

第 19 章

加密与解密技术

- » 数据的加密与解密
- » 文件的加密与解密

19.1 数据的加密与解密

实例 571

异或算法对数字进行加密与解密

光盘位置: 光盘\MR\19\571

中级

趣味指数: ★★★★★

实例说明

在实现本实例之前先来简要了解一下加密的概念,加密是指通过某种特殊的方法,更改已有信息的内容,使得未授权的用户即使得到了加密信息,如果没有正确解密的方法,也无法得到信息的内容。谈到加密的话题,一些读者一定非常感兴趣,而且会联想到复杂的加密算法,本实例主要使用异或“^”运算符简单地实现了对数字加密的功能。实例运行效果如图 19.1 所示。

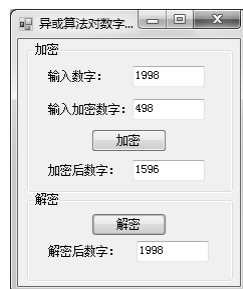



图 19.1 异或算法对数字进行加密与解密

关键技术

本实例实现时主要使用了“异或”运算符对数字进行“异或”运算,以达到简单加密数字的目的,下面对其进行详细讲解。

“异或”运算符“^”用于比较两个二进制数的相应位。在执行按位“异或”运算时,如果两个二进制数的相应位都为 1 或两个二进制数的相应位都为 0,则返回 0;如果两个二进制数的相应位其中一个为 1 一个为 0,则返回 1。

现在来了解一下使用“异或”加密或解密的执行过程,数值 23 转换为二进制为 10111,加密数字的数值 15 转换为二进制为 1111。对比两个二进制的值,从右向左按位对比,如果两个二进制数的相应位都为 1 或两个二进制数的相应位都为 0,则返回 0;如果两个二进制数的相应位中一个为 1 一个为 0,则返回 1,最后得到的结果为二进制值 11000,该值转换为十进制为 24,所以得到的加密结果为 24。而解密过程也很简单,只是将加密结果 24 与加密数字 15 进行“异或”运算,将 24 转换为二进制值 11000,将 15 转换为二进制值 1111,进行“异或”运算后,得到结果为 23,这样又还原了加密的数据。

 **说明:** 本实例只是简单地使用了“异或”运算符计算两个整型数值以达到加密的目的,所以本实例只可以对整型数值进行加密运算,并不适合其他数据的加密。

设计过程

(1) 打开 Visual Studio 2008 开发环境,新建一个 Windows 窗体应用程序,并将其命名为 Encrypt。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main,在该窗体中添加两个 GroupBox 容器控件,其中,在第一个 GroupBox 中放入 3 个 TextBox 控件和一个 Button 按钮,分别用于输入数字、输入加密数字、显示加密后的数字和计算加密信息;在第二个 GroupBox 中放入一个 TextBox 控件和一个 Button 按钮,分别用于显示解密后的信息和计算解密信息。

(3) 程序主要代码如下:

```
private void btn_Encrypt_Click(object sender, EventArgs e)
{
    int P_int_Num, P_int_Key; //定义两个值类型变量
    if (int.TryParse(txt_Num.Text, out P_int_Num) //判断输入是否是数值
        && int.TryParse(txt_Key.Text, out P_int_Key))
    {
        txt_Encrypt.Text = (P_int_Num ^ P_int_Key).ToString(); //加密数值
    }
    else

```

```

    {
        MessageBox.Show("请输入数值", "出现错误!"); //提示输入信息不正确
    }
}
private void btn_Revert_Click(object sender, EventArgs e)
{
    int P_int_Key, P_int_Encrypt; //定义两个值类型变量
    if (int.TryParse(txt_Encrypt.Text, out P_int_Key) //判断输入是否是数值
        && int.TryParse(txt_Key.Text, out P_int_Encrypt))
    {
        txt_Revert.Text = (P_int_Encrypt ^ P_int_Key).ToString(); //解密数值
    }
    else
    {
        MessageBox.Show("请输入数值", "出现错误!"); //提示输入信息不正确
    }
}
}

```

秘笈心法

心法领悟 571: 简述“异或”运算符。

本实例使用了“异或”运算符，但是在使用“异或”运算符之前，有必要了解“异或”运算符所做的“异或”运算的机制，“异或”运算符“^”用于比较两个二进制数的相应位。在执行按位“异或”运算时，如果两个二进制数的相应位都为 1 或两个二进制数的相应位都为 0，则返回 0；如果两个二进制数的相应位中一个为 1 一个为 0，则返回 1。

实例 572

使用 MD5 算法加密数据

光盘位置: 光盘\MR\19\572

中级

趣味指数: ★★★★★

实例说明

MD5 (Message-Digest Algorithm 5) 是一种被广泛使用的“消息-摘要算法”。“消息-摘要算法”实际上就是一个单向散列函数，数据块通过单向散列函数得到一个固定长度的散列值，数据块的签名就是计算数据块的散列值，MD5 算法的散列值为 128 位。本实例演示如何使用 MD5 算法对用户输入的密码进行加密，实例运行效果如图 19.2 所示。



图 19.2 使用 MD5 算法加密数据

关键技术

本实例在实现时主要用到了 MD5 类的 ComputeHash 方法，下面对其进行详细讲解。

MD5 类表示 MD5 哈希算法的所有实现均从中继承的抽象类，该类位于 System.Security.Cryptography 命名空间下，其 ComputeHash 方法有 3 种重载形式，分别介绍如下。

□ 计算指定字节数组的哈希值，语法格式如下：

```
public byte[] ComputeHash(byte[] buffer)
```

参数说明

- ① buffer: 要计算其哈希代码的输入。
- ② 返回值: 计算所得的哈希代码。

□ 计算指定 Stream 对象的哈希值，语法格式如下：

```
public byte[] ComputeHash(Stream inputStream)
```

参数说明

- ① inputStream: 要计算其哈希代码的输入。
- ② 返回值: 计算所得的哈希代码。

❑ 计算指定字节数组的指定区域的哈希值，语法格式如下：

```
public byte[] ComputeHash(byte[] buffer,int offset,int count)
```

ComputeHash 方法中的参数及说明如表 19.1 所示。

表 19.1 ComputeHash 方法中的参数及说明

参 数	说 明
buffer	要计算其哈希代码的输入
offset	字节数组中的偏移量，从该位置开始使用数据
count	数组中用作数据的字节数
返回值	计算所得的哈希代码

 说明：本实例用到了 ComputeHash 方法的第一种重载形式。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MD5Arithmetic。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来输入要加密的数据和显示加密后的字符串；添加一个 Button 控件，用来使用 MD5 算法对输入的数据进行加密。

(3) 程序主要代码如下：

```
public string Encrypt(string strPwd)
{
    MD5 md5 = new MD5CryptoServiceProvider();           //创建 MD5 对象
    byte[] data = System.Text.Encoding.Default.GetBytes(strPwd); //将字符编码为一个字节序列
    byte[] md5data = md5.ComputeHash(data);             //计算 data 字节数组的哈希值
    md5.Clear();                                       //清空 MD5 对象
    string str = "";                                   //定义一个变量，用来记录加密后的密码
    for (int i = 0; i < md5data.Length - 1; i++)       //遍历字节数组
    {
        str += md5data[i].ToString("x").PadLeft(2, '0'); //对遍历到的字节进行加密
    }
    return str;                                       //返回得到的加密字符串
}
private void button1_Click(object sender, EventArgs e)
{
    string P_str_Code = textBox1.Text;                 //记录要加密的密码
    textBox2.Text = Encrypt(P_str_Code);               //显示加密后的字符串
}
```

秘笈心法

心法领悟 572：如何判断是否为数字？

开发程序时，经常需要判断输入的字符串是否为数字，如判断输入的电话号码、货币金额和邮编等。在程序中判断是否为数字的方法有很多种，可以使用正则表达式、int.Parse 方法和 double.Parse 方法等。下面的代码通过 double.Parse 方法判断 textBox1 文本框中的输入是否为数字。

```
double.Parse(textBox1.Text);
```

实例 573

使用 ROT13 算法加密解密数据

光盘位置：光盘\MR\19\573

中级

趣味指数：★★★★☆

实例说明

文件加密可以避免造成重要信息的泄漏，复杂的加密算法可以将信息加密得非常繁杂，但是对于一般的应用，没有必要作类似于 PGP、RSA 或 DES 等复杂的加密算法。本实例介绍如何使用 ROT13 算法加密和解密数

据。实例运行效果如图 19.3 所示。

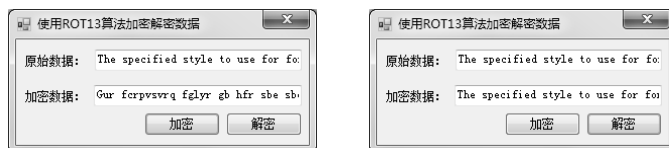


图 19.3 使用 ROT13 算法加密解密数据

关键技术

本实例实现时，主要是用 Convert 类的 ToChar 方法来获取单个字符的 Unicode 编码，然后将字母的前 13 个和后 13 个对调，从而实现加密的功能。下面对 Convert 类的 ToChar 方法进行详细讲解。

ToChar 方法返回指定的 Unicode 字符值，并且不执行任何实际的转换，其语法格式如下：

```
public static char ToChar (char value)
```

参数说明

value: 一个 Unicode 字符。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ROT13Encrypt。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示原始数据和解密后的数据；添加两个 Button 控件，分别用来实现利用 ROT13 算法加密和解密数据的功能。

(3) 程序主要代码如下：

```
public string ROT13Encode(string InputText)
{
    char tem_Character; //存储临时字符
    int UnicodeChar; //存储临时字符的字节值
    string EncodedText = ""; //存储加密或解密后的字符串
    for (int i = 0; i < InputText.Length; i++) //遍历字符串中的所有字符，只能加密字符串，无法加密汉字
    {
        tem_Character = System.Convert.ToChar(InputText.Substring(i, 1)); //获取字符串中指定的字符
        UnicodeChar = (int)tem_Character; //获取当前字符的 Unicode 编码
        if (UnicodeChar >= 97 && UnicodeChar <= 109) //对字符进行加密
        {
            UnicodeChar = UnicodeChar + 13;
        }
        else if (UnicodeChar >= 110 && UnicodeChar <= 122) //对字符进行解密
        {
            UnicodeChar = UnicodeChar - 13;
        }
        else if (UnicodeChar >= 65 && UnicodeChar <= 77) //对字符进行加密
        {
            UnicodeChar = UnicodeChar + 13;
        }
        else if (UnicodeChar >= 78 && UnicodeChar <= 90) //对字符进行解密
        {
            UnicodeChar = UnicodeChar - 13;
        }
        EncodedText = EncodedText + (char)UnicodeChar; //得到加密或解密字符串
    }
    return EncodedText; //返回加密或解密后的字符串
}
```

秘笈心法

心法领悟 573: 如何在字符串中查找指定字符?

