Cryptography in .NET

Venkat Subramaniam
venkats@agiledeveloper.com
http://www.agiledeveloper.com/download.aspx

Abstract

In this article we explore the capabilities provided by the Cryptography API in .NET. We will take a look at the facilities provided by the RSACryptoServiceProvider class. We will also look at some of the other algorithms available as part of the API. Our discussion will center on encryption of messages and techniques that could be used to encrypt information/password as well.

The strength of the API

The power of .NET comes from two places: One is the capabilities of Visual Studio .NET towards enhancing the productivity of the developers. The other is the strength of the API that is part of the .NET Framework. During my development efforts, I have come to realize the benefits offered by the framework a number of times. I truly believe that it allows us to write less code to implement what you may call as infrastructural functionalities. Writing less code is good, since it allows us to write more code related to our domain and application.

Why talk about Cryptography?

Recently I have been asked about this by at least a few clients. One request went some what like “We are using MSMQ and we want to encrypt the message that is being transmitted.” My first response was, “MSMQ has facility to encrypt the message.” Upon further discussion we realized that the message is encrypted during transmission. Once it is delivered, however, it is wide open for someone to browse from the message queue. So, the task on hand is to encrypt the message so that only the receiving client application can view it. Another request went some thing like, we have to keep our own application based user id and password in the database. However, the information in the database needs to be encrypted. These are just a few of the requests that lead us into the System.Security.Cryptography[1] API provided in the .NET Framework.

Code is worth a 1000 words

Let’s just delve into the code. First I want to create a pair of keys. I want to send the public key for others to use for encryption. Once they encrypt, using my public key, they may send me the information which I can decrypt using my private key. Let’s look at the code to create the public and private keys:

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
// Add Reference to System.Security assembly
using System.IO;
using System.Text;


namespace RSA
```

```
{
	/// <summary>
	/// Summary description for Crypto.
	/// </summary>
	public class Crypto
	{
		/// <summary>
		/// Creates a public and private key.
		/// The public key will be in a file named
		   keyFileNamePub.xml and private key in
		   keyFileName.xml.
		/// For instance if the fileName is given as key, then the
		   files will be keyPub.xml and key.xml
		/// </summary>
		/// <param name="keyFileName">Only the name of the file. An
		   XML extension will be created for you</param>
		public void createKey(String keyFileName)
		{
			RSAKeyValue theRSAKeyValue =
			                   new RSAKeyValue();
			String str =
			        theRSAKeyValue.Key.ToXmlString(true
			              /* include private
			                          parameters*/);
			TextWriter writer = new StreamWriter(
			                       keyFileName + ".xml");
			writer.Write(str);
			writer.Close();

			str = theRSAKeyValue.Key.ToXmlString(false);
			writer = new StreamWriter(keyFileName + "Pub.xml");
			writer.Write(str);
			writer.Close();
		}
```

The above code is pretty self explanatory. Once a pair of public and private key is created, it can be used for encryption/decryption. The RSAKeyValue's constructor creates a random key. The key property of the RSAKeyValue object is of type RSA. RSA is an abstract base class from which implementations of the RSA algorithm inherit. The RSACryptoServiceProvider, which is used in the example later, is a sealed class that derives from RSA. It performs asymmetric encryption and decryption and is the default implementation of RSA.

Encryption and Decryption

The following methods are part of our Crypto class (above):

