SECURITY

Protect Private Data with the Cryptography Namespaces of the .NET Framework

Dan Fox

This article assumes you're familiar with C# and the Crypto API

Level of Difficulty 1 2 3

SUMMARY The .NET Framework includes a set of cryptographic services that extend the services provided by Windows through the Crypto API. In this article, the author explores the System.Security.Cryptography namespace and the programming model used to apply cryptographic transformations. He discusses reasons why cryptography is easier in .NET than it was before, including the easy programmatic acccess developers have to the cryptography APIs and the difference between symmetric and asymmetric algorithms. Along the way, a brief discussion of the most widely used algorithms, including RSA, DSA, Rijndael, SHA, and other hash algorithms, is provided.

he explosive growth of e-commerce underscores the need for secure algorithms to safeguard the privacy of data. Microsoft® first addressed data privacy in 1996 with the introduction of the Cryptography API (Crypto API). Today, the continued evolution of security is driving the development of cryptographic services like the System.Security.Cryptography namespace of the Microsoft .NET common language runtime (CLR).

This namespace allows programmatic access to a variety of cryptographic services that you can incorporate into your applications to encrypt and decrypt data, ensure data integrity, and handle digital signatures and certificates. Some of these services are used elsewhere in the Framework (such as ASP.NET authentication). I'll walk through the basics of the cryptographic services provided in the .NET Framework and show a couple of code examples illustrating how you can use these classes in your applications.

The Cryptography Namespace

At the highest level, the Cryptography namespace can be broken into four primary divisions (see **Figure 1**). The primary function of the Cryptography namespace is to provide classes that implement algorithms for such things as encryption and creating hashes. These algorithms are implemented using an extensible pattern that consists of two levels of inheritance. An abstract base class such as AsymmetricAlgorithm or HashAlgorithm sits at the top of the hierarchy and denotes the type of algorithm. A second-level abstract class is then derived from the top-level class to provide the public interface for the algorithm. For example, the SHA1 (Secure Hash Algorithm) class is derived from HashAlgorithm and contains methods and properties particular to the SHA1 algorithm. Finally, the implementation of the algorithm is derived from the second level and is the one typically instantiated and used by client applications. At this concrete level, the implementations can be either managed, unmanaged, or both.

Unmanaged implementations typically contain the suffix "CryptoServiceProvider," for example SHA1CryptoServiceProvider, to indicate that the implementation is actually provided by the Cryptographic Service Provider (CSP) that is installed at the operating system level and acts as a wrapper around the Crypto API. Managed implementations will contain the suffix "Managed," as in SHA1Managed, do not rely on the Crypto API, and are therefore implemented purely in managed code.

The other point to note about these algorithms is that when one of the concrete classes is instantiated, the default constructors always populate the algorithm's default parameters with sensible and secure values (if possible). For example, the asymmetric algorithms that rely on public key cryptography generate a random key pair, while the symmetric algorithms generate both a random key and an initialization vector (IV) and automatically set properties such as Mode and Padding. Furthermore, when possible the algorithms will use "strong" defaults.

The second major collection of classes in the System.Security.Cryptography namespace includes the classes that are actually used during the process of encrypting and decrypting data, as well as various helper classes. The namespace contains classes such as the abstract RandomNumberGenerator class from which RNGCryptoServiceProvider is derived and the ToBase64Transform and FromBase64Transform classes used to transform data to and from base 64.

In addition to exposing cryptographic algorithms, the Cryptography namespace also contains the child namespace X509Certificates. This contains just three classes used to represent and manage Authenticode X.509 v.3 certificates. The X509Certificate class exposes the static methods CreateFromCertFile and CreateFromSignedFile to create an instance of the certificate:

```
Dim c As X509Certificate



c = X509Certificate.CreateFromCertFile("myCert.cer")



Console.WriteLine(c.GetName)

Console.WriteLine(c.GetPublicKeyString)

Console.WriteLine(c.GetSerialNumberString)

Console.WriteLine(c.GetExpirationDateString)
```

The certificate can then be used for several purposes, including validation against a Web server by attaching it to a client request via the ClientCertificates property (which exposes an X509CertificateCollection object) of the System.Net.HttpWebRequest object.