在字符串中查找指定字符时，可以先将字符串显示在 richTextBox 控件中，然后利用 richTextBox 类的 Find 方法在该控件中查找指定字符。在字符串中查找指定字符的代码如下：

```

M_int_index = richTextBox1.Find(textBox1.Text.Trim(), M_int_index, RichTextBoxFinds.MatchCase);
if (M_int_index == -1)
{
    MessageBox.Show("没有要查找的字符串", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    M_int_index = 0;
}
else
    M_int_index = M_int_index + textBox1.Text.Trim().Length;
richTextBox1.Focus();

```

实例 574

使用恺撒密码算法加密密码

光盘位置: 光盘\MR\19\574

中级

趣味指数: ★★★★★

实例说明

恺撒密码据传是古罗马恺撒大帝用来保护重要军情的加密系统，它是一种置换密码，通过将字母顺序推后起到加密作用。例如，将字母顺序推后 3 位，字母 A 将被推作为字母 D，字母 B 将被推作字母 E。本实例使用 C# 实现了恺撒加密的算法，实例运行效果如图 19.4 所示。

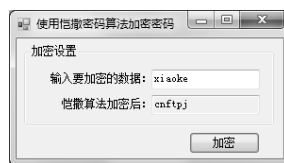


图 19.4 使用恺撒密码算法加密密码

关键技术

本实例实现时主要用到了 `string` 类的 `ToCharArray` 方法和 `Convert` 类的 `ToChar` 方法，下面分别对它们进行详细介绍。

(1) `string` 类的 `ToCharArray` 方法

`string` 类的 `ToCharArray` 方法用来将字符串中的字符复制到 `Unicode` 字符数组，该方法有两种重载形式，本实例中用到的它的重载形式如下：

```
public char[] ToCharArray()
```

参数说明

返回值：元素为此字符串的各字符的 `Unicode` 字符数组。如果此字符串是空字符串，则返回的数组为空且长度为零。

(2) `Convert` 类的 `ToChar` 方法

`Convert` 类的 `ToChar` 方法用来将指定的值转换为 `Unicode` 字符，该方法为可重载方法，本实例中用到的它的重载形式如下：

```
public static char ToChar(int value)
```

参数说明

- ❶ `value`: 32 位有符号整数。
- ❷ 返回值：等效于 `value` 的值的 `Unicode` 字符。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 `CaesarArithmetic`。

(2) 更改默认窗体 `Form1` 的 `Name` 属性为 `Frm_Main`，在该窗体中添加两个 `TextBox` 控件，分别用来输入要加密的数据和显示加密后的字符串；添加一个 `Button` 控件，用来使用恺撒密码算法对输入的数据进行加密。

(3) 程序主要代码如下：

```

public int AscII(string str) //获取字符的 ASCII 码
{
    byte[] array = new byte[1]; //创建字节数组
    array = System.Text.Encoding.ASCII.GetBytes(str); //为字节数组赋值
    int ascicode = (short)(array[0]); //获取字节数组的第一项
    return ascicode; //返回字节数组的第一项
}

```

```

public string Caesar(string str) //凯撒加密算法的实现
{
    char[] c = str.ToCharArray(); //创建字符数组
    string strCaesar = ""; //定义一个变量, 用来存储加密后的字符串
    for (int i = 0; i < str.Length; i++) //遍历字符串中的每一个字符
    {
        string ins = c[i].ToString(); //记录遍历到的字符
        string outs = ""; //定义一个变量, 用来记录加密后的字符串
        bool isChar = "0123456789abcdefghijklmnopqrstuvwxyz".Contains(ins.ToLower()); //判断指定的字符串中是否包含遍历到的字符
        bool isToUpperChar = isChar && (ins.ToUpper() == ins); //判断遍历到的字符是否是大写
        ins = ins.ToLower(); //将遍历到的字符转换为小写
        if (isChar) //判断指定的字符串中是否包含遍历到的字符
        {
            int offset = (Ascii(ins) + 5 - Ascii("a")) % (Ascii("z") - Ascii("a") + 1); //获取字符的 ASCII 码
            outs = Convert.ToChar(offset + Ascii("a")).ToString(); //转换为字符并记录
            if (isToUpperChar) //判断是否大写
            {
                outs = outs.ToUpper(); //全部转换为大写
            }
        }
        else
        {
            outs = ins; //记录遍历的字符
        }
        strCaesar += outs; //添加到加密字符串中
    }
    return strCaesar; //返回加密后的字符串
}

```

秘笈心法

心法领悟 574: 如何将新字符串添加到已有字符串中?

将新字符串添加到已有字符串中时, 可以先声明一个 `StringBuilder` 类对象, 以指定已有字符串的长度可变, 然后利用该对象的 `Append` 方法在字符串中添加指定字符串。将新字符串添加到已有字符串的代码如下:

```

StringBuilder strbuilder = new StringBuilder(textBox1.Text.Trim());
strbuilder.Append(textBox2.Text.Trim());
textBox3.Text = strbuilder.ToString();

```

实例 575

对数据报进行加密保障通信安全

光盘位置: 光盘\MR\19\575

高级

趣味指数: ★★★★★

实例说明

网络传输数据时, 有时候传输信息容易被不法分子截获而用作其他用途。这样, 如果传输的数据中包含有重要秘密, 将会造成非常严重的后果。为了防止这种情况的发生, 可以对网络中传输的数据进行加密, 用户接收到数据后再进行解密查看, 这样可以更好地保障网络通信安全。运行本实例, 首先设置端口号, 然后在窗体左下方的文本框中输入聊天信息, 单击“发送”按钮, 向局域网中发送聊天信息, 同时在右侧的“数据传输信息”栏中显示数据报的发送、接收及丢失情况。实例运行效果如图 19.5 所示。

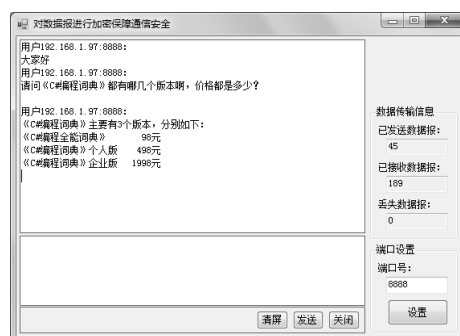


图 19.5 对数据报进行加密保障通信安全

关键技术

本实例获取数据报信息时主要用到 `IPGlobalProperties` 和 `UdpStatistics` 类, 而在对数据报加密时用到 `DESCrypto`

ServiceProvider 和 CryptoStream 类，其中 DESCryptoServiceProvider 继承于 DES 类。下面对本实例中用到的关键技术进行详细讲解。

(1) IPGlobalProperties 类

IPGlobalProperties 类提供有关本地计算机的网络连接的信息，本实例中用到它的 GetIPGlobalProperties 和 GetUdpIPv4Statistics 方法，下面分别进行介绍。

GetIPGlobalProperties 为静态方法，主要用来获取一个对象，该对象提供有关本地计算机的网络连接和通信统计数据的信息，其语法格式如下：

```
public static IPGlobalProperties GetIPGlobalProperties()
```

参数说明

返回值：IPGlobalProperties 对象，该对象包含有关本地计算机的信息。

GetUdpIPv4Statistics 方法主要用来提供本地计算机的用户数据报协议/Internet 协议版本 4（UDP/IPv4）统计数据，其语法格式如下：

```
public abstract UdpStatistics GetUdpIPv4Statistics()
```

参数说明

返回值：UdpStatistics 对象，提供本地计算机的 UDP/IPv4 通信统计数据。

例如，本实例中创建 IPGlobalProperties 对象，及调用其 GetUdpIPv4Statistics 方法创建 UdpStatistics 对象的代码如下：


```
IPGlobalProperties NetInfo = IPGlobalProperties.GetIPGlobalProperties();
UdpStatistics myUdpStat = null;
myUdpStat = NetInfo.GetUdpIPv4Statistics();
```

(2) UdpStatistics 类

UdpStatistics 类提供用户数据报协议（UDP）统计数据，本实例中主要用到其 DatagramsSent 属性、DatagramsReceived 属性和 IncomingDatagramsDiscarded 属性，其中，DatagramsSent 属性用来获取已发送的用户数据报协议（UDP）数据报的数量，DatagramsReceived 属性用来获取已接收的用户数据报协议（UDP）数据报的数量，IncomingDatagramsDiscarded 属性用来获取已收到但因端口错误而丢弃的用户数据报协议（UDP）数据报的数量。

例如，本实例中初始化已发送、已接收和丢失数据报的实现代码如下：

```
SendNum1 = Int32.Parse(myUdpStat.DatagramsSent.ToString()); //记录发送的数据报
ReceiveNum1 = Int32.Parse(myUdpStat.DatagramsReceived.ToString()); //记录接收的数据报
DisNum1 = Int32.Parse(myUdpStat.IncomingDatagramsDiscarded.ToString()); //记录丢失的数据报
```

 说明：IPGlobalProperties 类和 UdpStatistics 类位于 System.Net.NetworkInformation 命名空间下。

(3) DES 类

DES 类表示所有 DES 实现都必须从中派生的数据加密标准（DES）算法的基类，其 CreateEncryptor 方法和 CreateDecryptor 方法分别用来加密和解密。

