AES Encryption and Decryption in Microsoft .NET

William J. Buchanan

Centre for Distributed Computing and Security,
Edinburgh Napier University
{w.buchanan}@napier.ac.uk
http://cdcs.napier.ac.uk

Abstract. This paper outlines the usage of AES in Microsoft .NET. It provides a basic overview of the AES method, along with a review of other popular encryption methods and some sample code which can be used to implement AES.

1 Introduction

The future of the Internet, especially in expanding the range of applications, involves a much deeper degree of privacy, and authentication. Without these the Internet cannot be properly used to replace existing applications such as in voting, finance, and so on [3][2][1]. The future is thus towards data encryption which is the science of cryptographics , and provides a mechanism for two entities to communicate without any other entity being able to read their messages. In a secret communications system, Bob and Alice should be able to communicate securely, without Eve finding out the contents of their messages, or in keeping other details secure, such as their location, or the date that their messages are sent (Figure 1).

There are many ways that encryption can be used in modern application, including encrypting data buckets in an e-Health Cloud [6], in digital forensics [7] and in information sharing [9]. New methods have also been created related to new ways to encrypting data using cumulative encryption [8], which supports the usage of the encryption keys being added to the data so that it does not matter the one round that the ciper text is decrypted. For example if Alice encrypts with her key, and the Bob encrypts with his, then Alice or Bob can then apply their keys in any order, so that the data can be decrypted.

The two main methods used are to either use a unique algorithm which both Bob and Alice know, and do not tell Eve, or they use a well-known algorithm, which Eve also knows, and use some special electronic key to uniquely define how the message is converted into cipertext, and back again. A particular problem in any type of encryption is the passing of the secret algorithm or the key in a secure way, as Bob or Alice does not know if Eve is listening to their communications. If Eve finds-out the algorithm or the key, neither Bob nor Alice is able to detect this. This chapter looks at some of the basic principles of encryption, including

the usage of private-key and public-key methods. As we will find public and private key methods work together in perfect harmony, with, typically, private key methods providing in the actual core encryption, and public key methods providing ways to authenticate, and pass keys.

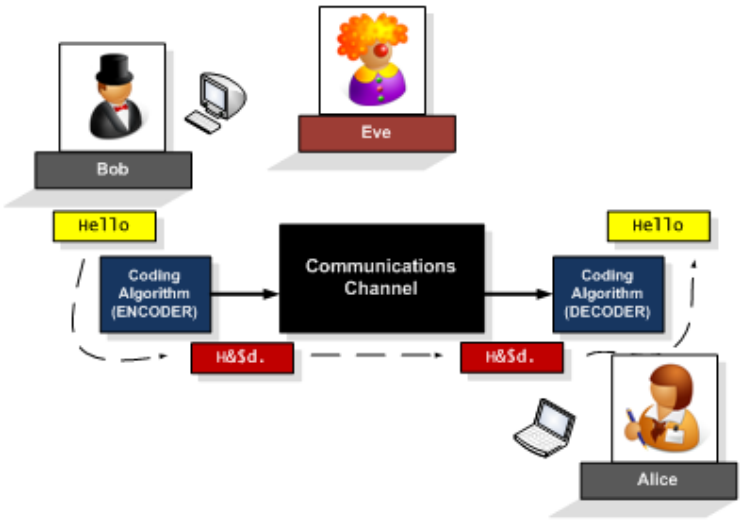

Fig. 1. Figure 1: Private key encryption

2 Private Key

Private-key (or secret-key) encryption techniques use a secret key which is only known by the two communicating parties, as illustrated in Figure 2 [5]. This key can generated by a phase-phase, or can be passed from the two parties over a secure communications link. The most popular private-key techniques include:

– DES (Data Encryption Standard) is a block cipher scheme which operates on 64-bit block sizes. The private key has only 56 useful bits, as eight of its bits are used for parity (which gives 2^{56} or 10^{17} possible keys). DES uses a complex series of permutations and substitutions, the result of these operations is XOR'ed with the input. This is then repeated 16 times using a different order of the key bits each time. DES is a strong code and has never been broken, although several high-powered computers are now available which, using brute force, can crack the code. A possible solution is 3DES (or triple DES) which uses DES three times in a row. First to encrypt, next to decrypt and finally to encrypt. This system allows a key-length of more than 128 bits. The technique uses two keys and three executions of the DES algorithm. A key, K1, is used in the first execution, then K2 is used and

finally K1 is used again. These two keys give an effective key length of 112 bits, that is 2^{64} key bits minus 16 parity bits. The Triple DES process is illustrated in Figure 3.