```
		/// <summary>
		/// Given the keyFileName (without any extension, .xml
		   extension assumed),
		/// encrypts or decrypts the inputBytes and puts the result
		   in outputBytes.
		/// </summary>
		/// <param name="keyFileName">Name of Key File (without
                assumed xml extension)</param>
        /// <param name="inputBytes">Data to be encrypted or
                decrypted</param>
        /// <param name="outputBytes">The resulting encrypted or
                decrypted data</param>
        /// <param name="encrypt">true to encrypt. false to
                decrypt</param>
        public void encryptOrDecrypt(String keyFileName,
                byte[] inputBytes, out byte[] outputBytes,
                bool encrypt)
        {
                String key = ReadFileToString(keyFileName + ".xml");
                RSACryptoServiceProvider
                        theRSACryptoServiceProvider
                                = new RSACryptoServiceProvider();
                theRSACryptoServiceProvider.FromXmlString(key);

                outputBytes = null;

                if (encrypt)
                        outputBytes =
                        theRSACryptoServiceProvider.Encrypt(inputBytes,
                         false /* Direct Encryption or OAEP Padding*/);
                else
                        outputBytes =
                        theRSACryptoServiceProvider.Decrypt(inputBytes,
                         false);
        }

        /// <summary>
        /// Reads the contents of the given file into a string
        /// </summary>
        /// <param name="keyFileName">File to read</param>
        /// <returns></returns>
        private String ReadFileToString(String fileName)
        {
                String keyString = "";

                byte[] buffer = new byte[new
                                  FileInfo(fileName).Length];
                FileStream strm = File.OpenRead(fileName);
                strm.Read(buffer, 0, buffer.Length);
                strm.Close();
                return new ASCIIEncoding().GetString(buffer);
        }
```

The Encrypt method of RSACryptoServiceProvider encrypts the data in inputBytes using the given key. The output of this method is a stream of encrypted bytes. The Decrypt method similarly decrypts the given set of inputBytes using the given key.

Test code to encrypt/decrypt

Let’s give this a try. Here is a sample code to use the Crypto class we wrote above:

```
        class TestCode
        {
```

```
private static void displayUsage()
{
      Console.WriteLine(
      "Usage: RSAEncryption ( -c keyFileName | " +
            " -e keyFileName inputfile outputfile |
            -d keyFileName inputfile outputfile");
}

private static void processRequest(String[] args)
{
      Crypto cryptoHelper = new Crypto();

      String request = args[0];

      bool okUsage = true;

      switch(request)
      {
            case "-c":
                  if (args.Length == 2)
                        cryptoHelper.createKey(args[1]);
                  else
                        okUsage = false;
                  break;

            case "-e":
                  if (args.Length == 4)
                  {
                        FileInfo fileInfo =
                              new FileInfo(args[2]);
                        byte[] inputBytes =
                              new byte[fileInfo.Length];
                        byte[] outputBytes = null;

                        FileStream strm =
                              File.OpenRead(args[2]);
                        strm.Read(inputBytes, 0,
                               inputBytes.Length);
                        strm.Close();

                        cryptoHelper.encryptOrDecrypt(
                              args[1], inputBytes,
                              out outputBytes, true);
                        strm = File.OpenWrite(args[3]);
                        strm.Write(outputBytes, 0,
                               outputBytes.Length);
                        strm.Close();
                  }
                  else
                        okUsage = false;
                  break;

            case "-d":
                  if (args.Length == 4)
                  {
                        FileInfo fileInfo =
                              new FileInfo(args[2]);
                              byte[] inputBytes =
                                    new byte[fileInfo.Length];
                              byte[] outputBytes = null;

                              FileStream strm =
                                    File.OpenRead(args[2]);
                              strm.Read(inputBytes, 0,
                                     inputBytes.Length);
                              strm.Close();

                              cryptoHelper.encryptOrDecrypt(
                                    args[1], inputBytes,
                                    out outputBytes, false);
                              strm = File.OpenWrite(args[3]);
                              strm.Write(outputBytes, 0,
                                    outputBytes.Length);
                              strm.Close();
                        }
                        else
                              okUsage = false;
                        break;
            }

            if (!okUsage)
            {
                  Console.WriteLine("Invalid input");
                  displayUsage();
            }
      }