CreateEncryptor 方法使用指定的 Key 属性和初始化向量（IV）创建对称加密器对象，其语法格式如下：

```
public abstract ICryptoTransform CreateEncryptor(byte[] rgbKey,byte[] rgbIV)
```

参数说明

- ❶ rgbKey：用于对称算法的密钥。
- ❷ rgbIV：用于对称算法的初始化向量。
- ❸ 返回值：对称加密器对象。

CreateDecryptor 方法使用指定的 Key 属性和初始化向量（IV）创建对称解密器对象，其语法格式如下：

```
public abstract ICryptoTransform CreateDecryptor(byte[] rgbKey,byte[] rgbIV)
```

参数说明

- ❶ rgbKey：用于对称算法的密钥。
- ❷ rgbIV：用于对称算法的初始化向量。
- ❸ 返回值：对称解密器对象。

(4) CryptoStream 类

CryptoStream 类定义将数据流链接到加密转换的流, 其构造函数的语法格式如下:

```
public CryptoStream(Stream stream, ICryptoTransform transform, CryptoStreamMode mode)
```

参数说明

- ❶ stream: 对其执行加密转换的流。
- ❷ transform: 要对流执行的加密转换。
- ❸ mode: CryptoStreamMode 枚举值之一, CryptoStreamMode 枚举值及说明如表 19.2 所示。

表 19.2 CryptoStreamMode 枚举值及说明


枚 举 值	说 明
Read	对加密流的读访问
Write	对加密流的写访问

另外, 在向加密或解密流中写入数据时用到 CryptoStream 类的 Write 方法, 该方法将一个字节序列写入当前 CryptoStream, 并将流中的当前位置提升写入的字节数, 其语法格式如下:

```
public override void Write(byte[] buffer, int offset, int count)
```

参数说明

- ❶ buffer: 字节数组, 此方法将 count 个字节从 buffer 复制到当前流。
- ❷ offset: buffer 中的字节偏移量, 从此偏移量开始将字节复制到当前流。
- ❸ count: 要写入当前流的字节数。

 说明: DES 类和 CryptoStream 类位于 System.Security.Cryptography 命名空间下。

设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 EncryptDataReport。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加两个 RichTextBox 控件, 分别用来输入聊天信息和显示聊天信息; 添加 4 个 TextBox 控件, 分别用来输入端口号和显示已发送数据报、已接收数据报、丢失数据报; 添加 4 个 Button 控件, 分别用来执行设置端口号、发送聊天信息、清空聊天信息和关闭应用程序操作。

(3) 程序主要代码如下。

Frm_Main 窗体的后台代码中, 首先创建程序所需要的 .NET 对象及公共变量, 代码如下:

```
#region 定义全局对象及变量
private IPEndPoint Server;           //服务器端
private IPEndPoint Client;         //客户端
private Socket mySocket;           //套接字
private EndPoint ClientIP;         //IP 地址
byte[] buffer, data;               //接收缓存
bool blFlag = true;                //标识是否第一次发送信息
bool ISPort = false;               //判断端口打开
int SendNum1, ReceiveNum1, DisNum1; //记录窗体加载时的已发送\已接收\丢失的数据报
int SendNum2, ReceiveNum2, DisNum2; //记录当前已发送\已接收\丢失的数据报
int SendNum3, ReceiveNum3, DisNum3; //缓存已发送\已接收\丢失的数据报
int port;                           //端口号
#endregion
```

Frm_Main 窗体加载时, 初始化已发送、已接收和丢失的数据报, 并使用全局变量记录, 实现代码如下:

```
//初始化已发送、已接收和丢失的数据报
private void Form1_Load(object sender, EventArgs e)
{
    if (blFlag == true)
    {
        IPGlobalProperties NetInfo = IPGlobalProperties.GetIPGlobalProperties(); //创建一个 IPGlobalProperties 对象
        UdpStatistics myUdpStat = null; //声明 UdpStatistics 对象
        myUdpStat = NetInfo.GetUdpIPv4Statistics(); //创建 UdpStatistics 对象
    }
}
```

```

        SendNum1 = Int32.Parse(myUdpStat.DatagramsSent.ToString()); //记录发送的数据报
        ReceiveNum1 = Int32.Parse(myUdpStat.DatagramsReceived.ToString()); //记录接收的数据报
        DisNum1 = Int32.Parse(myUdpStat.IncomingDatagramsDiscarded.ToString()); //记录丢失的数据报
    }
}

```

单击“设置”按钮，使用指定的端口号连接服务器端与客户端，并开始接收消息。“设置”按钮的 Click 事件的代码如下：

```

private void button4_Click(object sender, EventArgs e) //设置端口号
{
    try
    {
        port = Convert.ToInt32(textBox4.Text); //记录端口号
        CheckForIllegalCrossThreadCalls = false; //指定线程中可以调用窗体的控件对象
        buffer = new byte[1024];
        data = new byte[1024];
        Server = new IPEndPoint(IPAddress.Any, port); //创建服务器端
        Client = new IPEndPoint(IPAddress.Broadcast, port); //创建客户端
        ClientIP = (EndPoint)Server; //获取服务器端 IP 地址
        //创建 Socket 对象
        mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
        //设置 Socket 网络操作
        mySocket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.Broadcast, 1);
        mySocket.Bind(Server); //绑定服务器端
        //开始接收消息
        mySocket.BeginReceiveFrom(buffer, 0, buffer.Length, SocketFlags.None, ref ClientIP, new AsyncCallback(StartListener), null);
        ISPort = true; //打开指定端口号
    }
    catch {}
}

```

单击“发送”按钮，首先判断是否有打开的端口，如果没有，弹出提示信息，否则根据发送和接收的消息计算已发送、已接收和丢失的数据报，并显示在相应的文本框中，然后使用 DES 对要发送的消息进行加密发送。“发送”按钮的 Click 事件的代码如下：

```

//发送信息
private void button2_Click(object sender, EventArgs e)
{
    if (ISPort == true) //判断是否有打开的端口号
    {
        IPGlobalProperties NetInfo = IPGlobalProperties.GetIPGlobalProperties();
        UdpStatistics myUdpStat = null;
        myUdpStat = NetInfo.GetUdpIPv4Statistics();
        try
        {
            if (blFlag == false) //非第一次发送
            {
                SendNum2 = Int32.Parse(myUdpStat.DatagramsSent.ToString());
                ReceiveNum2 = Int32.Parse(myUdpStat.DatagramsReceived.ToString());
                DisNum2 = Int32.Parse(myUdpStat.IncomingDatagramsDiscarded.ToString());
                textBox1.Text = Convert.ToString(SendNum2 - SendNum3);
                textBox2.Text = Convert.ToString(ReceiveNum2 - ReceiveNum3);
                textBox3.Text = Convert.ToString(DisNum2 - DisNum3);
            }
            SendNum2 = Int32.Parse(myUdpStat.DatagramsSent.ToString());
            ReceiveNum2 = Int32.Parse(myUdpStat.DatagramsReceived.ToString());
            DisNum2 = Int32.Parse(myUdpStat.IncomingDatagramsDiscarded.ToString());
            SendNum3 = SendNum2; //记录本次的发送数据报
            ReceiveNum3 = ReceiveNum2; //记录本次的接收数据报
            DisNum3 = DisNum2; //记录本次的丢失数据报
            if (blFlag == true) //第一次发送
            {
                textBox1.Text = Convert.ToString(SendNum2 - SendNum1);
                textBox2.Text = Convert.ToString(ReceiveNum2 - ReceiveNum1);
                textBox3.Text = Convert.ToString(DisNum2 - DisNum1);
                blFlag = false;
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    string str = EncryptDES(rtbSend.Text, "mrsoftxk"); //加密要发送的信息
    data = Encoding.Unicode.GetBytes(str);
    mySocket.SendTo(data, data.Length, SocketFlags.None, Client); //发送消息
    rtbSend.Text = "";
}
else
{
    MessageBox.Show("请首先打开端口!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    button4.Focus();
}
}
}

```

上面的代码中用到了 `EncryptDES` 方法, 该方法为自定义的、返回值类型为 `string` 的方法, 主要用来使用 DES 加密数据报, 它有两个 `string` 类型的参数, 分别用来表示待加密的字符串和加密密钥, 返回值为加密后的字符串。EncryptDES 方法的实现代码如下:

```

#region DES 加密字符串
///<summary>
///DES 加密字符串
///</summary>
///<param name="str">待加密的字符串</param>
///<param name="key">加密密钥, 要求为 8 位</param>
///<returns>加密成功返回加密后的字符串, 失败返回源字符串</returns>
public string EncryptDES(string str, string key)
{
    try
    {
        byte[] rgbKey = Encoding.UTF8.GetBytes(key.Substring(0, 8)); //将加密密钥转换为字节数组
        byte[] rgbIV = Keys; //记录原始密钥数组
        byte[] inputByteArray = Encoding.UTF8.GetBytes(str); //将加密字符串转换为字节数组
        DESCryptoServiceProvider myDES = new DESCryptoServiceProvider(); //创建加密对象
        MemoryStream MStream = new MemoryStream(); //创建内存数据流
        //创建加密流对象
        CryptoStream CStream = new CryptoStream(MStream, myDES.CreateEncryptor(rgbKey, rgbIV), CryptoStreamMode.Write);
        CStream.Write(inputByteArray, 0, inputByteArray.Length); //向加密流中写入数据
        CStream.FlushFinalBlock(); //释放加密流对象
        return Convert.ToBase64String(MStream.ToArray()); //返回内存流中的数据
    }
    catch
    {
        return str;
    }
}
#endregion

```

秘笈心法

心法领悟 575: 如何根据标点符号分行?

