实 验 七　数据加密与解密技术

7.1　实验目的

网络数据在不安全信道上进行传输，会受到篡改与伪造等威胁，在传输前对数据进行加密，接收后再进行解密能够保证数据的完整性、安全性。本实验介绍数据的加密解密原理与实现方法。

7.2　加密与解密算法原理

网络中传输的数据会受到各种威胁，在传输数据前进行加密，接收到加密数据后再进行解密，能够保证数据的完整性、安全性。加密方法主要分为两大类：对称加密和不对称加密。

私钥加密算法使用单个私钥来加密和解密数据，私钥加密又称为对称加密，同一密钥既用于加密又用于解密。私钥加密算法的速度非常快（与公钥算法相比），它特别适用于对较大的数据流执行加密转换。私钥算法称为块密码，它用于一次加密一个数据块，加密或解密字节序列，必须逐块进行。简单块称为电子密码本 (ECB) 模式，它将相同的纯文本输入块加密为相同的密码文本输出块。如果输入纯文本流中存在重复的块，则输出密码文本流中也将存在重复的块。这些重复的输出块会使未经授权的用户察觉数据可能采用了不可靠的加密算法，进而想出可能的攻击模式。通过使用初始化向量 (IV) 加密第一个纯文本块，CBC 密码克服了与 ECB 密码关联的问题，每个密码文本块都依赖于它前面的所有块，相同的数据加密后的密文将不相同，增加破解密文的成本。

在.NET 类库的 System.Security.Cryptography 命名空间中，包含多种加密数据的类，其中实现对称加密算法的类有：

1. DESCryptoServiceProvider，使用 DES 算法；
2. TripleDESCryptoServiceProvider，使用 TripleDES 算法；
3. RijndaelManaged ，使用 Rijdael 算法；
4. AesManaged ，使用 AES 算法；
5. RC2CryptoServiceProvider ，使用 RC2 算法；
6. HMACSHA1 ，使用 SHA1 哈希函数计算基于哈希值的消息验证代码 (HMAC)。

私钥加密的缺点是双方持有相同的密钥和 IV 值，这在非安全信道中通信双方难以实现。公钥加密使用一个必须对未经授权的用户保密的私钥和一个可以对任何人公开的公钥。公钥和私钥在数学上是关联在一起的；用公钥加密的数据只能用私钥解密，而用私钥签名的数据只能用公钥验证，这种加密方法也叫不对称加密算法。经公钥加密的数据即使被截获也无法被破解。公钥算法不像私钥算法那样将数据链接成流，原因是它只能加密少量数据。公钥算法的速度很

慢（与私钥算法相比），不适合用来加密大量数据。在.NET 类库中实现不对称加密算法的类有：

1. DSACryptoServiceProvider，DSA 算法加密实现；
2. RSACryptoServiceProvider，RSA 算法加密实现；
3. ECDiffieHellmanCng，椭圆曲线 Diffie-Hellman 数字签名算法加密技术实现；
4. ECDsaCng，椭圆曲线数字签名算法的下一代加密技术实现；

RSA 允许同时进行加密和签名，但 DSA 只能用于签名，Diffie-Hellman 只能用于生成密钥。对称加密与非对称加密中的密钥不能直接以明文保存在文件中，文件被攻击者获取则失去加密意义，要采用密钥容器来保存密钥。

哈希函数是现代密码系统的基础。函数将任意长度的二进制字符串映射为固定长度的小二进制字符串（称为哈希值）。相同的数据经哈希运算值也相同，加密哈希函数一般具有的属性是：在计算时不可能将两个不同的输入通过哈希算法取为相同的值。如果数据经过少量的更改，其哈希值会有较大变化。

哈希函数通常用于数字签名和保持数据完整性。例如在传输数据前计算机对数据进行散列计算，经过传输的数据重新计算机哈希值与已有哈希值进行比较，如果哈希值匹配，则数据未被更改，如果值不匹配，则数据已被损坏。为使此系统发挥作用，计算后的哈希值须进行加密。在.NET 平台提供与哈希算法相关的类有下面几个：