      static void Main(string[] args)
      {
            if (args.Length < 2)
                  displayUsage();
            else
                  processRequest(args);
      }
}
```

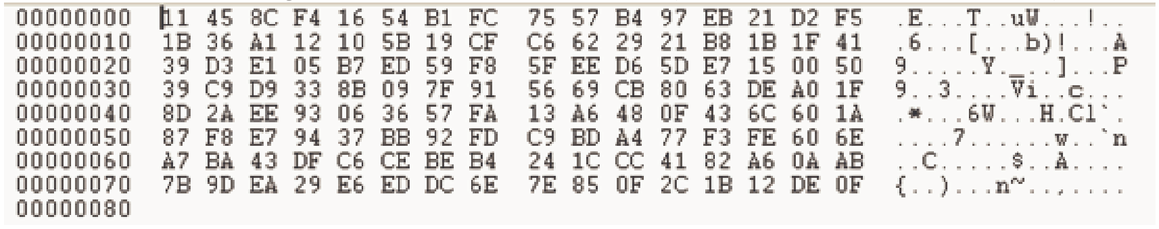
We will create a key file as follows:

RSA.exe -c key

This creates two files key.xml and keyPub.xml. The content of the keyPub.xml is as follows:
<RSAKeyValue><Modulus>oCA6Mh4p2zpXovIThy1vTBBvw0mwbiHVsYyl7omr2wu9mddFGCUS0rPdcOOhQAbL+Cbu9ZVZJ552QtA3EnNtYqbYAam/OKTWvdiYR2rAkfxNDppDc0WvdK2gWyc76cd0pnYLNFxU7MnYDpvl9l/EJABCYnxrstsCZcPw7uWa6PE=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>

Now, we will use this public key to encrypt a file. We have a file named secret.txt with the content “Hello, this is a test.” We will run the following command:

RSA.exe -e keyPub secret.txt msg.txt
The content of the msg.txt file created is as follows (shown within VS.NET editor):



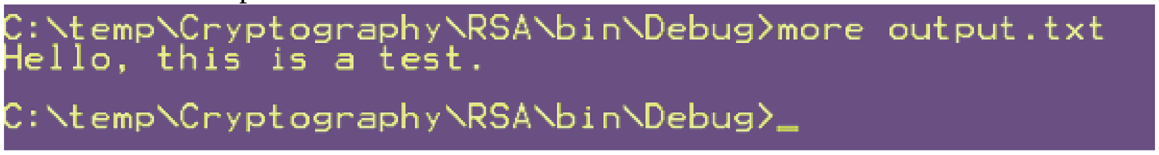
```
00000000  11 45 8C F4 16 54 B1 FC  75 57 B4 97 EB 21 D2 F5  .E...T..uW...!..
00000010  1B 36 A1 12 10 5B 19 CF  C6 62 29 21 B8 1B 1F 41  .6...[...b)!...A
00000020  39 D3 E1 05 B7 ED 59 F8  5F EE D6 5D E7 15 00 50  9.....Y._..]...P
00000030  39 C9 D9 33 8B 09 7F 91  56 69 CB 80 63 DE A0 1F  9..3....Vi..c...
00000040  8D 2A EE 93 06 36 57 FA  13 A6 48 0F 43 6C 60 1A  .*...6W...H.Cl`.
00000050  87 F8 E7 94 37 BB 92 FD  C9 BD A4 77 F3 FE 60 6E  ....7......w..`n
00000060  A7 BA 43 DF C6 CE BE B4  24 1C CC 41 82 A6 0A AB  ..C.....$..A....
00000070  7B 9D EA 29 E6 ED DC 6E  7E 85 0F 2C 1B 12 DE 0F  {..)...n~..,....
00000080
```

Note that we used the public key in keyPub.xml to encrypt.

Now, in order to decrypt, we will issue the following command:

RSA.exe -d key msg.txt output.txt

The content of output.txt is:



```
C:\temp\Cryptography\RSA\bin\Debug>more output.txt
Hello, this is a test.