根据标点符号分行时, 首先要使用 `string` 类的 `Split` 方法分割字符串, 然后再通过 “\n” 回车换行符将分割的字符串换行显示。根据标点符号分行的代码如下:

```

string oldstr = textBox1.Text.Trim();
string[] newstr = oldstr.Split('. ');
for (int i = 0; i < newstr.Length; i++)
{
    if (richTextBox1.Text == "")
        richTextBox1.Text = newstr[i].ToString();
    else
        richTextBox1.Text += "\n" + newstr[i].ToString();
}

```

实例 576

使用 one-time pad 算法加密数据

高级

光盘位置: 光盘\MR\19\576

趣味指数: ★★★★★

实例说明

在密码学里, 有一种理想的加密方案, 叫做一次一密乱码本, 即 one-time pad 算法, 该算法是最安全的加密算法, 双方一旦安全交换了密钥, 之后交换信息的过程就可以保证绝对安全。本实例使用 C# 实现了 one-time pad 加密算法, 实例运行效果如图 19.6 所示。

注意: 程序中使用 one-time pad 算法时, 一定要保证密钥和密文的长度是一样的。

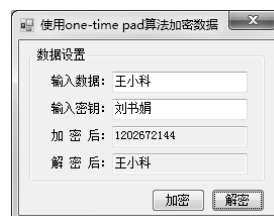


图 19.6 使用 one-time pad 算法加密数据

关键技术

本实例在实现 one-time pad 加密算法时, 主要用到了 Encoding 类的 GetBytes 方法和 GetString 方法, 下面分别对它们进行详细介绍。

(1) Encoding 类的 GetBytes 方法

Encoding 类表示字符编码, 其 GetBytes 方法主要用来将一组字符编码为一个字节序列, 该方法为可重载方法, 本实例中用到的它的重载形式如下:

```
public virtual byte[] GetBytes(string s)
```

参数说明

- ❶ s: 字符串。
- ❷ 返回值: 一个字节数组, 包含对指定的字符集进行编码的结果。

说明: Encoding 类位于 System.Text 命名空间下。

(2) Encoding 类的 GetString 方法

Encoding 类的 GetString 方法主要用来将一个字节序列解码为一个字符串, 该方法为可重载方法, 本实例中用到的它的重载形式如下:

```
public virtual string GetString(byte[] bytes)
```

参数说明

- ❶ bytes: 包含要解码的字节序列的字节数组。
- ❷ 返回值: 包含指定字节序列解码结果的字符串。

设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 OneTimePadArithmetic。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加 4 个 TextBox 控件, 分别用来输入要加密的数据和密钥, 以及显示加密后的数据和解密后的数据; 添加两个 Button 控件, 分别用来实现使用 one-time pad 算法加密数据和解密数据的功能。

(3) 程序主要代码如下。

在 Frm_Main 窗体中输入要加密的数据和密钥后, 单击“加密”按钮, 使用 one-time pad 算法对输入的数据进行加密, 实现代码如下:

```
private void button1_Click(object sender, EventArgs e)
{
    textBox2.Text = ""; //清空文本框
    Encoding encoding = Encoding.Default; //获取字符编码
    byte[] btData = encoding.GetBytes(textBox1.Text); //将要加密的数据转换为字节数组
    byte[] btKey = encoding.GetBytes(textBox4.Text); //将密钥转换为字节数组
}
```

```

if (btData.Length == btKey.Length) //判断长度是否相等
{
    byte[] btEncrypt = Encrypt(btData, btKey); //加密数据
    for (int i = 0; i < btEncrypt.Length; i++) //遍历加密后的字节数组
    {
        textBox2.Text += btEncrypt[i]; //显示在文本框中
    }
}
}

```

上面的代码中用到了 `Encrypt` 方法，该方法为自定义的、返回值类型为 `byte[]` 的方法，主要用来对指定的数据使用 `one-time pad` 算法进行加密。`Encrypt` 方法的实现代码如下：

```

public static byte[] Encrypt(byte[] btData, byte[] btKey)
{
    if (btKey.Length != btData.Length) //判断长度是否相等
    {
        MessageBox.Show("请确保要加密数据的长度与密钥的长度一致！");
    }
    byte[] btResult = new byte[btData.Length]; //声明一个字节数组，用来存储加密数据
    for (int i = 0; i < btResult.Length; ++i) //遍历字节数组
    {
        btResult[i] = (byte)(btKey[i] ^ btData[i]); //为字节数组赋值
    }
    return btResult; //返回得到的加密数据
}

```

单击“解密”按钮，调用 `Encrypt` 方法对加密过的数据进行逆向加密，并返回一个 `byte[]` 数组，然后使用 `Encoding` 类的 `GetString` 方法从该数组中获取解密字符串。“解密”按钮的 `Click` 事件的代码如下：

```

private void button2_Click(object sender, EventArgs e)
{
    Encoding encoding = Encoding.Default; //获取字符编码
    byte[] btData = encoding.GetBytes(textBox1.Text); //将要加密的数据转换为字节数组
    byte[] btKey = encoding.GetBytes(textBox4.Text); //将密钥转换为字节数组
    if (btData.Length == btKey.Length) //判断长度是否相等
    {
        byte[] btDecrypt = Encrypt(Encrypt(btData, btKey), btKey); //解密数据
        textBox3.Text = encoding.GetString(btDecrypt); //将解密后的字节数组转换为字符串并显示
    }
}

```

秘笈心法

心法领悟 576：如何在字符串中添加多个空格？

开发程序时，有时会根据需要在字符串中添加一些空格，这时可以使用 `string` 类的 `Insert` 方法，该方法可以在字符串中的指定位置插入一个新的字符串（包括空格）。在字符串中添加空格的代码如下：

```
textBox3.Text = textBox1.Text.Insert(Convert.ToInt32(textBox2.Text.Trim()), " ");
```

实例 577

使用伪随机数加密技术加密用户登录密码

光盘位置：光盘\MR\19\577

高级

趣味指数：★★★★★

实例说明

为了保障用户登录密码的安全，本实例使用伪随机数技术对用户的登录密码进行加密，运行本实例，当用户在“登录密码”文本框中输入登录密码时，程序会自动将使用过伪随机数加密技术加密过的登录密码显示在下面的“加密密码”文本框中，单击“登录”按钮，程序对“加密密码”文本框中的加密数据进行解密，然后再与用户输入的登录密码相比较，如果相同，则登录成功；否则，登录失败。实例运行效果如图 19.7 所示。



图 19.7 使用伪随机数加密技术加密用户登录密码

关键技术

本实例对用户登录密码加密时用到伪随机数加密技术，伪随机数加密技术实质上就是通过伪随机数序列使登录密码字符串的字节值发生变化而产生密文，由于相同的初值能得到相同的随机数序列，因此，可以采用同样的伪随机数序列来对密文进行解密。产生伪随机数时主要用到 `Random` 类，该类表示伪随机数生成器，它是一种能够产生满足某些随机性统计要求的数字序列的设备，其 `Next` 方法用来返回随机数，语法格式如下：

```
public virtual int Next(int maxValue)
```

参数说明

- ❶ `maxValue`：要生成的随机数的上界（随机数不能取该上界值），`maxValue` 必须大于等于零。
- ❷ 返回值：大于等于零且小于 `maxValue` 的 32 位带符号整数，即返回值的范围通常包括零但不包括 `maxValue`；不过，如果 `maxValue` 等于零，则返回 `maxValue`。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 `PRanDataEncrypt`。

(2) 更改默认窗体 `Form1` 的 `Name` 属性为 `Frm_Main`，在该窗体中添加 3 个 `TextBox` 控件，分别用来输入登录用户、登录密码和显示加密密码；添加两个 `Button` 控件，分别用来执行用户登录和清空文本框操作。

(3) 程序主要代码如下。

`Frm_Main` 窗体的后台代码中，首先定义加密用户密码所用的伪随机数，代码如下：

```
//定义加密用户密码所用的伪随机数
```

```
private string randStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz";
```

当在“登录密码”文本框中输入登录密码时，实时将使用伪随机数加密过的登录密码显示在“加密密码”文本框中，实现代码如下：

```
private void textBox2_TextChanged(object sender, EventArgs e)
```

```
{
```

```
    textBox3.Text = EncryptPwd(textBox2.Text);
```

```
//显示加密后的用户登录密码
```

```
}
```

上面的代码中用到 `EncryptPwd` 方法，该方法为自定义的、返回值类型为 `string` 的方法，主要用来使用伪随机数技术加密用户登录密码，它有一个参数，用来表示用户登录密码。`EncryptPwd` 方法的实现代码如下：

```
/// <summary>
```

```
/// 使用伪随机数加密用户登录密码
```

```
/// </summary>
```

```
/// <param name="str">用户登录密码</param>
```

```
/// <returns>加密后的用户登录密码</returns>
```

```
private string EncryptPwd(string str)
```

```
{
```

```
    byte[] btData = Encoding.Default.GetBytes(str);
```

```
//将登录密码转换为字节数组
```

```
    int j, k, m;
```

```
    int len = randStr.Length;
```

```
//记录伪随机数长度
```

```
    StringBuilder sb = new StringBuilder();
```

```
//创建 StringBuilder 对象
```

```
    Random rand = new Random();
```

```
//创建 Random 对象
```

```
    for (int i = 0; i < btData.Length; i++)
```

```
    {
```

```
        j = (byte)rand.Next(6);
```

```
//产生伪随机数
```

```
        btData[i] = (byte)((int)btData[i] ^ j);
```

```
//使用伪随机数对密码字节数组进行移位
```

```
        k = (int)btData[i] % len;
```

```
        m = (int)btData[i] / len;
```

```
        m = m * 8 + j;
```

```
        sb.Append(randStr.Substring(k, 1) + randStr.Substring(m, 1));
```

```
//组合加密字符串
```

```
    }
```

```
    return sb.ToString();
```

```
//返回生成的加密字符串
```

```
}
```

单击“登录”按钮，判断“加密密码”文本框是否为空。如果不为空，调用 `DecryptPwd` 方法解密“加密密码”文本框中的字符串；然后使用解密后的字符串与“登录密码”文本框中的字符串相比较，如果相同，则用户登录成功；否则，弹出提示信息。“登录”按钮的 `Click` 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox3.Text != "")
    {
        if (DecryptPwd(textBox3.Text) == textBox2.Text) //对加密过的登录密码进行解密
            MessageBox.Show("用户登录成功!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        else
            MessageBox.Show("用户密码错误!", "错误", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

上面的代码中用到了 `DecryptPwd` 方法, 该方法为自定义的、返回值类型为 `string` 的方法, 主要用来解密用户登录密码, 它有一个参数, 主要用来表示经过加密的用户登录密码。 `DecryptPwd` 方法的实现代码如下:

```
/// <summary>
/// 解密用户登录密码
/// </summary>
/// <param name="str">经过加密的用户登录密码</param>
/// <returns>解密后的用户登录密码</returns>
private string DecryptPwd(string str)
{
    try
    {
        int j, k, m, n = 0;
        int len = randStr.Length;
        byte[] btData = new byte[str.Length / 2]; //获取伪随机数长度
        for (int i = 0; i < str.Length; i += 2) //定义一个字节数组, 并指定长度
        { //对登录密码进行解密
            k = randStr.IndexOf(str[i]);
            m = randStr.IndexOf(str[i + 1]);
            j = m / 8;
            m = m - j * 8;
            btData[n] = (byte)(j * len + k);
            btData[n] = (byte)((int)btData[n] ^ m);
            n++;
        }
        return Encoding.Default.GetString(btData); //返回解密后的登录密码
    }
    catch { return ""; }
}
```

秘笈心法

心法领悟 577: 如何将字符串颠倒输出?

