**ASP.NET（C#）常用数据加密和解密方法汇总**

数据加密技术是网络中最基本的安全技术，主要是通过对网络张传输的信息进行数据加密来保障其安全性，这是一种主动安全防御策略，用最小的代价即可为信息提供相当大的安全保护。

# 1 加密的基本概念

“加密”，是一种限制对网络上传输数据的访问权的技术。原始数据（也称为明文，plaintext）被加密设备（硬件或软件）和密钥加密而产生的经过编码的数据称为密文（ciphertext）。将密文还原成原始明文的过程称为解密，它是加密的反向处理，但解密者必须利用相同类型的加密设备和密钥对密文进行解密。

## 加密的基本功能

1. 防止不速之客查看机密的数据文件；
2. 防止机密数据被泄露或篡改；
3. 防止特权用户（如系统管理员）查看私人数据文件；
4. 使入侵者不能轻易地查找一个系统的文件；

数据加密是确保计算机网络安全的一种重要机制，虽然由于成本、技术和管理上的复杂性等原因，目前尚未在网路中普及，但数据加密的确是实现分布式系统和网络环境下数据安全的重要手段之一。

## 1.2 加密方式

**数据加密可在网路OSI七层协议（OSI是Open System Interconnect的缩写，意在开放式系统互联。国际标准组织（国际标准化组织）制定了OSI模型。这个模型把网络通信的工作分为7层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层）的多层上实现，所以从加密技术应用的逻辑位置看，有三种方式：**

1. **链路加密**：通常把网络层以下的加密叫链路加密，主要用于保护通信节点间传输的数据，加解密由置于线路上的密码设备实现。根据传递的数据的同步方式又可分为***同步通信加密***和***异步通信加密***两种，同步通信加密又包含***字节同步通信加密***和***位同步通信加密***。
2. **节点加密**：是对链路加密的改进。在协议传输层上进行加密，主要是对源节点和目标节点之间传输数据进行加密保护，与链路加密类似.只是加密算法要结合在依附于节点的加密模件中，克服了链路加密在节点处易遭非法存取的缺点。
3. **端对端加密：**网络层以上的加密称为端对端加密。是面向网络层主体。对应用层的数据信息进行加密，易于用软件实现，且成本低，但密钥管理问题困难，主要适合大型网络系统中信息在多个发方和收方之间传输的情况。

# 2 数据加密的应用

1. 媒体加密：DRM
2. 文件加密：文本加密、pdf、word
3. 数据加密：ASP.NET（C#）中的数据加密
4. 硬件加密：加密狗

# 3 加密技术发展趋势

1. **私用密钥加密技术与公开密钥加密技术相结合:**鉴于两种密码体制加密的特点，在实际应用中可以采用折衷方案，即结合使用DES/IDEA和RSA，以DES为"内核"，RSA为"外壳"，对于网络中传输的数据可用DES或IDEA加密，而加密用的密钥则用RSA加密传送，此种方法既保证了数据安全又提高了加密和解密的速度，这也是目前加密技术发展的新方向之一。
2. **寻求新算法:跳出以常见的迭代为基础的构造思路，脱离基于某些数学问题复杂性的构造方法。**如刘尊全先生提出的刘氏算法，是一种基于密钥的公开密钥体制，它采用了随机性原理构造加解密变换，并将其全部运算控制隐匿于密钥中，密钥长度可变。它是采用选取一定长度的分割来构造大的搜索空间，从而实现一次非线性变换。此种加密算法加密强度高、速度快、计算开销低。
3. **加密最终将被集成到系统和网络中**，例如IPV6协议就已有了内置加密的支持，在硬件方面，Intel公司正研制一种加密协处理器。它可以集成到微机的主极上。

# 4 加密技术的分类

加密类型可以简单地分为四种：

1. **根本不考虑解密问题；**
2. **私用密钥加密技术：**对称式加密（Symmetric Key Encryption）：对称式加密方式对加密和解密使用相同的密钥。通常，这种加密方式在应用中难以实施，因为用同一种安全方式共享密钥很难。如：RC4、RC2、DES和AES系列加密算法。
3. **公开密钥加密技术：**非对称密钥加密（Asymmetric Key Encryption）：非对称密钥加密使用一组公共/私人密钥系统，加密时使用一种密钥，解密时使用另一种密钥。公共密钥可以广泛共享和透露。当需要用加密方式向服务器外部传送数据时，这种加密方式更方便。如：RSA
4. **数字证书（Certificate）：**数字证书是一种非对称密钥加密，但是，一个组织可以使用证书并通过数字签名将一组公钥和私钥与其拥有者相关联。