The Cryptography namespace also contains a child XML namespace that is used by the .NET Framework security system to digitally sign XML objects. The framework adheres to a W3C draft specification (http://www.w3.org/TR/2000/WD-xmldsig-core-20000228/) on XML signature syntax and processing. The specification includes XML syntax and processing rules for creating and representing digital signatures that can be applied to any content internal or external to the XML document that contains the signature. While the specification does not address encryption of XML (yet), it is certainly important to provide integrity, message authentication,

and signer authentication services for XML documents.

For the remainder of this article, I will concentrate on the encryption algorithms and the programming model for applying cryptographic transformations.

Encryption Algorithms

As mentioned in **Figure 1**, one type of encryption algorithm found in the Cryptography namespace is a symmetric cryptographic algorithm. Symmetric algorithms are so named because they have a single secret key used for both encryption and decryption. Obviously, both the sender and receiver must keep the key secret for the encryption to remain secure. In addition, when in CBC mode, symmetric algorithms require an IV, a non-secret binary value used to initialize the algorithm and introduce additional cryptographic variance. The SymmetricAlgorithm class is an abstract base class from which other algorithm-specific classes are derived.

The supported symmetric algorithms include Data Encryption Standard (DES), RC2, Rijndael, and Triple Data Encryption Standard (TripleDES). Each algorithm contains an abstract base class derived from SymmetricAlgorithm, such as DES. They also contain a service provider class or managed class that has been derived from the base class—DESCryptoServiceProvider, for example. This class contains the methods used to encrypt and decrypt data. The hierarchy of symmetric algorithms can be seen in **Figure 2**.

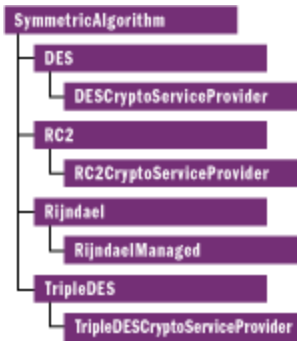

Figure 2 Symmetric Algorithms

The concrete classes such as RijndaelManaged, shown in **Figure 2**, can then be instantiated and their properties called (see **Figure 3**). In **Figure 3**, a new instance of the algorithm is created that automatically generates both a key value and an IV as Byte arrays, which are then serialized to String values and printed to the console. Note that properties such as KeySize and BlockSize can be used to determine the length of the key, and how much data (in bits) can be encrypted or decrypted in one operation.

The second type is known as a public-key or asymmetric algorithm; it derives from the abstract class AsymmetricAlgorithm. They include well-known algorithms such as Digital Signature Algorithm (DSA) and RSA (named after its creators Ron Rivest, Adi Shamir, and Len Adleman). Asymmetric algorithms rely on a pair of keys, one private and the other public. Typically, the public key is made available to anyone and is used by a sender to

encrypt data, whereas the private key remains secure and is used to decrypt data encrypted with the public key. RSA is often used in this way to secure data.

These algorithms are ultimately derived from the abstract classes DSA and RSA, which are themselves derived from AsymmetricAlgorithm, as shown in **Figure 4**. Furthermore, the complex nature of the algorithms means that additional helper classes, such as those derived from AsymmetricKeyExchangeFormatter and AsymmetricSignatureFormatter, are needed.

Figure 4 Asymmetric Algorithms

In addition to allowing the default constructor of asymmetric algorithms to generate a key pair, you can retrieve an existing key pair from storage maintained by the CSP. To do so, you instantiate asymmetric algorithms with the key container name of an existing pair in Crypto API key storage by populating a CspParameters object with the key container name and supplying this instance to the constructor of the asymmetric algorithm. An example of how to store and retrieve asymmetric key pairs can be found at http://www.gotdotnet.com/team/clr/about_security.aspx.

The final type of algorithm exposed by the Cryptography namespace is a hash algorithm. This computes a fixed-size unique sequence of binary values (called a digest) based on a longer sequence of bytes. You use a hash algorithm to determine if data has been tampered with. When you send the digest with the data, the recipient can recompute it. Since the output from a hash algorithm is different if the data input is different, a comparison of the new digest to the old will indicate if the data has changed. Hash algorithms are typically used as the basis for digital signatures.