- RC4. RC4 is a stream cipher designed by RSA Data Security, Inc and was a secret until information on it appeared on the Internet. The secure socket layer (SSL) protocol and wireless communications (IEEE 802.11a/b/g) use RC4. It uses a pseudo random number generator, where the output of the generator is XOR'ed with the plaintext. It is a fast algorithm and can use any key-length. Unfortunately the same key cannot be used twice. Recently a 40-bit key version was broken in eight days without special computer power.
- AES/Rijndael. AES (Advanced Encryption Standard) is a new standard for encryption, and uses 128, 192 or 256 bits. It was selected by NIST in 2001 (after a five year standardisation process). The name Rijndael comes from its Belgium creators: Joan Daemen and Vincent Rijmen. The future of wireless systems (WPA-2) is likely to be based around AES (while WPA uses TKIP which is a session key method which is based around stream encryption using RC4).
- IDEA. IDEA (International Data Encryption Algorithm) is similar to DES. It operates on 64-bit blocks of plaintext, using a 128-bit key, and has over 17 rounds with a complicated mangler function. During decryption this function does not have to be reversed and can simply be applied in the same way as during encryption (this also occurs with DES). IDEA uses a different key expansion for encryption and decryption, but every other part of the process is identical. The same keys are used in DES decryption, but in the reverse order. The key is devised in eight 16-bit blocks; the first six are used in the first round of encryption the last two are used in the second run. It is free for use in non-commercial version and appears to be a strong cipher.
- RC5. RC5 is a fast block cipher designed by Rivest for RSA Data Security. It has a parameterized algorithm with a variable block size (32, 64 or 128 bits), a variable key size (0 to 2048 bits) and a variable number of rounds (0 to 255). It has a heavy use of data dependent rotations, and the mixture of different operations, which assures that RC5 is secure.

The major advantage that private-key encryption has over public-key is that it is typically much faster to decrypt, and can thus be used where a fast conversion is required, such as in real-time encryption.

3 Coding

AES (or Rijndael) is the new replacement for DES, and uses 128-bit blocks with 128, 192 and 256 bit encryption keys. It was selected by NIST in 2001 (after a five year standardisation process). The name Rijndael comes from its Belgium creators: Joan Daemen and Vincent Rijmen. The key has an IV and a key element, where the IV gives the overall key some variation. In this case the key is 256 bits, and the IV is 128 bits. The following defines the code used to implement AES using Microsoft .NET [4].

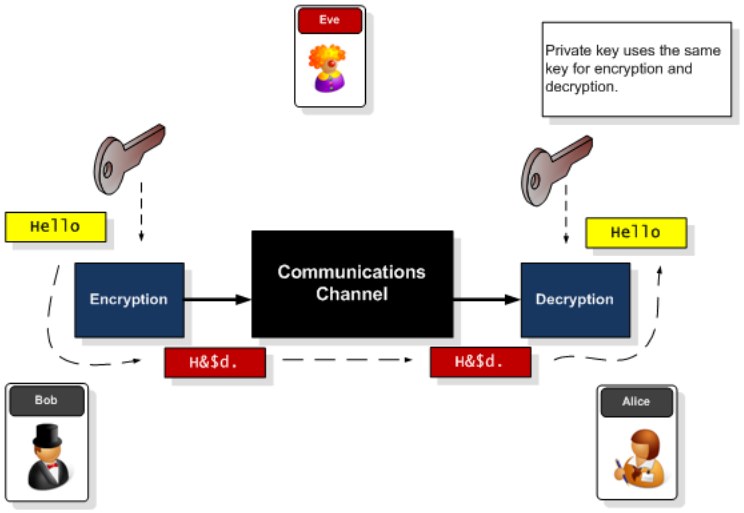

Fig. 2. Figure 2: Private key encryption

Listing 1.1. 3DES Code.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections;
using System.Security.Cryptography;
using System.IO;
using System.Text;

public partial class _Default5 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button3_Click(object sender, EventArgs
        e)
    {
        try
        {
            Rijndael myRijndael = new RijndaelManaged();
```