# 5 对称加密之DES加密与解密

## 5.1 对称加密

对称加密，是一种比较传统的加密方式，其加密运算、解密运算使用的是同样的密钥，信息的发送者和信息的接收者在进行信息的传输与处理时，必须共同持有该密码（称为对称密码）。因此，通信双方都必须获得这把钥匙，并保持钥匙的秘密。

单钥密码系统的安全性依赖于以下两个因素：

1. 加密算法必须是足够强的，仅仅基于密文本身去解密信息在实践上是不可能的。
2. 加密方法的安全性依赖于密钥的秘密性，而不是算法的秘密性，因此，我们没有必要确保算法的秘密性（事实上，现实中使用的很多单钥密码系统的算法都是公开的），但是我们一定要保证密钥的秘密性。

DES(Data Encryption Standard)和TripleDES是对称加密的两种实现。

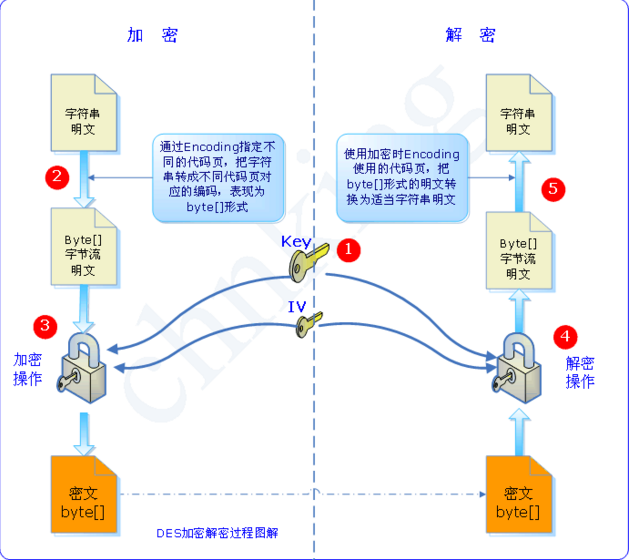
DES和TripleDES基本算法一致，只是TripleDES算法提供的key位数更多，加密可靠性更高。

DES使用的密钥key为8字节，初始向量IV也是8字节。

TripleDES使用24字节的key，初始向量IV也是8字节。

两种算法都是以8字节为一个块进行加密，一个数据块一个数据块的加密，一个8字节的明文加密后的密文也是8字节。如果明文长度不为8字节的整数倍，添加值为0的字节凑满8字节整数倍。所以加密后的密文长度一定为8字节的整数倍。

## 5.2 加密解密过程



DES加密解密过程图

上图是整个DES和TripleDES算法的加密解密过程，下面以TripleDES为例，结合dotnet分析加密解密的各个步骤，并给出相关实现代码。

**1、 生成key和IV**

System.Security.Cryptography. TripleDESCryptoServiceProvider类是dotnet中实现TripleDES算法的主要的类。

TripleDESCryptoServiceProvider类只有一个构造方法TripleDESCryptoServiceProvider（），这个方法把一些属性初始化：

KeySize（加密密钥长度，以位为单位）= 192（24字节）

BlockSize（加密处理的数据块大小，以位为单位）= 64（8字节）

FeedbackSize（加密数据块后返回的数据大小，以位为单位）= 64（8字节）

TripleDESCryptoServiceProvider构造方法同时会初始化一组随机的key和IV。

默认的TripleDESCryptoServiceProvider的key为24字节，IV为8字节，加密数据块为8字节。

生成key和IV的代码很简单：

TripleDESCryptoServiceProvider tDESalg = new TripleDESCryptoServiceProvider();

byte[] keyArray = tDESalg.Key;

byte[] IVArray = tDESalg.IV;

生成的key和IV在加密过程和解密过程都要使用。

**2、 字符串明文转成某一代码页对应的编码字节流**

待加密的数据可能有两种形式，一种是二进制的数据，本身就是一组字节流，这样的数据可以跳过这一步，直接进入加密步骤。还有一种情况是字符串数据，字符串中同样的字符使用不同的代码页会生成不同的字节码，所以从字符串到字节流的转换是需要指定使用何种编码的。在解密之后，要从字节流转换到字符串就要使用相同的代码页解码，否则就会出现乱码。