颠倒输出字符串时, 可以先将要输出的字符串保存到一个 `char` 类型的数组中, 然后使用 `Array` 类的 `Reverse` 方法。将字符串颠倒输出的代码如下:

```
string str1 = textBox1.Text.Trim();
char[] charstr = str1.ToCharArray();
Array.Reverse(charstr);
string str2 = new string(charstr);
textBox2.Text = str2;
```

实例 578

以 XML 格式导入导出密钥

光盘位置: 光盘\MR\19\578

高级

趣味指数: ★★★★★

实例说明

本实例主要实现以 XML 格式导入导出密钥, 从而实现对数据进行加密和解密的功能。运行本实例, 首先在窗体中显示生成的公钥和私钥, 然后输入明文数据, 单击“加密”按钮, 对输入的明文数据进行加密; 单击“解密”按钮, 对加密后的数据进行解密。实例运行效果如图 19.8 所示。



图 19.8 以 XML 格式导入导出密钥

关键技术

本实例实现时主要用到了 `RSACryptoServiceProvider` 类的 `ToXmlString` 方法、`Encrypt` 方法和 `Decrypt` 方法，下面对本实例中用到的关键技术进行详细讲解。

(1) `RSACryptoServiceProvider` 类的 `ToXmlString` 方法

`RSACryptoServiceProvider` 类用来使用加密服务提供程序 (CSP) 提供的 RSA 算法的实现执行不对称加密和解密，其 `ToXmlString` 方法主要用来创建并返回包含当前 RSA 对象的密钥的 XML 字符串，该方法的语法格式如下：

```
public override string ToXmlString(bool includePrivateParameters)
```

参数说明

- ❶ `includePrivateParameters`: `true` 表示同时包含 RSA 公钥和私钥，`false` 表示仅包含公钥。
- ❷ 返回值：包含当前 RSA 对象的密钥的 XML 字符串。

说明：`RSACryptoServiceProvider` 类位于 `System.Security.Cryptography` 命名空间下。

(2) `RSACryptoServiceProvider` 类的 `Encrypt` 方法

该方法主要使用 RSA 算法对数据进行加密，其语法格式如下：

```
public byte[] Encrypt(byte[] rgb,bool fOAEP)
```

参数说明

- ❶ `rgb`: 要加密的数据。
- ❷ `fOAEP`: 如果为 `true`，则使用 OAEP 填充（仅在运行 Microsoft Windows XP 或更高版本的计算机上可用）执行直接的 RSA 加密；如果为 `false`，则使用 PKCS#1.5 版填充。
- ❸ 返回值：字节数组，表示已加密的数据。

(3) `RSACryptoServiceProvider` 类的 `Decrypt` 方法

该方法主要使用 RSA 算法对数据进行解密，其语法格式如下：

```
public byte[] Decrypt(byte[] rgb,bool fOAEP)
```

参数说明

- ❶ `rgb`: 要解密的数据。
- ❷ `fOAEP`: 如果为 `true`，则使用 OAEP 填充（仅在运行 Microsoft Windows XP 或更高版本的计算机上可用）执行直接的 RSA 解密；如果为 `false`，则使用 PKCS#1.5 版填充。
- ❸ 返回值：字节数组，表示已解密的数据，它是加密前的原始纯文本。

设计过程

- (1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 `KeyToXML`。
- (2) 更改默认窗体 `Form1` 的 `Name` 属性为 `Frm_Main`，在该窗体中添加 5 个 `TextBox` 控件，分别用来显示

公钥、显示私钥、输入明文数据、显示加密后的数据和显示解密后的数据；添加两个 Button 控件，分别用来执行数据加密和解密操作。

(3) 程序主要代码如下。

在 Frm_Main 窗体的后台代码中，首先创建 RSACryptoServiceProvider 对象，并且定义一个字节数组，用来存储临时数据，代码如下：

```
RSACryptoServiceProvider RSACrypto = new RSACryptoServiceProvider(); //创建 RSA 算法加密解密对象
byte[] M_bt_Data; //定义一个字节数组，用来存储临时数据
```

Frm_Main 窗体加载时，在文本框中显示程序自动生成的公钥和私钥数据，代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.textBox1.Text = RSACrypto.ToXmlString(true); //显示生成的公钥
    this.textBox2.Text = RSACrypto.ToXmlString(false); //显示生成的私钥
}
```

当用户输入明文数据之后，单击“加密”按钮，调用 RSACryptoServiceProvider 类的 Encrypt 方法对数据进行加密，并且使用 Encoding 类的 UTF8 编码方式的 GetString 方法得到加密后的数据，显示在文本框中。“加密”按钮的 Click 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox3.Text != "") //判断是否输入了要加密的数据
    {
        byte[] P_bt_Encrypt = Encoding.UTF8.GetBytes(textBox3.Text); //将要加密的数据转换为字节数组
        M_bt_Data = RSACrypto.Encrypt(P_bt_Encrypt, false); //加密数据
        textBox4.Text = Encoding.UTF8.GetString(M_bt_Data); //显示加密数据
    }
}
```

单击“解密”按钮，调用 RSACryptoServiceProvider 类的 Decrypt 方法对加密过的数据进行解密，并且使用 Encoding 类的 UTF8 编码方式的 GetString 方法得到解密后的数据，显示在文本框中。“解密”按钮的 Click 事件代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (textBox4.Text != "") //判断是否有加密过的数据
    {
        byte[] P_bt_Decrypt = RSACrypto.Decrypt(M_bt_Data, false); //对数据进行解密
        textBox5.Text = Encoding.UTF8.GetString(P_bt_Decrypt); //显示解密数据
    }
}
```

秘笈心法

心法领悟 578：如何判断字符串是否为日期格式？

判断字符串是否为日期格式时，可以使用正则表达式。验证日期格式的正则表达式主要有以下 3 种：

```
\b(?:<year>\d{2,4})/(?:<month>\d{1,2})/(?:<day>\d{1,2})\b
```

或

```
\b(?:<year>\d{2,4})-(?:<month>\d{1,2})-(?:<day>\d{1,2})\b
```

或

```
\b(?:<year>\d{2,4})年(?:<month>\d{1,2})月(?:<day>\d{1,2})日\b
```

实例 579

以参数格式导入导出密钥

光盘位置：光盘\MR\19\579

高级

趣味指数：★★★★★

实例说明

本实例主要实现以参数格式导入导出密钥，从而实现对数据进行加密和解密的功能。运行本实例，在窗体

中输入明文数据，单击“加密”按钮，对输入的明文数据进行加密；单击“解密”按钮，对加密后的数据进行解密。实例运行效果如图 19.9 所示。

关键技术

本实例实现时主要用到了 RSACryptoServiceProvider 类的 ExportParameters 方法、ImportParameters 方法、Encrypt 方法和 Decrypt 方法，下面对本实例中用到的关键技术进行详细讲解。

(1) RSACryptoServiceProvider 类的 ExportParameters 方法

该方法主要用来导出 RSAPParameters 标准参数，其语法格式如下：

```
public override RSAPParameters ExportParameters(bool includePrivateParameters)
```

参数说明

- ❶ includePrivateParameters: 如果要包括私有参数，则为 true；否则为 false。
- ❷ 返回值: RSA 算法的标准参数。


(2) RSACryptoServiceProvider 类的 ImportParameters 方法

该方法主要用来导入指定的 RSAPParameters 标准参数，其语法格式如下：

```
public override void ImportParameters(RSAPParameters parameters)
```

参数说明

parameters: RSA 算法的标准参数。

 **说明:** 关于 RSACryptoServiceProvider 类的 Encrypt 方法和 Decrypt 方法的详细讲解，请参见实例 578 中的关键技术。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 KeyToParameter。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 TextBox 控件，分别用来输入明文数据、显示加密后的数据和解密后的数据；添加两个 Button 控件，分别用来执行数据加密和解密操作。

(3) 程序主要代码如下。

Frm_Main 窗体的后台代码中，首先创建 RSACryptoServiceProvider 对象和 RSAPParameters 标准参数对象，并且定义一个字节数组，用来存储临时数据，代码如下：

```
RSACryptoServiceProvider RSACrypto;           //声明 RSA 算法加密解密对象
RSAPParameters RSAParam;                       //声明 RSAPParameters 参数对象
byte[] M_bt_Data;                             //定义一个字节数组，用来存储临时数据
```

在 Frm_Main 窗体的构造函数中，调用 RSACryptoServiceProvider 类的 ImportParameters 方法导入 RSAPParameters 标准参数，实现代码如下：