The Cryptography namespace contains a base class called HashAlgorithm and derived classes that support the algorithms MD5, SHA1, SHA256, SHA384, and SHA512. The MD5 algorithm produces a 128-bit hash, while the SHA1 produces a 160-bit hash. The numbers associated with the other versions of SHA indicate the length of their hash. The larger the hash size, the more secure the algorithm and the more difficult it is to break via brute force. These algorithms come in both unmanaged and managed versions, as shown in **Figure 5**.

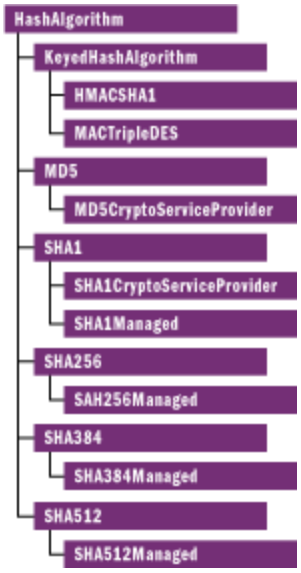

Figure 5 Hash Algorithms

To compute a digest, you simply need to instantiate the hash algorithm and call its overloaded ComputeHash method inherited from HashAlgorithm, like so:

```
  Dim fsData As New FileStream("mydata.txt", _

    FileMode.Open, FileAccess.Read)

  Dim digest() As Byte



Dim oSHA As New SHA512Managed()

digest = oSHA.ComputeHash(fsData)

fsKey.Close()
```

Here the ComputeHash method is passed a Stream object, but it can also accept an array of bytes as an argument.

The Cryptography namespace also contains an abstract class called KeyedHashAlgorithm (see **Figure 5**). Algorithms implemented in the HMACSHA1 and MACTripleDES classes are derived from KeyedHashAlgorithm to generate Message Authentication Codes (MAC). A MAC can be used to determine whether data sent over an insecure channel has been tampered with, as long as both the sender and receiver share a secret key.

Although the abstract base classes SymmetricAlgorithm, AsymmetricAlgorithm, HashAlgorithm, and KeyedHashAlgorithm can't be directly instantiated by a client, they each expose an overloaded static (shared) method called Create. This method can be invoked without parameters to create an instance of the particular algorithm, which by default is RijndaelManaged for symmetric, RSACryptoServiceProvider for asymmetric, SHA1CryptoServiceProvider for hash, and HMACSHA1 for keyed hash. A second version also accepts a string that

maps to the implementation (the list of mappings can be found in the .NET Framework online help). In exactly the same fashion, the abstract class for each algorithm, for example SHA, also contains an overloaded static Create method that instantiates the default implementation for that particular algorithm or accepts the string identifier.

In either case, the object is created by invoking the CreateFromName method of the CryptoConfig class and passing it a string identifier. This list of mappings handles the default values used when the Create method is called. In addition, it handles the mapping of strings passed to the Create method to a particular class. For example, when called without a parameter the Create method maps the default string "System.Security.Cryptography.HashAlgorithm" to the SHA1CryptoServiceProvider class. When passed the string "SHA1," the method maps to the same class.

To illustrate these methods, consider the following code where each statement is equivalent to the others and creates an instance of the RijndaelManaged class:

```
Dim r, r1, r2, r3 As RijndaelManaged



r3 = CType(CryptoConfig.CreateFromName("Rijndael"), "_

           RijndaelManaged)

r2 = CType(SymmetricAlgorithm.Create, RijndaelManaged)

r1 = CType(Rijndael.Create, RijndaelManaged)

r = New RijndaelManaged()
```

This may look confusing, but the classes were designed in this way so that the decision about which algorithm to use could be deferred until run time or specified with the configuration system. In fact, the systemwide configuration file (machine.config) can be augmented with a cryptoNameMapping section that can be used to override the default mappings or add to the list of mappings hard coded in Mscorlib.dll. For example, the XML shown in **Figure 6** can be added to the configuration section of the machine.config file to set the default hash implementation to MD5CryptoServiceProvider and create a new mapping from the string "dan" to the implementation.