// 待加密的字符串

string plainTextString = "Here is some data to encrypt. 这里是一些要加密的数据。";

// 使用utf-8编码（也可以使用其它的编码）

Encoding sEncoding = Encoding.GetEncoding("utf-8");

// 把字符串明文转换成utf-8编码的字节流

byte[] plainTextArray = sEncoding.GetBytes(plainTextString);

**3、 加密操作**

加密的原料是明文字节流，TripleDES算法对字节流进行加密，返回的是加密后的字节流。同时要给定加密使用的key和IV。

// 把字符串明文转换成utf-8编码的字节流

byte[] plainTextArray = sEncoding.GetBytes(plainTextString);

public static byte[] EncryptString(byte[] plainTextArray, byte[] Key, byte[] IV)

{

    // 建立一个MemoryStream，这里面存放加密后的数据流

    MemoryStream mStream = new MemoryStream();

    // 使用MemoryStream 和key、IV新建一个CryptoStream 对象

    CryptoStream cStream = new CryptoStream(mStream,

        new TripleDESCryptoServiceProvider().CreateEncryptor(Key, IV),

        CryptoStreamMode.Write);

    // 将加密后的字节流写入到MemoryStream

    cStream.Write(plainTextArray, 0, plainTextArray.Length);

    //把缓冲区中的最后状态更新到MemoryStream，并清除cStream的缓存区

    cStream.FlushFinalBlock();

    // 把解密后的数据流转成字节流

    byte[] ret = mStream.ToArray();

    // 关闭两个streams.

    cStream.Close();

    mStream.Close();

    return ret;

}

**4、 解密操作**

解密操作解密上面步骤生成的密文byte[]，需要使用到加密步骤使用的同一组Key和IV。

// 调用解密方法，返回已解密数据的byte[]

byte[] finalPlainTextArray = DecryptTextFromMemory(Data, keyArray, IVArray);

public static byte[] DecryptTextFromMemory(byte[] EncryptedDataArray, byte[] Key, byte[] IV)

{

    // 建立一个MemoryStream，这里面存放加密后的数据流

    MemoryStream msDecrypt = new MemoryStream(EncryptedDataArray);

    // 使用MemoryStream 和key、IV新建一个CryptoStream 对象

    CryptoStream csDecrypt = new CryptoStream(msDecrypt,

        new TripleDESCryptoServiceProvider().CreateDecryptor(Key, IV),

        CryptoStreamMode.Read);

    // 根据密文byte[]的长度（可能比加密前的明文长），新建一个存放解密后明文的byte[]

    byte[] DecryptDataArray = new byte[EncryptedDataArray.Length];

    // 把解密后的数据读入到DecryptDataArray

    csDecrypt.Read(DecryptDataArray, 0, DecryptDataArray.Length);

    msDecrypt.Close();

    csDecrypt.Close();

    return DecryptDataArray;

}

有一点需要注意，DES加密是以数据块为单位加密的，8个字节一个数据块，如果待加密明byte[]的长度不是8字节的整数倍，算法先用值为“0”的byte补足8个字节，然后进行加密。所以加密后的密文长度一定是8的整数倍。这样的密文解密后如果补了0值的byte，则解密后这些0值的byte依然存在。比如上例中要加密的明文是：

“Here is some data to encrypt. 这里是一些要加密的数据。”

转成明文byte[]后是66个字节，DES算法就会补上6个0值的byte，补到72个字节。这样加密后再解密回来的密文byte[]解码后的字符串就是这样的：

"Here is some data to encrypt. 这里是一些要加密的数据。\0\0\0\0\0\0"

**5、 从编码字节流转成字符串明文**

// 使用前面定义的Encoding，utf-8的编码把byte[]转成字符串

plainTextString = sEncoding.GetString(finalPlainTextArray);

# 6 非对称加密之RSA加密和解密的讲解

RSA公钥加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman在（美国麻省理工学院）开发的。RSA取名来自开发他们三者的名字。RSA是目前最有影响力的公钥加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。RSA算法是第一个能同时用于加密和数字签名的算法，也易于理解和操作。