```
public Frm_Main()
{
    InitializeComponent();
    RSACrypto = new RSACryptoServiceProvider(); //初始化 RSA 算法加密解密对象
    RSAParam = RSACrypto.ExportParameters(true); //初始化 RSAPParameters 参数
    RSACrypto.Clear();                          //清空 RSACryptoServiceProvider 对象
    RSACrypto = new RSACryptoServiceProvider(); //初始化 RSA 算法加密解密对象
    RSACrypto.ImportParameters(RSAParam);       //导入密钥
}
```

当用户输入明文数据之后，单击“加密”按钮，调用 RSACryptoServiceProvider 类的 Encrypt 方法对数据进行加密，并且使用 Encoding 类的 UTF8 编码方式的 GetString 方法得到加密后的数据，显示在文本框中。“加密”按钮的 Click 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "") //判断是否输入了要加密的数据
    {
```



图 19.9 以参数格式导入导出密钥

```

byte[] P_bt_Encrypt = Encoding.UTF8.GetBytes(textBox1.Text);           //将要加密的数据转换为字节数组
M_bt_Data = RSACrypto.Encrypt(P_bt_Encrypt, false);                   //加密数据
textBox2.Text = Encoding.UTF8.GetString(M_bt_Data);                   //显示加密数据
}

```

单击“解密”按钮，调用 `RSACryptoServiceProvider` 类的 `Decrypt` 方法对加密过的数据进行解密，并且使用 `Encoding` 类的 `UTF8` 编码方式的 `GetString` 方法得到解密后的数据，显示在文本框中。“解密”按钮的 `Click` 事件代码如下：

```

private void button2_Click(object sender, EventArgs e)
{
    if (textBox2.Text != "")                                           //判断是否有加密过的数据
    {
        byte[] P_bt_Decrypt = RSACrypto.Decrypt(M_bt_Data, false);    //对数据进行解密
        textBox3.Text = Encoding.UTF8.GetString(P_bt_Decrypt);       //显示解密数据
    }
}

```

秘笈心法

心法领悟 579：巧截字符串中的数字。

截取字符串中的数字时，可以先使用 `CharEnumerator` 对象的 `MoveNext` 方法循环访问字符串中的每个字符，并将字符用 `System.Text.Encoding` 类中 `ASCII` 编码方式的 `GetBytes` 方法进行编码，然后判断经过编码之后的字符的 `ASCII` 码值是否介于 48 和 57 之间，如果是，则将其显示在 `textBox` 文本框中。截取字符串中数字的代码如下：

```

CharEnumerator CEnumerator = textBox1.Text.GetEnumerator();
while (CEnumerator.MoveNext())
{
    byte[] array = new byte[1];
    array = System.Text.Encoding.ASCII.GetBytes(CEnumerator.Current.ToString());
    int ascicode = (short)(array[0]);
    if (ascicode >= 48 && ascicode <= 57)
    {
        textBox2.Text += CEnumerator.Current.ToString();
    }
}

```

19.2 文件的加密与解密

实例 580

文本文件加密与解密

光盘位置：光盘\MR\19\580

高级

趣味指数：★★★★★

实例说明

在本实例的窗体中，首先选择要加密或解密的文本文件，然后单击“加密”或“解密”按钮对文本文件进行加密或解密。实例运行效果如图 19.10 所示。

关键技术

本实例实现时主要用到了 `System.Security.Cryptography` 命名空间下的 `RijndaelManaged` 类的 `CreateDecryptor` 方法、`CreateEncryptor` 方法和 `CryptoStream` 类的 `Write` 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）`RijndaelManaged` 类

该类是访问 `System.Security.Cryptography.Rijndael` 对称加密算法的托管版本，其语法格式如下：

```
public sealed class RijndaelManaged : Rijndael
```

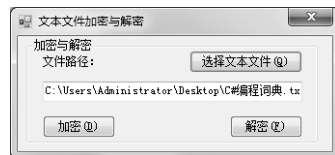



图 19.10 文本文件加密与解密

 **注意：**此算法支持 128、192 或 256 位的密钥长度。

(2) CreateDecryptor 方法

该方法位于 RijndaelManaged 类中，使用指定的 Key 和初始化向量 (IV) 创建对称的 Rijndael 解密器对象，其语法格式如下：

```
public override ICryptoTransform CreateDecryptor (byte[] rgbKey,byte[] rgbIV)
```

参数说明

- ❶ rgbKey: 用于对称算法的机密密钥。
- ❷ rgbIV: 用于对称算法的 IV。
- ❸ 返回值: 对称的 Rijndael 解密器对象。


(3) CreateEncryptor 方法

该方法位于 RijndaelManaged 类中，使用指定的 Key 和初始化向量 (IV) 创建对称的 Rijndael 加密器对象，其语法格式如下：

```
public override ICryptoTransform CreateEncryptor (byte[] rgbKey,byte[] rgbIV)
```

参数说明

- ❶ rgbKey: 用于对称算法的机密密钥。
- ❷ rgbIV: 用于对称算法的 IV。
- ❸ 返回值: 对称的 Rijndael 加密器对象。

 **说明：**关于 CryptoStream 类的 Write 方法的详细讲解，请参见实例 575 中的关键技术。

设计过程

(1) 打开 Visual Studio 2008 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EncryptTextFileOne。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示文本文件路径；添加一个 OpenFileDialog 控件，用来选择要加密或解密的文本文件；添加 3 个 Button 控件，用来执行选择文本文件、加密和解密操作。

(3) 程序主要代码如下。

单击“加密”按钮实现对选择的文本文件进行加密，“加密”按钮的 Click 事件的代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "") //若未选择要加密的文本文件
    { MessageBox.Show("请选择要加密的文件"); } //如果没有选择则弹出提示
    else
    {
        try{
            string strPath = textBox1.Text; //加密文件的路径
            int intLent=strPath.LastIndexOf("\\")+1; //设置截取的起始位置
            int intLong = strPath.Length; //设置截取的长度
            string strName = strPath.Substring(intLent,intLong-intLent); //要加密的文件名称
            int intTxt = strName.LastIndexOf("."); //设置截取的起始位置
            int intTextLeng = strName.Length; //设置截取的长度
            string strTxt = strName.Substring(intTxt,intTextLeng-intTxt); //取出文件的扩展名
            strName = strName.Substring(0,intTxt);
            //加密后的文件名及路径
            string strOutName = strPath.Substring(0, strPath.LastIndexOf("\\")+ 1) + strName + "Out" + strTxt;
            //加密文件密钥
            byte[] key = { 24, 55, 102, 24, 98, 26, 67, 29, 84, 19, 37, 118, 104, 85, 121, 27, 93, 86, 24, 55, 102, 24, 98, 26, 67, 29, 9, 2, 49, 69, 73, 92 };
            byte[] IV = { 22, 56, 82, 77, 84, 31, 74, 24, 55, 102, 24, 98, 26, 67, 29, 99 };
            RijndaelManaged myRijndael = new RijndaelManaged();
            FileStream fsOut = File.Open(strOutName, FileMode.Create, FileAccess.Write);
            FileStream fsIn = File.Open(strPath, FileMode.Open, FileAccess.Read);
            //写入加密文本文件
            CryptoStream csDecrypt = new CryptoStream(fsOut, myRijndael.CreateEncryptor(key, IV), CryptoStreamMode.Write);
            BinaryReader br = new BinaryReader(fsIn); //创建阅读器来读加密文本
            csDecrypt.Write(br.ReadBytes((int)fsIn.Length), 0, (int)fsIn.Length); //将数据写入加密文本
        }
    }
}
```

```

csDecrypt.FlushFinalBlock();
csDecrypt.Close(); //关闭 CryptoStream 对象
fsIn.Close(); //关闭 FileStream 对象
fsOut.Close(); //关闭 FileStream 对象
if (MessageBox.Show("加密成功!加密后的文件名及路径为: \n" + strOutName + ",是否删除源文件", "信息提示", MessageBoxButtons.YesNo) == DialogResult.Yes)
{
    File.Delete(strPath); //删除指定文件
    textBox1.Text = ""; //清空文本框
}
else
{
    textBox1.Text = "";
}
}
catch (Exception ee) //如果出现异常
{
    MessageBox.Show(ee.Message); //输出异常信息
}
}
}

```

单击“解密”按钮实现对加密的文本文件进行解密，“解密”按钮的 Click 事件代码如下：

```

private void button3_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "") //若未选择要解密的文件
    {
        MessageBox.Show("请选择要解密的文件路径"); //如果没有选择则弹出提示
    }
    else
    {
        string strPath = textBox1.Text; //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1; //设置截取字符串的起始位置
        int intLong = strPath.Length; //设置截取长度
        string strName = strPath.Substring(intLent, intLong - intLent); //要加密的文件名称
        int intTxt = strName.LastIndexOf("."); //截取字符串的起始位置
        int intTextLeng = strName.Length; //截取长度
        strName = strName.Substring(0, intTxt); //获取扩展名
        if (strName.LastIndexOf("Out") != -1)
        {
            strName = strName.Substring(0, strName.LastIndexOf("Out"));
        }
        else
        {
            strName = strName + "In";
        }
        //加密后的文件名及路径
        string strInName = strPath.Substring(0, strPath.LastIndexOf("\\") + 1) + strName + ".txt";
        //解密文件密钥
        byte[] key = { 24, 55, 102, 24, 98, 26, 67, 29, 84, 19, 37, 118, 104, 85, 121, 27, 93, 86, 24, 55, 102, 24, 98, 26, 67, 29, 9, 2, 49, 69, 73, 92 };
        byte[] IV = { 22, 56, 82, 77, 84, 31, 74, 24, 55, 102, 24, 98, 26, 67, 29, 99 };
        RijndaelManaged myRijndael = new RijndaelManaged(); //创建 RijndaelManaged 对象
        //创建 FileStream 对象
        FileStream fsOut = File.Open(strPath, FileMode.Open, FileAccess.Read);
        CryptoStream csDecrypt = new CryptoStream(fsOut, myRijndael.CreateDecryptor(key, IV), CryptoStreamMode.Read);
        StreamReader sr = new StreamReader(csDecrypt); //把文件读出来
        StreamWriter sw = new StreamWriter(strInName); //解密后写入一个新文件
        sw.Write(sr.ReadToEnd());
        sw.Flush();
        sw.Close();
        sr.Close();
        fsOut.Close();
        if (MessageBox.Show("解密成功!解密后的文件名及路径为: "+strInName+", 是否删除源文件", "信息提示", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            File.Delete(strPath); //删除指定文件
            textBox1.Text = ""; //清空文本框
        }
        else
        {

```