1. HashAlgorithm，哈希算法实现由其派生的基类；
2. MD5CryptoServiceProvider，计算输入数据的 MD5 哈希值；
3. SHA1CryptoServiceProvider，实现计算输入数据的 SHA1 哈希值；

7.3 实验内容

7.3.1 对称算法加密本地文件

对称加密算法有多个，其中 RijndaelManaged 类需要提供用于加密的密钥和偏移向量，下面使用 RijndaelManaged 对文本执行对称加密的示例代码:

```
public static string EncrypTxt(string srctxt)
{
    RijndaelManaged aesAlg;
    aesAlg = new RijndaelManaged();
    string key = "abcdefghijklmnop";//16 个字符
    string iv = "abcdefghijklmnop";//16 个字符
    aesAlg.Key = ASCIIEncoding.ASCII.GetBytes(key);
    aesAlg.IV = ASCIIEncoding.ASCII.GetBytes(iv);
    ICryptoTransform rijndaelEncrypt = aesAlg.CreateEncryptor();
    byte[] inputData = Encoding.Unicode.GetBytes(srctxt);
    byte[] encryptedData = rijndaelEncrypt.TransformFinalBlock(inputData, 0, inputData.Length);
    return Encoding.Unicode.GetString(encryptedData);
}
```

使用 RijndaelManaged 对文本执行解密:

```
public static string DecrypTxt(string srcen)
```

```
{
    RijndaelManaged aesAlg;
    aesAlg = new RijndaelManaged();
    string key = "abcdefghijklmnop";//16 个字符
    string iv = "abcdefghijklmnop";//16 个字符
    aesAlg.Key = ASCIIEncoding.ASCII.GetBytes(key);
    aesAlg.IV = ASCIIEncoding.ASCII.GetBytes(iv);
    ICryptoTransform rijndaelDecrypt = aesAlg.CreateDecryptor();
    byte[] inputData = Encoding.Unicode.GetBytes(srcen);
    byte[] decryptedData = rijndaelDecrypt.TransformFinalBlock(inputData, 0, inputData.Length);
    return Encoding.Unicode.GetString(decryptedData);
}
```

读入加密的文本，用于用户名与密码的较验：

```
if (File.Exists("user.txt"))
{
    u_list = new ArrayList();
    pw_list = new ArrayList();
    string astr;
    StreamReader sr = new StreamReader("user.txt");//读入用户信息
    astr = sr.ReadLine();
    while (astr != null)
    {
        astr = Data.DecrypTxt(astr);
        u_list.Add(astr.Substring(9));//username:admin
        astr = sr.ReadLine();//passwd:123456
        astr = Data.DecrypTxt(astr);
        pw_list.Add(astr.Substring(7));
        astr = sr.ReadLine();//username:admin
    }
    sr.Close();
}
```

7.3.2 非对称加密算法

本小节实现非对称加密算法对文本进行加密，密钥保存在密钥容器中。加密与解密操作的数据是字节数组，图7-1是运行 RSA 算法的界面截图。

实现加密的函数如下面代码所示：

```
private string RSAEncrypt(string text)
{
    RSACryptoServiceProvider rsa = GetRSAProviderFromContainer("rsa1");
```

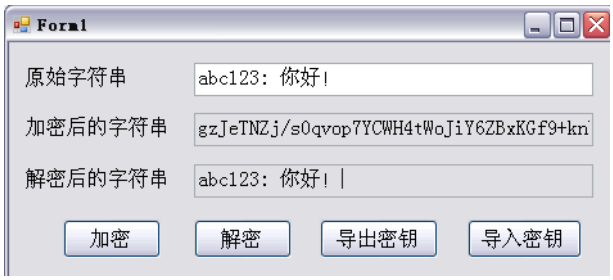

图 7-1 RSA 加密算法窗体界面

```
    byte[] bytes = Encoding.Unicode.GetBytes(text);
    byte[] encryptedData = rsa.Encrypt(bytes, true);
    return Convert.ToBase64String(encryptedData);
}
```