What about creating your own mappings? You can add a cryptoClass element within the cryptoClasses element to define a name to be referenced, in this case "myMD5," and to fully describe the class and assembly. The nameEntry elements are then used to create the mappings. Then this client code can be written:

```
Dim m, m1 As MD5CryptoServiceProvider



m = CType(HashAlgorithm.Create, MD5CryptoServiceProvider)
```

```
m1 = CType(CryptoConfig.CreateFromName("dan"), "_

           MD5CryptoServiceProvider)
```

In both instances, objects of type MD5CryptoServiceProvider are created.

Using Cryptographic Services

A stream-based programming model pervades the .NET Framework. Stream classes derived from System.IO.Stream are used for representing data from stores such as text files, XML documents, MSMQ messages, memory, and the network, and can be used interchangeably to move data into and out of those stores. It should be no surprise that the Cryptography namespace piggybacks on that concept, providing an elegant and efficient way to encrypt and decrypt data using the algorithms discussed previously. The heart of this functionality is the CryptoStream class, which is derived from System.IO.Stream and which serves as a stream-based model for cryptographic transformations.

To illustrate how to use the basic cryptographic functionality in this namespace, the simple TextFileCrypto class is shown in **Figure 7**. This class can be used to symmetrically encrypt and decrypt a file using DES with a key generated by the class and saved to a file.

As you can see, the class constructor instantiates a class-level DESCryptoServiceProvider class that exposes the CreateEncryptor and CreateDecryptor methods. These methods are used later to return objects that perform the encryption or decryption. The public KeyFile and FileName properties are used to open an existing file that contains the key and IV and to set the name of the file to encrypt or decrypt. If the key file exists, the file is opened using the private OpenKeyFile method, and the key and IV are read into Byte arrays. The SaveKeyFile method calls the GenerateKey and GenerateIV methods of the underlying DES class to generate a random key and IV. These values are then saved to the file passed in as an argument. At this point, the key file should be kept secret and can be placed in the possession of someone who will use it to decrypt the data.

It should be noted that the DES algorithm uses a 56-bit (7-byte) key. Although DES supports only a single key size, the valid key sizes for other algorithms can be found by querying the LegalKeySizes property, which returns an array of KeySize objects that represent the legal sizes with the properties MinSize, MaxSize, and SkipSize (increment). For example, the Rijndael algorithm supports key sizes of 128, 192, and 256 bits. For stronger encryption, a new size can be set using the KeySize property.

The EncryptFile and DecryptFile methods are the core of the TextFileCrypto class. The first thing the EncryptFile method does is open the file to encrypt and a temporary file to store the encrypted data. It then creates an encryptor object using the CreateEncryptor method of the service provider (in this case DESCryptoServiceProvider). This object implements the ICryptoTransform interface and therefore can be passed to the constructor of CryptoStream that transforms the data as it is written out to a file backed by a FileStream (fsOutput in this case). The original file is then overwritten with the temporary file. The result is an encrypted file that can be safely sent over FTP or as an e-mail attachment via the Internet.

The DecryptFile method does the reverse by creating a decryptor using the CreateDecryptor method. This object is likewise passed to the constructor of the CryptoStream to decrypt the file as it is read from the FileStream fsRead. Finally, the decrypted data is saved to a text file by the StreamWriter class using the ReadToEnd method of the StreamReader class.

A client can use the TextFileCrypto class to encrypt data in four simple lines of code as follows:

```
Dim objCrypt As New TextFileCrypto()



objCrypt.FileName = "students.txt"



' Generate and save the key

objCrypt.SaveKeyFile("mykey.dat")



' Encrypt the file

objCrypt.EncryptFile()
```

When a client receives the file, he can decrypt it just as easily:

```
Dim objCrypt As New TextFileCrypto()



objCrypt.FileName = "students.txt"



' Read in the key

objCrypt.KeyFile = "mykey.dat"



' Decrypt the file

objCrypt.DecryptFile()
```

Strong Encryption in Windows

While an old legal restriction limiting the export of strong-encrypted products has been lifted, the .NET Framework will not use strong encryption for unmanaged implementations if your operating system is an export version. To circumvent this, you can install the High Encryption Pack. Service Pack 2 for Windows 2000 includes

the High Encryption Pack, which you can get from Windows 2000 High Encryption Pack.