```

        textBox1.Text = "";
    }
}
}

```

秘笈心法

心法领悟 580：如何存储变长字符串？

在程序中存储变长字符串时，需要使用 `StringBuilder` 对象。相对于 `string` 对象来说，`StringBuilder` 对象是可变的，不用生成中间对象，因此，在连接的字符串较多或字符串长度较长时，通常都使用 `StringBuilder` 对象。

实例 581

利用图片加密文件

光盘位置：光盘\MR\19\581

高级

趣味指数：★★★★★

实例说明

本实例在加密时，使用指定的图片生成加密密钥，然后对文本文件进行加密；在解密时，使用加密时的图片生成解密密钥，然后对加密的文本文件进行解密。运行本实例，首先打开一张图片，用来生成加密或解密的密钥，然后选择要加密或解密的文本文件，最后单击“加密”或“解密”按钮，实现对文本文件的加密或解密。实例运行效果如图 19.11 所示。



图 19.11 利用图片加密文件

关键技术

本实例实现时主要用到了 `RC2CryptoServiceProvider` 类、`BinaryWriter` 类的 `Write` 方法、`File` 类的 `Delete` 方法和 `Copy` 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）`RC2CryptoServiceProvider` 类

该类定义访问 RC2 算法的加密服务提供程序（CSP）实现的包装对象，无法继承此类。

（2）`BinaryWriter` 类

该类以二进制形式将基元类型写入流，并支持用特定的编码写入字符串，其构造器的语法格式如下：

```
public BinaryWriter (Stream output)
```

参数说明

`output`：表示输出流。

（3）`BinaryWriter` 类的 `Write` 方法

该方法将一个无符号字节写入当前流，并将流的位置提升一个字节，其语法格式如下：

```
public virtual void Write (byte value)
```

参数说明

`value`：表示要写入的无符号字节。

（4）`File` 类的 `Delete` 方法

`File` 类提供用于创建、复制、删除、移动和打开文件的静态方法，并协助创建 `FileStream` 对象，该类是个静态类，其 `Delete` 方法用于删除指定的文件，如果指定的文件不存在，则引发异常。该方法的语法格式如下：

```
public static void Delete (string path)
```

参数说明

`path`：表示要删除的文件的名称。

（5）`File` 类的 `Copy` 方法

该方法将现有文件复制到新文件，不允许改写同名的文件，其语法格式如下：

```
public static void Copy (string sourceFileName, string destFileName)
```

参数说明

- ❶ sourceFileName: 要复制的文件。
- ❷ destFileName: 目标文件的名称, 不能是一个目录或现有文件。

设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 EncryptTextFileTwo。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加一个 TextBox 控件, 用来显示加密或解密文件的路径; 添加一个 OpenFileDialog 控件, 用来选择要加密或解密的文件和打开密钥的图片; 添加 4 个 Button 控件, 分别用来执行加密、解密、打开文件和打开图片操作; 添加一个 PictureBox 控件, 用于显示密钥图片。

(3) 程序主要代码如下。

单击“加密”按钮, 实现利用图片对文本文件进行加密的功能, “加密”按钮的 Click 事件的代码如下:

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        if (pictureBox1.ImageLocation == null) //判断是否选择了图片
        { MessageBox.Show("请选择一幅图片用于加密"); return; } //如果没有选择则弹出提示
        if (textBox1.Text == "") //若未选择需要加密的文件
        { MessageBox.Show("请选择加密文件路径"); return; } //如果没有选择则弹出提示
        //图片流
        FileStream fsPic = new FileStream(pictureBox1.ImageLocation, FileMode.Open, FileAccess.Read);
        //加密文件流
        FileStream fsText = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);
        //初始化对称算法的密钥和向量
        byte[] byKey = new byte[16]; //定义存储密钥的字节数组
        byte[] byIv = new byte[8]; //定义存储向量的字节数组
        fsPic.Read(byKey, 0, 16); //把图片流写入密钥缓冲区
        fsPic.Read(byIv, 0, 8); //把图片流写入向量缓冲区
        //临时加密文件
        string strPath = textBox1.Text; //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1;
        int intLong = strPath.Length;
        string strName = strPath.Substring(intLent, intLong - intLent); //要加密的文件名称
        string strLinPath = "C:\\\" + strName; //临时加密文件路径
        FileStream fsOut = File.Open(strLinPath, FileMode.Create, FileAccess.Write);
        //开始加密, 首先创建 RC2CryptoServiceProvider 对象
        RC2CryptoServiceProvider desc = new RC2CryptoServiceProvider();
        BinaryReader br = new BinaryReader(fsText); //创建 BinaryReader 对象
        //创建 CryptoStream 对象, 用于写入临时加密文件
        CryptoStream cs = new CryptoStream(fsOut, desc.CreateEncryptor(byKey, byIv), CryptoStreamMode.Write);
        cs.Write(br.ReadBytes((int)fsText.Length), 0, (int)fsText.Length); //写入加密流
        cs.FlushFinalBlock();
        cs.Flush();
        cs.Close();
        fsPic.Close();
        fsText.Close();
        fsOut.Close();
        File.Delete(textBox1.Text.TrimEnd()); //删除原文件
        File.Copy(strLinPath, textBox1.Text); //复制加密文件
        File.Delete(strLinPath); //删除临时文件
        MessageBox.Show("加密成功");
        pictureBox1.ImageLocation = null;
        textBox1.Text = "";
    }
    catch (Exception ee)
    {
        MessageBox.Show(ee.Message);
    }
}
```

单击“解密”按钮，实现利用图片对加密的文本文件进行解密的功能，“解密”按钮的 Click 事件的代码如下：

```
private void button4_Click(object sender, EventArgs e)
{
    try
    {
        //图片流
        FileStream fsPic = new FileStream(pictureBox1.ImageLocation, FileMode.Open, FileAccess.Read);
        //解密文件流
        FileStream fsOut = File.Open(textBox1.Text, FileMode.Open, FileAccess.Read);
        //初始化对称算法的密钥和向量
        byte[] bykey = new byte[16]; //定义存储密钥的字节数组
        byte[] bylv = new byte[8]; //定义存储向量的字节数组
        fsPic.Read(bykey, 0, 16); //把图片流写入密钥缓冲区
        fsPic.Read(bylv, 0, 8); //把图片流写入向量缓冲区
        //创建临时解密文件
        string strPath = textBox1.Text; //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1; //获取不含文件名的路径长度
        int intLong = strPath.Length; //获取含文件名的路径长度
        //获取要解密文件的名称，即加密文件的名称
        string strName = strPath.Substring(intLent, intLong - intLent);
        string strLinPath = "C:\\\" + strName; //临时解密文件路径
        FileStream fs = new FileStream(strLinPath, FileMode.Create, FileAccess.Write);
        //开始解密，首先创建 RC2CryptoServiceProvider 对象
        RC2CryptoServiceProvider desc = new RC2CryptoServiceProvider();
        //创建 CryptoStream 对象，用于读取加密文件
        CryptoStream csDecrypt = new CryptoStream(fsOut, desc.CreateDecryptor(bykey, bylv), CryptoStreamMode.Read);
        BinaryReader sr = new BinaryReader(csDecrypt); //创建 BinaryReader 对象
        BinaryWriter sw = new BinaryWriter(fs); //创建 BinaryWriter 对象
        sw.Write(sr.ReadBytes(Convert.ToInt32(fsOut.Length))); //写入解密流
        sw.Flush();
        sw.Close();
        sr.Close();
        fs.Close();
        fsOut.Close();
        fsPic.Close();
        csDecrypt.Flush();
        File.Delete(textBox1.Text.TrimEnd()); //删除原文件
        File.Copy(strLinPath, textBox1.Text); //复制加密文件
        File.Delete(strLinPath); //删除临时文件
        MessageBox.Show("解密成功"); //弹出提示信息
        pictureBox1.ImageLocation = null; //清空图片
        textBox1.Text = ""; //清空文本框
    }
    catch (Exception ee) //如果出现异常
    {
        MessageBox.Show(ee.Message); //输出异常
    }
}
}
```

秘笈心法

心法领悟 581：如何去除字符串尾空格？

去除字符串尾空格需要使用 `string` 类的 `Trim` 方法，该方法用来从字符串的开始和末尾处移除空白字符的所有匹配项。例如，下面的代码用来去掉 `textBox1` 文本框中字符串的尾空格，并将结果显示在 `textBox2` 文本框中：

```
textBox2.Text = textBox1.Text.Trim();
```

实例 582

对文件进行加密保护

光盘位置：光盘\MR\19\582

高级

趣味指数：★★★★★

实例说明

随着计算机的普及，文件的安全越来越重要，本实例使用 C# 制作了一个对文件进行加密保护的实例。运行

本实例,选择要加密或解密的文件,用程序来判断是否是加密过的文件,如果不是,输入加密密码,单击“加密”按钮,加密已选择的文件;如果是,输入解密密码,单击“解密”按钮,解密选择的加密文件。实例运行效果如图 19.12 所示。

关键技术

本实例制作对文件进行加密保护程序时,首先选择要加密或解密的文件,并输入加密或解密密码,然后启动一个新的线程,使用输入的密码对指定的文件进行加密或解密操作。另外,如果对文件执行的是加密操作,则加密成功后删除原文件。具体实现过程中,主要用到了 DES 类的 CreateEncryptor 和 CreateDecryptor 方法、CryptoStream 类的构造函数及其 Write 方法。

 **说明:** 关于 DES 类的 CreateEncryptor 方法和 CreateDecryptor 方法、CryptoStream 类的构造函数及其 Write 方法的详细讲解,请参见实例 575 中的关键技术。

设计过程

(1) 打开 Visual Studio 2008 开发环境,新建一个 Windows 窗体应用程序,并将其命名为 ProtectFile。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main,在该窗体中添加一个 OpenFileDialog 控件,用来显示“打开”对话框;添加两个 TextBox 控件,分别用来显示选择的文件路径和输入加密、解密密码;添加 3 个 Button 控件,分别用来执行选择加密或解密的文件、加密文件和解密文件操作;添加一个 ProgressBar 控件,用来显示加密或解密的进度。

