.NET cryptography model

02/26/2021 • 4 minutes to read • 

In this article

.NET provides implementations of many standard cryptographic algorithms, and the .NET cryptography model is extensible.

Object inheritance

The .NET cryptography system implements an extensible pattern of derived class inheritance. The hierarchy is as follows:

- Algorithm type class, such as SymmetricAlgorithm, AsymmetricAlgorithm, or HashAlgorithm. This level is abstract.
- Algorithm class that inherits from an algorithm type class; for example, Aes, RSA, or ECDiffieHellman. This level is abstract.
- Implementation of an algorithm class that inherits from an algorithm class; for example, AesManaged, RC2CryptoServiceProvider, or ECDiffieHellmanCng. This level is fully implemented.

This pattern of derived classes lets you add a new algorithm or a new implementation of an existing algorithm. For example, to create a new public-key algorithm, you would inherit from the AsymmetricAlgorithm class. To create a new implementation of a specific algorithm, you would create a non-abstract derived class of that algorithm.

How algorithms are implemented in .NET

As an example of the different implementations available for an algorithm, consider symmetric algorithms. The base for all symmetric algorithms is SymmetricAlgorithm, which is inherited by Aes, TripleDES, and others that are no longer recommended.

Aes is inherited by AesCryptoServiceProvider, AesCng, and AesManaged.

In .NET Framework on Windows:

- `*CryptoServiceProvider` algorithm classes, such as AesCryptoServiceProvider, are wrappers around the Windows Cryptography API (CAPI) implementation of an algorithm.
- `*Cng` algorithm classes, such as ECDiffieHellmanCng, are wrappers around the Windows Cryptography Next Generation (CNG) implementation.
- `*Managed` classes, such as AesManaged, are written entirely in managed code. `*Managed` implementations are not certified by the Federal Information Processing Standards (FIPS), and may be slower than the `*CryptoServiceProvider` and `*Cng` wrapper classes.

In .NET Core and .NET 5 and later versions, all implementation classes (`*CryptoServiceProvider`, `*Managed`, and `*Cng`) are wrappers for the operating system (OS) algorithms. If the OS algorithms are FIPS-certified, then .NET uses FIPS-certified algorithms. For more information, see Cross-Platform Cryptography.

In most cases, you don't need to directly reference an algorithm implementation class, such as `AesCryptoServiceProvider`. The methods and properties you typically need are on the base algorithm class, such as `Aes`. Create an instance of a default implementation class by using a factory method on the base algorithm class, and refer to the base algorithm class. For example, see the highlighted line of code in the following example:

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}");
}
```

Cryptographic configuration

Cryptographic configuration lets you resolve a specific implementation of an algorithm to an algorithm name, allowing extensibility of the .NET cryptography classes. You can add your own hardware or software implementation of an algorithm and map the implementation to the algorithm name of your choice. If an algorithm is not specified in the configuration file, the default settings are used.

Choose an algorithm

You can select an algorithm for different reasons: for example, for data integrity, for data privacy, or to generate a key. Symmetric and hash algorithms are intended for protecting data for either integrity reasons (protect from change) or privacy reasons

(protect from viewing). Hash algorithms are used primarily for data integrity.

Here is a list of recommended algorithms by application:

- Data privacy:
  - Aes
- Data integrity:
  - HMACSHA256
  - HMACSHA512
- Digital signature:
  - ECDsa
  - RSA
- Key exchange:
  - ECDiffieHellman
  - RSA
- Random number generation:
  - RandomNumberGenerator.Create
- Generating a key from a password:
  - Rfc2898DeriveBytes

See also

- Cryptographic Services
- Cross-Platform Cryptography
- ASP.NET Core Data Protection

Is this page helpful?

Yes No

Recommended content

Cryptographic Services

An overview of the encryption methods and practices supported by .NET.

Cryptographic Signatures

Learn more about: Cryptographic Signatures

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.

Generating Keys for Encryption and Decryption

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

Show more