For Windows NT® 4.0, Service Packs are distributed in both standard and high encryption versions. If you do not have one already installed, download the high encryption version of Service Pack 6a.

For Windows ME, Windows 98 and Windows 95, Microsoft Internet Explorer 5.5 includes the High Encryption Pack. If you are running a version of Internet Explorer older than 5.5, you can obtain the corresponding High Encryption Pack.

For related articles see:
A Simple Guide to Cryptography
For background information see:
http://www.gotdotnet.com/team/clr/about_security.aspx
http://www.gotdotnet.com/team/clr/cryptofaq.htm

Dan Fox is a Technology Director for Quilogy in Overland Park, Kansas. He is the author of *Pure Visual Basic* (Sams, 1999) and *Building Distributed Application with Visual Basic .NET* (Sams, 2002), and coauthor of *Exam Cram*: *Visual Basic 6* (The Coriolis Group, 1999). You can reach Dan at dfox@quilogy.com.

From the June 2002 issue of MSDN Magazine.
Get it at your local newsstand, or better yet, *subscribe*.

Figure 1 Components of Cryptography

Division	Description
Encryption Algorithms	A set of classes used to implement both symmetric and asymmetric encryption and hash algorithms
Helper Classes	Classes used to generate random numbers, perform conversions, interact with the Crypto API store, and actually perform the encryption using a stream-based model
X.509 Certificates	Classes defined in the System.Security. Cryptography.X509Certificates namespace used to represent digital certificates
XML Digital Signatures	Classes defined in the System. Cryptography.Xml namespace used to represent digital signatures in XML documents

---

Figure 3 Using RijndaelManaged

```
Dim oEnc As New RijndaelManaged()

Dim i As Short

Dim strKey, strIV As String



For i = 1 To (oEnc.KeySize / 8)
```

```
  strKey &= oEnc.Key(i - 1).ToString & " "
Next
For i = 1 To 16
  strIV &= oEnc.IV(i - 1).ToString & " "
Next


Console.WriteLine(strKey)
Console.WriteLine(strIV)
Console.WriteLine(oEnc.KeySize.ToString)
Console.WriteLine(oEnc.BlockSize.ToString)
```

Figure 6 Configuring Cryptography

```
<mscorlib>
 <cryptographySettings>
   <cryptoNameMapping>
     <cryptoClasses>
       <cryptoClass myMD5=
        "System.Security.Cryptography.MD5CryptoServiceProvider,
        mscorlib, Ver=1.0.2411.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089"/>
       </cryptoClasses>
       <nameEntry name="dan" class="myMD5"/>
       <nameEntry name="System.Security.Cryptography.HashAlgorithm"
        class="myMD5"/>
     </cryptoNameMapping>
  </cryptographySettings>
</mscorlib>
```