RSA是被研究得最广泛的公钥算法，从提出到现在已近二十年，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的公钥方案之一。RSA的安全性依赖于大数的因子分解，但并没有从理论上证明破译RSA的难度与大数分解难度等价。即RSA的重大缺陷是无法从理论上把握它的保密性能如何，而且密码学界多数人士倾向于因子分解不是NPC问题。

RSA的缺点主要有：

A)产生密钥很麻烦，受到素数产生技术的限制，因而难以做到一次一密。

B)分组长度太大，为保证安全性，n 至少也要 600bits以上，使运算代价很高，尤其是速度较慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化。目前，SET(Secure Electronic Transaction)协议中要求CA采用2048bits长的密钥，其他实体使用1024比特的密钥。

C)RSA密钥长度随着保密级别提高，增加很快。下表列出了对同一安全级别所对应的密钥长度。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 保密级别 | 对称密钥长度（bit） | RSA密钥长度（bit） | ECC密钥长度（bit） | 保密年限 |
| 80 | 80 | 1024 | 160 | 2010 |
| 112 | 112 | 2048 | 224 | 2030 |
| 128 | 128 | 3072 | 256 | 2040 |
| 192 | 192 | 7680 | 384 | 2080 |
| 256 | 256 | 15360 | 512 | 2120 |

这种算法1978年就出现了，它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest, AdiShamir 和Leonard Adleman。早在1973年，英国国家通信总局的数学家Clifford Cocks就发现了类似的算法。但是他的发现被列为绝密，直到1998年才公诸于世。

RSA算法是一种非对称密码算法，所谓非对称，就是指该算法需要一对密钥，使用其中一个加密，则需要用另一个才能解密。

RSA的算法涉及三个参数，n、e1、e2。

其中，n是两个大质数p、q的积，n的二进制表示时所占用的位数，就是所谓的密钥长度。

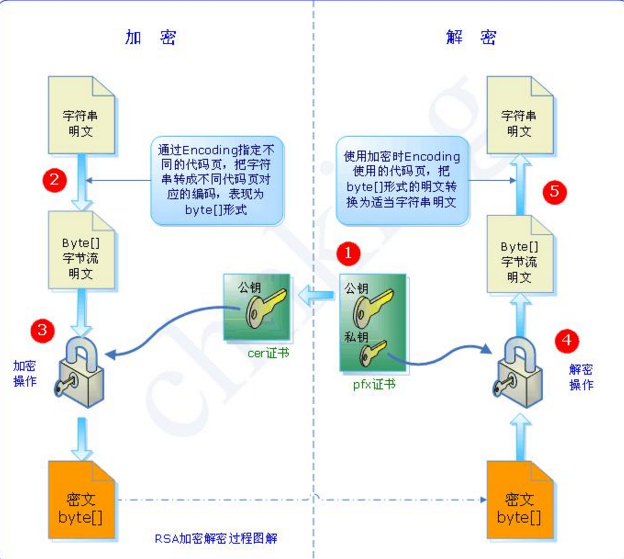
e1和e2是一对相关的值，e1可以任意取，但要求e1与(p-1)\*(q-1)互质；再选择e2，要求(e2\*e1)mod((p-1)\*(q-1))=1。

(n及e1),(n及e2)就是密钥对。

RSA加解密的算法完全相同,设A为明文，B为密文，则：A=B^e1 mod n；B=A^e2 mod n；

e1和e2可以互换使用，即：

A=B^e2 mod n；B=A^e1 mod n;



C#代码实现

需引用using System.Security.Cryptography;

/// <summary>  
        /// RSA加密  
        /// </summary>  
        /// <param name="publickey"></param>  
        /// <param name="content"></param>  
        /// <returns></returns>  
        public static string RSAEncrypt(string publickey, string content)  
        {  
            publickey = @"<RSAKeyValue><Modulus>5m9m14XH3oqLJ8bNGw9e4rGpXpcktv9MSkHSVFVMjHbfv+SJ5v0ubqQxa5YjLN4vc49z7SVju8s0X4gZ6AzZTn06jzWOgyPRV54Q4I0DCYadWW4Ze3e+BOtwgVU1Og3qHKn8vygoj40J6U85Z/PTJu3hN1m75Zr195ju7g9v4Hk=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>";  
            RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();  
            byte[] cipherbytes;  
            rsa.FromXmlString(publickey);  
            cipherbytes = rsa.Encrypt(Encoding.UTF8.GetBytes(content), false);  
  