实现解密的函数如下面代码所示：

```
private string RSADescrpt(string text)
{
    RSACryptoServiceProvider rsa = GetRSAProviderFromContainer("rsa1");
    byte[] encryptedData = Convert.FromBase64String(text);
    byte[] decryptedData = rsa.Decrypt(encryptedData, true);
    return Encoding.Unicode.GetString(decryptedData);
}
```

将密钥信息由 XML 文本中读入，保存在密钥容器中，代码如下：

```
private static void SaveKeyInfoToContainer(string containerName)
{
    CspParameters cp = new CspParameters();
    //将 ProviderType 字段初始化为值 24，该值指定 PROV_RSA_AES 提供程序
    cp.ProviderType = 24;
    cp.KeyContainerName = containerName;
    string rsaKeyInfo = System.IO.File.ReadAllText("keyinfo.txt");
    RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);
    rsa.FromXmlString(rsaKeyInfo);
    //true 表示将密钥永久驻留在 CSP 中，false 表示从密钥容器中删除该密钥
    rsa.PersistKeyInCsp = true;
}
```

在使用非对称加密算法前，首先创建密钥容器，并用密钥容器初始化 RSA 算法类，代码如下：

```
private static RSACryptoServiceProvider GetRSAProviderFromContainer(string containerName)
{
    CspParameters cp = new CspParameters();
    //将 ProviderType 初始化为值 24，该值指定 PROV_RSA_AES 提供程序
    cp.ProviderType = 24;
```

```
    //如果不存在名为 containerName 的密钥容器，则创建之，并初始化 cp
    //如果存在，则直接根据它保存的内容初始化 cp
    cp.KeyContainerName = containerName;
    RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(cp);
    return rsa;
}
```

加密按钮事件代码：

```
textBoxEncrypted.Text = RSAEncrypt(textBoxInput.Text);
```

解密按钮作事件代码：

```
textBoxDecrypted.Text = RSADescrpt(textBoxEncrypted.Text);
```

导出密钥按钮事件代码：

```
RSACryptoServiceProvider rsa = GetRSAProviderFromContainer("rsa1");
//ToXmlString 方法导出不对称加密密钥的 xml 形式，
//方法使用参数值为 true 表示同时包含 RSA 公钥和私钥；false 表示仅包含公钥。
string rsaKeyInfo = rsa.ToXmlString(true);
System.IO.File.WriteAllText("keyinfo.txt", rsaKeyInfo);
MessageBox.Show(" 密钥信息成功导出到 keyinfo.txt 中, 请妥善保存该文件");
```

导入密钥按钮事件代码：

```
SaveKeyInfoToContainer("rsa1");
MessageBox.Show(" 导入成功");
```

7.3.3 HASH 算法与数字签名

下面是使用 SHA1Managed 类进行哈希运算的演示函数：

```
static void Main()
{
    //原始字符串"This is the original message!" 使用
    //类 SHA1Managed 哈希运算生成的值。
    byte[] SentHashValue = { 59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197,
213, 134, 130, 135 };
    //与哈希值对应的原始字符串，可看作是经过网络传输后获得的
    string MessageString = "This is the original message!";
    byte[] CompareHashValue;
    //将字符串转化为 Unicode 编码的字节数组.
    UnicodeEncoding UE = new UnicodeEncoding();
    byte[] MessageBytes = UE.GetBytes(MessageString);
    //新建 SHA1Managed 对象
    SHA1Managed SHhash = new SHA1Managed();
    //计算字节数组的哈希值.
    CompareHashValue = SHhash.ComputeHash(MessageBytes);
```

```
    //比较字节数组值是否相等.
    bool Same = true;
    for (int x = 0; x < SentHashValue.Length; x++)
    {
      if (SentHashValue[x] != CompareHashValue[x])
      {
        Same = false;
      }
    }
    //输出比较结果.
    if (Same)
    {
      Console.WriteLine("The hash codes match.");
    }
    else
    {
      Console.WriteLine("The hash codes do not match.");
    }
  }
```

7.4 实验作业

1. 采用对称加密算法实现用户名密码加密到文本文件中，并能由文件中读出进行验证。
2. 编写网络通信程序，使用加密方法对传输的数据进行加密。