Figure 7 Basic Cryptography Using the DES Algorithm

```
Option Strict On



Imports System.Security.Cryptography

Imports System.IO

Imports System.Threading



Public Class TextFileCrypto

  ' Encrypts and decrypts text file given a key



  Private mstrFile As String ' File to decrypt

  Private mstrKey As String ' Key file

  Private mKey(7) As Byte  ' DES key

  Private mDES As DESCryptoServiceProvider

  Private mIV(7) As Byte ' Initialization Vector



  Public Sub New()

    mDES = New DESCryptoServiceProvider()

  End Sub



  Public Property KeyFile() As String

    Get

      Return mstrKey

    End Get

    Set(ByVal Value As String)

      If File.Exists(Value) Then
```

```
      mstrKey = Value
      OpenKeyFile() ' Open the key file and read its contents
    Else
      Throw New FileNotFoundException(Value & " file does not exist.")
    End If
  End Set
End Property

Public Property FileName() As String
  Get
    Return mstrFile
  End Get
  Set(ByVal Value As String)
    If File.Exists(Value) Then
      mstrFile = Value
    Else
      Throw New FileNotFoundException(Value & " does not exist.")
    End If
  End Set
End Property

Private Sub OpenKeyFile()
  Dim fsKey As New FileStream(mstrKey, _
    FileMode.Open, FileAccess.Read)

  ' Open the key file and read the key from it
```

```
  fsKey.Read(mKey, 0, 8)
  fsKey.Read(mIV, 0, 8)
  fsKey.Close()

  mDES.Key = mKey
  mDES.IV = mIV
End Sub

Public Function SaveKeyFile(ByVal FilePath As String) As Boolean
  Dim fsKey As New FileStream(FilePath, _
    FileMode.OpenOrCreate, FileAccess.Write)

  ' Generate a new random key and IV and save it to the file
  ' Note these will be generated randomly automatically
  ' if you don't do it
  mDES.GenerateKey()
  mDES.GenerateIV()

  mKey = mDES.Key
  mIV = mDES.IV

  fsKey.Write(mKey, 0, mKey.Length)
  fsKey.Write(mIV, 0, mKey.Length)
  fsKey.Close()

  mstrKey = FilePath
```

```
End Function



Public Function EncryptFile() As Boolean

  ' Encrypt the given file



  ' Check the key

  If mKey Is Nothing Then

    Throw New Exception("You must have a key in place first.")

    Return False

  End If



  Dim fsInput As New FileStream(mstrFile, _

    FileMode.Open, FileAccess.Read)

  Dim fsOutput As New FileStream("temp.dat", _

    FileMode.Create, FileAccess.Write)

  fsOutput.SetLength(0)



  Dim arInput() As Byte



  ' Create DES Encryptor from this instance

  Dim desEncrypt As ICryptoTransform = mDES.CreateEncryptor()

  ' Create Crypto Stream that transforms file stream using DES encryption

  Dim sCrypto As New CryptoStream(fsOutput, desEncrypt, _

    CryptoStreamMode.Write)



  ReDim arInput(Convert.ToInt32(fsInput.Length - 1))
```

```
  fsInput.Read(arInput, 0, Convert.ToInt32(fsInput.Length))
  fsInput.Close()

  ' Write out DES encrypted file
  sCrypto.Write(arInput, 0, arInput.Length)
  sCrypto.Close()

  ' Delete and rename
  File.Copy("temp.dat", mstrFile, True)
  File.Delete("temp.dat")

End Function

Public Function DecryptFile() As Boolean
  ' Decrypt the given file

  ' Check the key
  If mKey Is Nothing Then
    Throw New Exception("You must have a key in place first.")
    Return False
  End If

  ' Create file stream to read encrypted file back
  Dim fsRead As New FileStream(mstrFile, FileMode.Open, _
    FileAccess.Read)
  Dim fsOutput As New FileStream("temp.dat", _
```

```
    FileMode.Create, FileAccess.Write)

  ' Create DES Decryptor from our des instance

  Dim desDecrypt As ICryptoTransform = mDES.CreateDecryptor()

  ' Create crypto stream set to read and do a des decryption

  ' transform on incoming bytes

  Dim sCrypto As New CryptoStream(fsRead, desDecrypt, _

    CryptoStreamMode.Read)

  Dim swWriter As New StreamWriter(fsOutput)

  Dim srReader As New StreamReader(sCrypto)



  ' Write out the decrypted file

  swWriter.Write(srReader.ReadToEnd)



  ' Close and clean up

  swWriter.Close()

  fsOutput.Close()

  fsRead.Close()



  ' Delete and rename

  WaitForExclusiveAccess(mstrFile)

  File.Copy("temp.dat", mstrFile, True)

  File.Delete("temp.dat")

End Function



Private Sub WaitForExclusiveAccess(ByVal fullPath As String)

  While (True)
```

```
      Try
        Dim file As FileStream
        file = New FileStream(fullPath, FileMode.Append, _
          FileAccess.Write, FileShare.None)
        file.Close()
        Exit Sub
      Catch e As Exception
        Thread.Sleep(100)
      End Try
    End While
  End Sub


End Class
```