            return Convert.ToBase64String(cipherbytes);  
        }  
  
        /// <summary>  
        /// RSA解密  
        /// </summary>  
        /// <param name="privatekey"></param>  
        /// <param name="content"></param>  
        /// <returns></returns>  
        public static string RSADecrypt(string privatekey, string content)  
        {  
            privatekey = @"<RSAKeyValue><Modulus>5m9m14XH3oqLJ8bNGw9e4rGpXpcktv9MSkHSVFVMjHbfv+SJ5v0ubqQxa5YjLN4vc49z7SVju8s0X4gZ6AzZTn06jzWOgyPRV54Q4I0DCYadWW4Ze3e+BOtwgVU1Og3qHKn8vygoj40J6U85Z/PTJu3hN1m75Zr195ju7g9v4Hk=</Modulus><Exponent>AQAB</Exponent><P>/hf2dnK7rNfl3lbqghWcpFdu778hUpIEBixCDL5WiBtpkZdpSw90aERmHJYaW2RGvGRi6zSftLh00KHsPcNUMw==</P><Q>6Cn/jOLrPapDTEp1Fkq+uz++1Do0eeX7HYqi9rY29CqShzCeI7LEYOoSwYuAJ3xA/DuCdQENPSoJ9KFbO4Wsow==</Q><DP>ga1rHIJro8e/yhxjrKYo/nqc5ICQGhrpMNlPkD9n3CjZVPOISkWF7FzUHEzDANeJfkZhcZa21z24aG3rKo5Qnw==</DP><DQ>MNGsCB8rYlMsRZ2ek2pyQwO7h/sZT8y5ilO9wu08Dwnot/7UMiOEQfDWstY3w5XQQHnvC9WFyCfP4h4QBissyw==</DQ><InverseQ>EG02S7SADhH1EVT9DD0Z62Y0uY7gIYvxX/uq+IzKSCwB8M2G7Qv9xgZQaQlLpCaeKbux3Y59hHM+KpamGL19Kg==</InverseQ><D>vmaYHEbPAgOJvaEXQl+t8DQKFT1fudEysTy31LTyXjGu6XiltXXHUuZaa2IPyHgBz0Nd7znwsW/S44iql0Fen1kzKioEL3svANui63O3o5xdDeExVM6zOf1wUUh/oldovPweChyoAdMtUzgvCbJk1sYDJf++Nr0FeNW1RB1XG30=</D></RSAKeyValue>";  
            RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();  
            byte[] cipherbytes;  
            rsa.FromXmlString(privatekey);  
            cipherbytes = rsa.Decrypt(Convert.FromBase64String(content), false);  
  
            return Encoding.UTF8.GetString(cipherbytes);  
        }

# 7 ASP.NET(C#)常用加密类调用的讲解

## 7.1 C#常用加密解密类库代码如下：

/// <summary>

      /// MD5 加密静态方法

      /// </summary>

      /// <param name="EncryptString">待加密的密文</param>

      /// <returns>returns</returns>

      public static string MD5Encrypt(string EncryptString)

      {

          if (string.IsNullOrEmpty(EncryptString)) { throw (new Exception("密文不得为空")); }

          MD5 m\_ClassMD5 = new MD5CryptoServiceProvider();

          string m\_strEncrypt = "";

          try

          {

              m\_strEncrypt = BitConverter.ToString(m\_ClassMD5.ComputeHash(Encoding.Default.GetBytes(EncryptString))).Replace("-", "");

          }

         catch (ArgumentException ex) { throw ex; }

         catch (CryptographicException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

          finally { m\_ClassMD5.Clear(); }

          return m\_strEncrypt;

      }

      /// <summary>

      /// DES 加密(数据加密标准，速度较快，适用于加密大量数据的场合)

      /// </summary>

      /// <param name="EncryptString">待加密的密文</param>

      /// <param name="EncryptKey">加密的密钥</param>

      /// <returns>returns</returns>

      public static string DESEncrypt(string EncryptString, string EncryptKey)

      {

          if (string.IsNullOrEmpty(EncryptString)) { throw (new Exception("密文不得为空")); }

          if (string.IsNullOrEmpty(EncryptKey)) { throw (new Exception("密钥不得为空")); }

          if (EncryptKey.Length != 8) { throw (new Exception("密钥必须为8位")); }

          byte[] m\_btIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };

          string m\_strEncrypt = "";

