class (System.Security.Cryptography)

Implements password-based key derivation functionality, PBKDF2, by using a pseudo-random number generator based on HMACSHA1.

Rijndael Class (System.Security.Cryptography)

Represents the base class from which all implementations of the Rijndael symmetric encryption algorithm must inherit.

DES.Create Method (System.Security.Cryptography)

Creates an instance of a cryptographic object to perform the Data Encryption Standard (DES) algorithm.

Show more