C:\temp\Cryptography\RSA\bin\Debug>_
```

Note that an attempt to decrypt using keyPub.xml (the public key) will result in an exception.

Asymmetric Algorithm

The RSA[2] (Rivest, Shamir, and Adleman) cryptography is based on using public and private keys which are large prime numbers. As against symmetric keys, which use the same key to encrypt and decrypt, RSA uses asymmetric keys. Asymmetric algorithms, also known as Public-key algorithms, require the sender and receiver to maintain a pair of related keys. It is difficult to factor the private key from the public key. Further, message encrypted using public key can not be decrypted using the public key. One use of asymmetric keys is in encryption. Another use is in digital signatures used to authenticate the sender of information. These algorithms and related API play a significant role in providing confidentiality, integrity and authentication.

Help with passwords

What if an application wants to manage user ids and passwords. Storing passwords as readable strings is of course not advisable. One possibility is to use a class available as part of the cryptography API to manage passwords.

In comparing passwords, it is advisable that you read the user entered password, transform it and compare it with the pre-transformed password. The following sample does just that. We show one method, createPassword, which will transform the user entered password and store it in an xml file. This may be used, for instance, when creating a new user in our application. We then show another method, checkPassword, which takes a user entered password, transforms and verifies if it is equal to the (pre-transformed) password.

MD5 and SHA256

MD5 is a hash function that takes a binary string of arbitrary length and maps it to a binary string of fixed length. It is such that no two different inputs would map to the same hash value. Hashes of two data match if the data also match. The MD5CryptoServiceProvider is a class that provides default implementation of this in the Cryptography API. SHA256 is similar where it uses a hash size of 256 bits. SHA256Managed is the default implementation of this.

Creating and validating Password

The code to create the password and the code to validate the password is shown below:

```
class InvalidPassword : ApplicationException
{
      public InvalidPassword() : base("Invalid Password") {}
}

public class PasswordHelper
{
      /// <summary>
      /// Transforms the given password usign SHA256 hash
            algorithm
      /// </summary>
      /// <param name="password">Password to transform</param>
      /// <returns>Hash value of the given password</returns>
      public string transformPassword(string password)
      {
            HashAlgorithm alg = new SHA256Managed();
            byte[] hashCode =
                  alg.ComputeHash(new
                        System.Text.UnicodeEncoding().GetBytes(
                                                      password));

            return BitConverter.ToString(hashCode);
      }

      /// <summary>
      /// Checks if the password given matches the password hash
                  value
      /// </summary>
      /// <param name="password">Password to match</param>
      /// <param name="passwordHashValue">hash value to match
                  with</param>
      /// <exception cref="InvalidPassword">Throws if the match
                  fails</exception>
      public void checkPassword(string password,
                                    string passwordHashValue)
      {
            HashAlgorithm alg = new SHA256Managed();
            byte[] hashCode =
                  alg.ComputeHash(new
                        System.Text.UnicodeEncoding().GetBytes(
                                                      password));

            password = BitConverter.ToString(hashCode);
```

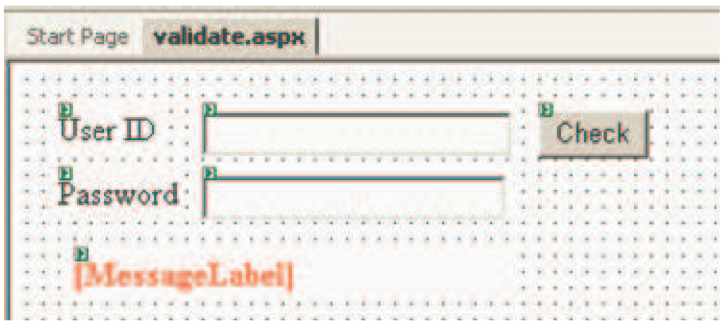
```
                    if (password != passwordHashValue)
                          throw new InvalidPassword();
              }
        }
```

Test code for password validation

Assume this is a web based application. We will maintain user id and password in an xml document. The page named validate.aspx will allow us to validate the password for a given user. We assume that the following XML document has already been created using other means (the password values may be generated using the transformPassword method above).