         DESCryptoServiceProvider m\_DESProvider = new DESCryptoServiceProvider();

          try

          {

              byte[] m\_btEncryptString = Encoding.Default.GetBytes(EncryptString);

              MemoryStream m\_stream = new MemoryStream();

              CryptoStream m\_cstream = new CryptoStream(m\_stream, m\_DESProvider.CreateEncryptor(Encoding.Default.GetBytes(EncryptKey), m\_btIV), CryptoStreamMode.Write);

              m\_cstream.Write(m\_btEncryptString, 0, m\_btEncryptString.Length);

              m\_cstream.FlushFinalBlock();

              m\_strEncrypt = Convert.ToBase64String(m\_stream.ToArray());

              m\_stream.Close(); m\_stream.Dispose();

              m\_cstream.Close(); m\_cstream.Dispose();

          }

          catch (IOException ex) { throw ex; }

          catch (CryptographicException ex) { throw ex; }

          catch (ArgumentException ex) { throw ex; }

          catch (Exception ex) { throw ex; }

          finally { m\_DESProvider.Clear(); }

          return m\_strEncrypt;

      }

      /// <summary>

      /// DES 解密(数据加密标准，速度较快，适用于加密大量数据的场合)

      /// </summary>

      /// <param name="DecryptString">待解密的密文</param>

      /// <param name="DecryptKey">解密的密钥</param>

      /// <returns>returns</returns>

      public static string DESDecrypt(string DecryptString, string DecryptKey)

      {

         if (string.IsNullOrEmpty(DecryptString)) { throw (new Exception("密文不得为空")); }

          if (string.IsNullOrEmpty(DecryptKey)) { throw (new Exception("密钥不得为空")); }

          if (DecryptKey.Length != 8) { throw (new Exception("密钥必须为8位")); }

          byte[] m\_btIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };

         string m\_strDecrypt = "";

         DESCryptoServiceProvider m\_DESProvider = new DESCryptoServiceProvider();

         try

        {

             byte[] m\_btDecryptString = Convert.FromBase64String(DecryptString);

             MemoryStream m\_stream = new MemoryStream();

             CryptoStream m\_cstream = new CryptoStream(m\_stream, m\_DESProvider.CreateDecryptor(Encoding.Default.GetBytes(DecryptKey), m\_btIV), CryptoStreamMode.Write);

             m\_cstream.Write(m\_btDecryptString, 0, m\_btDecryptString.Length);

             m\_cstream.FlushFinalBlock();

             m\_strDecrypt = Encoding.Default.GetString(m\_stream.ToArray());

             m\_stream.Close(); m\_stream.Dispose();

             m\_cstream.Close(); m\_cstream.Dispose();

         }

         catch (IOException ex) { throw ex; }

         catch (CryptographicException ex) { throw ex; }

         catch (ArgumentException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

         finally { m\_DESProvider.Clear(); }

         return m\_strDecrypt;

     }

     /// <summary>

     /// RC2 加密(用变长密钥对大量数据进行加密)

     /// </summary>

     /// <param name="EncryptString">待加密密文</param>

     /// <param name="EncryptKey">加密密钥</param>

     /// <returns>returns</returns>

     public static string RC2Encrypt(string EncryptString, string EncryptKey)

     {

         if (string.IsNullOrEmpty(EncryptString)) { throw (new Exception("密文不得为空")); }

         if (string.IsNullOrEmpty(EncryptKey)) { throw (new Exception("密钥不得为空")); }

         if (EncryptKey.Length < 5 || EncryptKey.Length > 16) { throw (new Exception("密钥必须为5-16位")); }

         string m\_strEncrypt = "";

         byte[] m\_btIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };

         RC2CryptoServiceProvider m\_RC2Provider = new RC2CryptoServiceProvider();

         try

        {

             byte[] m\_btEncryptString = Encoding.Default.GetBytes(EncryptString);

             MemoryStream m\_stream = new MemoryStream();

             CryptoStream m\_cstream = new CryptoStream(m\_stream, m\_RC2Provider.CreateEncryptor(Encoding.Default.GetBytes(EncryptKey), m\_btIV), CryptoStreamMode.Write);

             m\_cstream.Write(m\_btEncryptString, 0, m\_btEncryptString.Length);

             m\_cstream.FlushFinalBlock();

             m\_strEncrypt = Convert.ToBase64String(m\_stream.ToArray());

             m\_stream.Close(); m\_stream.Dispose();

            m\_cstream.Close(); m\_cstream.Dispose();

