Decrypting data

Article • 09/15/2021 • 4 minutes to read • 15 contributors

In this article

Decryption is the reverse operation of encryption. For secret-key encryption, you must know both the key and IV that were used to encrypt the data. For public-key encryption, you must know either the public key (if the data was encrypted using the private key) or the private key (if the data was encrypted using the public key).

Symmetric decryption

The decryption of data encrypted with symmetric algorithms is similar to the process used to encrypt data with symmetric algorithms. The CryptoStream class is used with symmetric cryptography classes provided by .NET to decrypt data read from any managed stream object.

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 decryption on a CryptoStream object. This example first creates a new instance of the Aes implementation class. It reads the initialization vector (IV) value from a managed stream variable, `fileStream`. Next it instantiates a CryptoStream object and initializes it to the value of the `fileStream` instance. The SymmetricAlgorithm.CreateDecryptor method from the Aes instance is passed the IV value and the same key that was used for encryption.

C# Copy

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

The following example shows the entire process of creating a stream, decrypting the stream, reading from the stream, and closing the streams. A file stream object is created that reads a file named *TestData.txt*. The file stream is then decrypted using the **CryptoStream** class and the **Aes** class. This example specifies key value that is used in the symmetric encryption example for Encrypting Data. It does not show the code needed to encrypt and transfer these values.

C# Copy

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

try
{
    using (FileStream fileStream = new("TestData.txt", FileMode.Open))
    {
        using (Aes aes = Aes.Create())
        {
            byte[] iv = new byte[aes.IV.Length];
            int numBytesToRead = aes.IV.Length;
            int numBytesRead = 0;
            while (numBytesToRead > 0)
            {
                int n = fileStream.Read(iv, numBytesRead, numBytesToRead);
                if (n == 0) break;

                numBytesRead += n;
                numBytesToRead -= n;
            }

            byte[] key =
            {
                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16
            };

            using (CryptoStream cryptoStream = new(
               fileStream,
               aes.CreateDecryptor(key, iv),
               CryptoStreamMode.Read))
            {
                using (StreamReader decryptReader = new(cryptoStream))
                {
```

```
                    string decryptedMessage = await decryptReader.ReadToEndAsync();
                    Console.WriteLine($"The decrypted original message: {decryptedMessage}");
                }
            }
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine($"The decryption failed. {ex}");
}
```

The preceding example uses the same key, and algorithm used in the symmetric encryption example for Encrypting Data. It decrypts the *TestData.txt* file that is created by that example and displays the original text on the console.

Asymmetric decryption

Typically, a party (party A) generates both a public and private key and stores the key either in memory or in a cryptographic key container. Party A then sends the public key to another party (party B). Using the public key, party B encrypts data and sends the data back to party A. After receiving the data, party A decrypts it using the private key that corresponds. Decryption will be successful only if party A uses the private key that corresponds to the public key Party B used to encrypt the data.

For information on how to store an asymmetric key in secure cryptographic key container and how to later retrieve the asymmetric key, see How to: Store Asymmetric Keys in a Key Container.

The following example illustrates the decryption of two arrays of bytes that represent a symmetric key and IV. For information on how to extract the asymmetric public key from the RSA object in a format that you can easily send to a third party, see Encrypting Data.

C# Copy

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

// Export the public key information and send it to a third party.
// Wait for the third party to encrypt some data and send it back.

//Decrypt the symmetric key and IV.
```

```
symmetricKey = rsa.Decrypt(encryptedSymmetricKey, RSAEncryptionPadding.Pkcs1);
symmetricIV = rsa.Decrypt(encryptedSymmetricIV , RSAEncryptionPadding.Pkcs1);
```

See also

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

Recommended content

Generating Keys for Encryption and Decryption

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

Encrypting data

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

Aes Class (System.Security.Cryptography)

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

How to: store asymmetric keys in a key container

Learn how to store asymmetric keys in a key container in .NET. See how to create an asymmetric key, save it in a key container, and retrieve and delete the key.

AesManaged Class (System.Security.Cryptography)

Provides a managed implementation of the Advanced Encryption Standard (AES) symmetric algorithm.

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.

RSACryptoServiceProvider Class (System.Security.Cryptography)

Performs asymmetric encryption and decryption using the implementation of the RSA algorithm provided by the cryptographic service provider (CSP). This class cannot be inherited.

Aes.Create Method (System.Security.Cryptography)

Creates a cryptographic object that is used to perform the symmetric algorithm.

Show more