```
25
26
27              myRijndael.Key = StringToByte(this.tbKey.Text
                    , 32); // convert to 32 characters - 256
                    bits
28              myRijndael.IV = StringToByte("0123456789
                    ABCDEF"); // 16 chars for IV
29
30              byte[] key = myRijndael.Key;
31              byte[] IV = myRijndael.IV;
32
33              ICryptoTransform encryptor = myRijndael.
                    CreateEncryptor(key, IV);
34
35              MemoryStream msEncrypt = new MemoryStream();
36              CryptoStream csEncrypt = new CryptoStream(
                    msEncrypt, encryptor, CryptoStreamMode.
                    Write);
37
38              // Write all data to the crypto stream and
                    flush it.
39              csEncrypt.Write(StringToByte(this.tbMessage.
                    Text), 0, StringToByte(this.tbMessage.Text
                    ).Length);
40              csEncrypt.FlushFinalBlock();
41
42              // Get the encrypted array of bytes.
43              byte[] encrypted = msEncrypt.ToArray();
44
45              this.tbEncrypt.Text = ByteToString(encrypted)
                    ;
46
47              ICryptoTransform decryptor = myRijndael.
                    CreateDecryptor(key, IV);
48
49              // Now decrypt the previously encrypted
                    message using the decryptor
50              MemoryStream msDecrypt = new MemoryStream(
                    encrypted);
51              CryptoStream csDecrypt = new CryptoStream(
                    msDecrypt, decryptor, CryptoStreamMode.
                    Read);
52
53              this.tbDecrypt.Text = ByteToString(csDecrypt)
                    ;
```

```
54          }
55          catch (Exception ex)
56          {
57              this.tbEncrypt.Text = ex.Message.ToString();
58          }
59
60      }
61      public static byte[] StringToByte(string
            StringToConvert)
62      {
63
64          char[] CharArray = StringToConvert.ToCharArray();
65          byte[] ByteArray = new byte[CharArray.Length];
66          for (int i = 0; i < CharArray.Length; i++)
67          {
68              ByteArray[i] = Convert.ToByte(CharArray[i]);
69          }
70          return ByteArray;
71      }
72      public static byte[] StringToByte(string
            StringToConvert, int length)
73      {
74
75          char[] CharArray = StringToConvert.ToCharArray();
76          byte[] ByteArray = new byte[length];
77          for (int i = 0; i < CharArray.Length; i++)
78          {
79              ByteArray[i] = Convert.ToByte(CharArray[i]);
80          }
81          return ByteArray;
82      }
83      public static string ByteToString(CryptoStream buff)
84      {
85          string sbinary = "";
86          int b = 0;
87          do
88          {
89              b = buff.ReadByte();
90              if (b != -1) sbinary += ((char)b);
91
92          } while (b != -1);
93          return (sbinary);
94      }
95      public static string ByteToString(byte[] buff)
96      {
```

```
97            string sbinary = "";
98            for (int i = 0; i < buff.Length; i++)
99            {
100               sbinary += buff[i].ToString("X2"); // hex
                      format
101           }
102           return (sbinary);
103       }
104   }
```

4 Testing

In this case, if we try "test" as the key, and test message of: "This is a test message" which should get:

54A6B8A846B61EFBFD258AF2B1E7BF129A24545CAEDC315DA1D3F924E4AA2F00

Also, a key of "test" with a message of "test" gives:

AECC52950EFC49F6B2B2407ECEE65FE5

which is 32 characters, and thus relates to 128 bits, which is the block size (as "test" fits into a single block). All our outputs will thus be a multiple of 32 hex characters.

References

1. William J Buchanan. *Distributed Systems And Networks*. McGraw-Hill Higher Education, 2001.
2. William J Buchanan. *The complete handbook of the Internet*. Springer, 2002.
3. William J Buchanan. *Handbook of Data Communications and Networks*. Kluwer Academic Publishers, 2005.
4. William J Buchanan. Aes encryption in .net. `http://buchananweb.co.uk/security15.aspx`, April 2011.
5. William J Buchanan. Security and forensic computing: Encryption. `http://buchananweb.co.uk/index_sfc_napier.html`, March 2011.
6. L Fan, W Buchanan, C. Thuemmler, O. Lo, A. Khedim, Uthmani O., A. Lawson, and D. Bell. Dacar platform for ehealth services cloud. *IEEE Cloud 2011*, 2011.
7. Z. Kwecka, W. Buchanan, and D. Spiers. Minimising collateral damage: Privacy-preserving investigative data acquisition platform. *International Journal of Information Technologies and Systems Approach (IJITSA) : Special issue on Privacy and Security Issues in IT*, 4(2), 2010.
8. Z. Kwecka, W. Buchanan, and D. Spiers. Privacy-preserving data acquisition protocol. *IEEE International Conference of Computational Methods in Electrical and Electronics Engineering*, pages 131 – 136, 2010.
9. O. Uthmani, W. Buchanan, A. Lawson, and L. Fan. Novel information sharing syntax for data sharing between police and community partners, using role-based security. *Proceedings of the 9th European Conference on Information Warfare and Security*, pages 394–402, 2010.