         }

         catch (IOException ex) { throw ex; }

         catch (CryptographicException ex) { throw ex; }

         catch (ArgumentException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

         finally { m\_RC2Provider.Clear(); }

         return m\_strEncrypt;

     }

     /// <summary>

     /// RC2 解密(用变长密钥对大量数据进行加密)

     /// </summary>

     /// <param name="DecryptString">待解密密文</param>

     /// <param name="DecryptKey">解密密钥</param>

     /// <returns>returns</returns>

     public static string RC2Decrypt(string DecryptString, string DecryptKey)

     {

         if (string.IsNullOrEmpty(DecryptString)) { throw (new Exception("密文不得为空")); }

         if (string.IsNullOrEmpty(DecryptKey)) { throw (new Exception("密钥不得为空")); }

        if (DecryptKey.Length < 5 || DecryptKey.Length > 16) { throw (new Exception("密钥必须为5-16位")); }

         byte[] m\_btIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };

         string m\_strDecrypt = "";

         RC2CryptoServiceProvider m\_RC2Provider = new RC2CryptoServiceProvider();

        try

        {

            byte[] m\_btDecryptString = Convert.FromBase64String(DecryptString);

             MemoryStream m\_stream = new MemoryStream();

             CryptoStream m\_cstream = new CryptoStream(m\_stream, m\_RC2Provider.CreateDecryptor(Encoding.Default.GetBytes(DecryptKey), m\_btIV), CryptoStreamMode.Write);

             m\_cstream.Write(m\_btDecryptString, 0, m\_btDecryptString.Length);

             m\_cstream.FlushFinalBlock();

             m\_strDecrypt = Encoding.Default.GetString(m\_stream.ToArray());

             m\_stream.Close(); m\_stream.Dispose();

             m\_cstream.Close(); m\_cstream.Dispose();

         }

         catch (IOException ex) { throw ex; }

        catch (CryptographicException ex) { throw ex; }

         catch (ArgumentException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

        finally { m\_RC2Provider.Clear(); }

         return m\_strDecrypt;

     }

     /// <summary>

     /// 3DES 加密(基于DES，对一块数据用三个不同的密钥进行三次加密，强度更高)

     /// </summary>

     /// <param name="EncryptString">待加密密文</param>

     /// <param name="EncryptKey1">密钥一</param>

     /// <param name="EncryptKey2">密钥二</param>

    /// <param name="EncryptKey3">密钥三</param>

     /// <returns>returns</returns>

     public static string DES3Encrypt(string EncryptString, string EncryptKey1, string EncryptKey2, string EncryptKey3)

     {

         string m\_strEncrypt = "";

         try

         {

             m\_strEncrypt = DESEncrypt(EncryptString, EncryptKey3);

             m\_strEncrypt = DESEncrypt(m\_strEncrypt, EncryptKey2);

            m\_strEncrypt = DESEncrypt(m\_strEncrypt, EncryptKey1);

         }

         catch (Exception ex) { throw ex; }

         return m\_strEncrypt;

     }

     /// <summary>

     /// 3DES 解密(基于DES，对一块数据用三个不同的密钥进行三次加密，强度更高)

     /// </summary>

     /// <param name="DecryptString">待解密密文</param>

     /// <param name="DecryptKey1">密钥一</param>

     /// <param name="DecryptKey2">密钥二</param>

     /// <param name="DecryptKey3">密钥三</param>

     /// <returns>returns</returns>

     public static string DES3Decrypt(string DecryptString, string DecryptKey1, string DecryptKey2, string DecryptKey3)

     {

         string m\_strDecrypt = "";

         try

         {

            m\_strDecrypt = DESDecrypt(DecryptString, DecryptKey1);

             m\_strDecrypt = DESDecrypt(m\_strDecrypt, DecryptKey2);

             m\_strDecrypt = DESDecrypt(m\_strDecrypt, DecryptKey3);

         }

         catch (Exception ex) { throw ex; }

         return m\_strDecrypt;

     }

     /// <summary>

     /// AES 加密(高级加密标准，是下一代的加密算法标准，速度快，安全级别高，目前 AES 标准的一个实现是 Rijndael 算法)

     /// </summary>

     /// <param name="EncryptString">待加密密文</param>

     /// <param name="EncryptKey">加密密钥</param>

    /// <returns></returns>

    public static string AESEncrypt(string EncryptString, string EncryptKey)