```
<!-- userinfo.xml -->
<userinfo>
<venkat>06-E4-4D-C1-B9-5C-46-9F-43-AA-CC-B4-9E-93-C3-68-27-62-62-66-EE-
D5-57-5E-CE-D7-4A-F9-A0-16-C9-CD</venkat>
<kim>BE-DD-71-F7-10-BC-29-9A-FF-E8-98-AF-C7-5E-6F-B2-80-B5-B9-C4-99-D6-
6C-FE-DD-51-87-24-58-DC-04-09</kim>
<john>C0-2F-B2-5F-CE-06-15-A9-38-E3-7D-AB-AB-82-64-AD-AD-E7-DF-2E-2C-
61-1D-AE-1C-4C-6E-B8-53-FD-1A-29</john>
</userinfo>
```

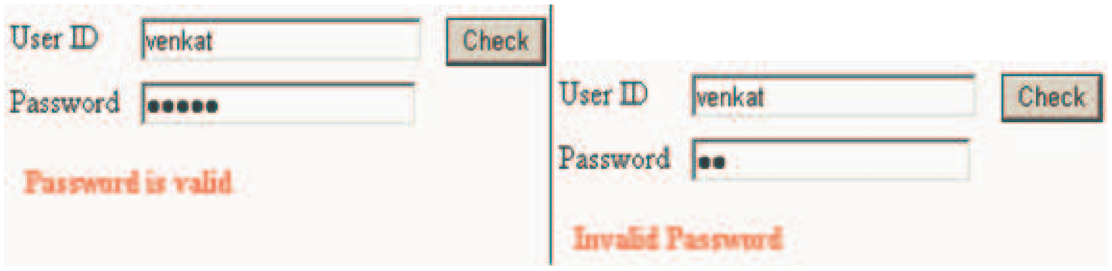
The validate page and the related code is shown below:



```
            private string getPassword(string userID)
            {
                  string result = null;

                  XmlDocument doc = new XmlDocument();
                  doc.Load(Server.MapPath("") + "/userinfo.xml");

                  XmlElement element = (XmlElement)
                        doc.SelectSingleNode("userinfo/" + userID);

                  if (element != null)
                  {
                        result = element.FirstChild.Value;
                  }

                  return result;
            }
```

```
private void CheckButton_Click(object sender,
                   System.EventArgs e)
{
      string userID = UserIDTextBox.Text;
      string givenPassword = PasswordTextBox.Text;

      try
      {
            string storedPassword = getPassword(userID);


            if (storedPassword == null)
                  throw new
                    ApplicationException("UserID Invalid");

            PasswordUtil.PasswordHelper helper =
                        new PasswordUtil.PasswordHelper();

            helper.checkPassword(givenPassword,
                        storedPassword);
      }
      catch(Exception ex)
      {
            MessageLabel.Text = ex.Message;
            return;
      }

      MessageLabel.Text = "Password is valid";
}
```

A sample execution of the program is shown below:



Password is valid

Invalid Password

In the above screen snapshot, the password “hello” is valid while the password “ho” is not. The file userinfo.xml shown previously has the hash value for “hello” as content of the <venkat> element. Note that the application did not transform the hash value to text “hello.” Instead, the hash value of the user entered password is computed and then compared with the hash value stored in the xml document.

Conclusion

In this article we presented the capabilities of some of the classes in the System.Security.Cryptography namespace in .NET. The facilities for RSA algorithm in

.NET makes it easier to implement applications that require encryption or authentication. We also explored the hash algorithm that helps us store information like password.

References

1. http://msdn.microsoft.com.
2. Rivest, et. al., *A method for obtaining digital signatures and public-key cryptosystems,* Communications of the ACM (2) 21 (1978), 120-126.