     {

        if (string.IsNullOrEmpty(EncryptString)) { throw (new Exception("密文不得为空")); }

         if (string.IsNullOrEmpty(EncryptKey)) { throw (new Exception("密钥不得为空")); }

         string m\_strEncrypt = "";

         byte[] m\_btIV = Convert.FromBase64String("Rkb4jvUy/ye7Cd7k89QQgQ==");

         Rijndael m\_AESProvider = Rijndael.Create();

         try

         {

             byte[] m\_btEncryptString = Encoding.Default.GetBytes(EncryptString);

             MemoryStream m\_stream = new MemoryStream();

             CryptoStream m\_csstream = new CryptoStream(m\_stream, m\_AESProvider.CreateEncryptor(Encoding.Default.GetBytes(EncryptKey), m\_btIV), CryptoStreamMode.Write);

             m\_csstream.Write(m\_btEncryptString, 0, m\_btEncryptString.Length); m\_csstream.FlushFinalBlock();

             m\_strEncrypt = Convert.ToBase64String(m\_stream.ToArray());

             m\_stream.Close(); m\_stream.Dispose();

             m\_csstream.Close(); m\_csstream.Dispose();

         }

         catch (IOException ex) { throw ex; }

         catch (CryptographicException ex) { throw ex; }

         catch (ArgumentException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

         finally { m\_AESProvider.Clear(); }

         return m\_strEncrypt;

     }

     /// <summary>

     /// AES 解密(高级加密标准，是下一代的加密算法标准，速度快，安全级别高，目前 AES 标准的一个实现是 Rijndael 算法)

     /// </summary>

     /// <param name="DecryptString">待解密密文</param>

     /// <param name="DecryptKey">解密密钥</param>

     /// <returns></returns>

     public static string AESDecrypt(string DecryptString, string DecryptKey)

     {

         if (string.IsNullOrEmpty(DecryptString)) { throw (new Exception("密文不得为空")); }

         if (string.IsNullOrEmpty(DecryptKey)) { throw (new Exception("密钥不得为空")); }

         string m\_strDecrypt = "";

         byte[] m\_btIV = Convert.FromBase64String("Rkb4jvUy/ye7Cd7k89QQgQ==");

         Rijndael m\_AESProvider = Rijndael.Create();

         try

         {

             byte[] m\_btDecryptString = Convert.FromBase64String(DecryptString);

             MemoryStream m\_stream = new MemoryStream();

             CryptoStream m\_csstream = new CryptoStream(m\_stream, m\_AESProvider.CreateDecryptor(Encoding.Default.GetBytes(DecryptKey), m\_btIV), CryptoStreamMode.Write);

             m\_csstream.Write(m\_btDecryptString, 0, m\_btDecryptString.Length); m\_csstream.FlushFinalBlock();

             m\_strDecrypt = Encoding.Default.GetString(m\_stream.ToArray());

             m\_stream.Close(); m\_stream.Dispose();

             m\_csstream.Close(); m\_csstream.Dispose();

         }

         catch (IOException ex) { throw ex; }

         catch (CryptographicException ex) { throw ex; }

         catch (ArgumentException ex) { throw ex; }

         catch (Exception ex) { throw ex; }

         finally { m\_AESProvider.Clear(); }

         return m\_strDecrypt;

     }

## 7.2 数据加密和解密简单代码调用如下：

Response.Write("<br>-----------MD5加密---------------<br>");

Response.Write(SDKSecurity.MD5Encrypt("仰天一笑"));  
Response.Write("<br>-----------DES加密---------------<br>");

Response.Write(SDKSecurity.DESEncrypt("仰天一笑", "anson-xu"));

Response.Write("<br>-----------DES解密---------------<br>");

Response.Write(SDKSecurity.DESDecrypt("l06JvJ45r/lb9iKzSXl47Q==", "anson-xu"));  
Response.Write("<br>-----------AES加密---------------<br>");

Response.Write(SDKSecurity.AESEncrypt("仰天一笑", "ansonxuyu"));

Response.Write("<br>-----------AES解密---------------<br>");

Response.Write(SDKSecurity.AESDecrypt("avwKL+MO8+zoLHvzk0+TBA==", "ansonxuyu"));

## 7.3 数据加密和解密调用后运行效果图如下：