(3) 程序主要代码如下。

Frm_Main 窗体加载时,首先将加密文件菜单写入到注册表中,然后判断系统中是否有打开的加密文件,如果有,则将“解密”按钮设置为可用,实现代码如下:

```
private void Frm1_Load(object sender, EventArgs e)
{
    FileMenu(Application.ExecutablePath + ",0", Application.ExecutablePath); //在注册表中创建加密文件菜单项
    string[] str = Environment.GetCommandLineArgs(); //记录系统中的打开项
    try
    {
        string strFile = ""; //定义一个字符串,用来记录文件名
        for (int i = 2; i < str.Length; i++)
            strFile += str[i]; //为文件名赋值
        FileInfo FInfo = new FileInfo(strFile); //创建 FileInfo 对象
        if (FInfo.Extension.ToLower() == ".mr") //判断文件的扩展名
        {
            textBox1.Text = strFile; //显示文件名
            button2.Enabled = false;
            button3.Enabled = true;
        }
    }
    catch {}
}
```

向注册表中写入加密文件菜单时用到 FileMenu 方法,该方法为自定义的、无返回值类型的方法,主要用来向注册表中写入加密文件菜单,它有两个 string 类型的参数,分别用来表示加密程序环境变量及加密程序路径。

FileMenu 方法的实现代码如下:

```
public static void FileMenu(string strPath, string strName) //创建快捷菜单
{
    try
    {
        Registry.ClassesRoot.CreateSubKey(".mr"); //在注册表中创建加密文件子键
        RegistryKey RKey1 = Registry.ClassesRoot.OpenSubKey(".mr", true); //打开新创建的子键
        RKey1.SetValue("", "mrfile"); //为新创建的子键赋值
    }
}
```

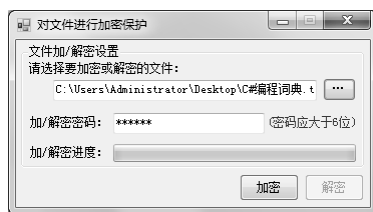


图 19.12 对文件进行加密保护

```

RKey1.Close(); //关闭子键
Registry.ClassesRoot.CreateSubKey("mrfile"); //创建 mrfile 子键
RegistryKey RKey2 = Registry.ClassesRoot.OpenSubKey("mrfile", true); //打开 mrfile 子键
RKey2.CreateSubKey("DefaultIcon"); //创建 DefaultIcon 子键
RKey2.CreateSubKey("shell"); //创建 shell 子键
RKey2.Close(); //关闭子键
RegistryKey RKey3 = Registry.ClassesRoot.OpenSubKey("mrfile\\DefaultIcon", true); //打开子键
RKey3.SetValue("", strPath); //为子键设置值
RKey3.Close(); //关闭子键
RegistryKey RKey4 = Registry.ClassesRoot.OpenSubKey("mrfile\\shell", true); //打开子键
RKey4.CreateSubKey("解密"); //创建子键
RKey4.Close(); //关闭子键
RegistryKey RKey5 = Registry.ClassesRoot.OpenSubKey("mrfile\\shell\\解密", true); //打开子键
RKey5.CreateSubKey("command"); //创建子键
RKey5.Close(); //关闭子键
RegistryKey RKey6 = Registry.ClassesRoot.OpenSubKey("mrfile\\shell\\解密\\command", true); //打开子键
RKey6.SetValue("", strName + "\\F %1"); //为子键设置值
RKey6.Close(); //关闭子键
}
catch
{
}
}

```

 **说明：**对注册表进行操作时，需要添加 Microsoft.Win32 命名空间，下面遇到类似情况时将不再提示。

单击“加密”按钮，首先判断是否选定要加密的文件，如果已经选定，则判断用户密码位数，这里要求密码必须大于 6 位，如果条件符合，则对选择的文件进行加密。“加密”按钮的 Click 事件的代码如下：

```

private void button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "") //判断加密文件是否为空
    {
        if (textBox2.Text.Length < 6) //判断加密密码是否小于 6 位
            MessageBox.Show("密码不能小于 6 位！", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        else
        {
            EDncrypt myEDncrypt = new EDncrypt(textBox1.Text, textBox2.Text, progressBar1); //创建加/解密对象
            myEDncrypt.StartEncrypt(); //启动加密线程
            progressBar1.Value = 0; //初始化加密进度条值
        }
    }
}

```

上面的代码中用到了 EDncrypt 类构造函数，该类定义了对文件进行加密和解密的方法，其构造函数用来初始化全局对象及变量，主要代码如下：

```

#region 定义全局变量及对象
private string strFile = ""; //旧文件
private string strNewFile = ""; //新文件
private string strPwd = ""; //加/解密密码
private ProgressBar PBar = null; //声明进度条对象
private Thread EThread = null; //声明加密线程对象
private Thread DThread = null; //声明解密线程对象
#endregion
//含参数的构造函数，用来初始化全局变量及对象
public EDncrypt(string name, string pwd, ProgressBar pb)
{
    strFile = name; //为旧文件赋值
    strPwd = pwd; //为加/解密密码赋值
    PBar = pb; //创建进度条对象
    EThread = new Thread(new ThreadStart(this.myEThread)); //创建加密线程对象
    DThread = new Thread(new ThreadStart(this.myDThread)); //创建解密线程对象
}

```

对文件进行加密时需要运行加密线程，加密线程的代码如下：

```

public void StartEncrypt()
{

```

```
EThread.Start(); //运行加密线程
}
```

加密线程中用到了 myEThread 方法, 该方法为自定义的、无返回值类型的方法, 它主要用来根据用户输入的加密密码对指定文件进行加密。myEThread 方法的实现代码如下:

```
private void myEThread() //文件加密
{
    byte[] btRKey = new byte[0]; //定义一个字节数组, 用来记录加密密码
    if (strPwd.Length == 6) //判断密码位数
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[0], (byte)strPwd[1] }; //记录加密密码
    }
    if (strPwd.Length == 7)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[0] };
    }
    if (strPwd.Length >= 8)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[7] };
    }
    FileStream FStream = new FileStream(strFile, FileMode.Open, FileAccess.Read); //创建 FileStream 对象
    //创建一个 FileStream 对象, 用来创建加密文件
    FileStream NewFStream = new FileStream(strFile + ".mr", FileMode.OpenOrCreate, FileAccess.Write);
    NewFStream.SetLength((long)0); //设置加密文件初始长度为 0
    byte[] buffer = new byte[0x400]; //定义一个字节数组
    int MinNum = 0; //设置进度条最小值
    long length = FStream.Length; //记录文件长度
    int MaxNum = (int)(length / ((long)0x400)); //计算进度条最大值
    PBar.Maximum = MaxNum; //设置进度条的最大值
    DES myDES = new DESCryptoServiceProvider(); //创建 DES 加密对象
    //创建 CryptoStream 加密流对象
    CryptoStream CStream = new CryptoStream(NewFStream, myDES.CreateEncryptor(btRKey, btRKey), CryptoStreamMode.Write);
    while (MinNum < length)
    {
        int count = FStream.Read(buffer, 0, 0x400); //对要加密的文件进行流读取
        CStream.Write(buffer, 0, count); //向加密流中写入数据
        MinNum += count; //记录已经读取的数据位置
        try
        {
            if (PBar.Value < PBar.Maximum)
            {
                PBar.Value++; //增加进度条的值
            }
        }
        catch
        {
        }
    }
    CStream.Close();
    NewFStream.Close();
    FStream.Close();
    File.Delete(strFile); //删除旧文件
    MessageBox.Show("文件加密成功!", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
}
```

单击“解密”按钮, 首先判断是否选定要解密的文件, 如果已经选定, 则判断解密密码位数, 这里要求密码必须大于 6 位, 如果条件符合, 则对选择的加密文件进行解密。“解密”按钮的 Click 事件的代码如下:

```
private void button3_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "") //判断解密文件是否为空
    {
        if (textBox2.Text.Length < 6) //判断解密密码是否小于 6 位
            MessageBox.Show("密码不能小于 6 位!", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        else
    }
}
```

```

    {
        EDncrypt myEDncrypt = new EDncrypt(textBox1.Text, textBox2.Text, progressBar1); //创建加/解密对象
        myEDncrypt.StartDncrypt(); //启动解密线程
        progressBar1.Value = 0; //初始化解密进度条值
    }
}

```

对文件进行解密时需要运行解密线程，解密线程的代码如下：

```

public void StartDncrypt()
{
    DThread.Start(); //运行解密线程
}

```

解密线程中用到了 myDThread 方法，该方法为自定义的、无返回值类型的方法，它主要用来根据用户输入的解密密码对指定的加密文件进行解密。myDThread 方法的实现代码如下：

```

private void myDThread() //文件解密
{
    FileStream FStream = null; //声明 FileStream 对象，用来记录旧文件
    FileStream NewFStream = null; //声明 FileStream 对象，用来记录新文件
    CryptoStream CStream = null; //声明 CryptoStream 对象，用来解密文件
    try
    {
        try
        {
            byte[] btRKey = new byte[0]; //定义一个字节数组，用来记录解密密码
            if (strPwd.Length == 6) //判断密码位数
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[0], (byte)strPwd[1] }; //记录解密密码
            }
            if (strPwd.Length == 7)
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[0] };
            }
            if (strPwd.Length >= 8)
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[7] };
            }
            FStream = new FileStream(strFile, FileMode.Open, FileAccess.Read); //创建 FileStream 对象
            strNewFile = strFile.Substring(0, strFile.Length - 3); //获取新文件名
            NewFStream = new FileStream(strNewFile, FileMode.OpenOrCreate, FileAccess.Write); //创建解密文件
            NewFStream.SetLength((long)0); //设置解密文件初始长度为 0
            byte[] buffer = new byte[0x400]; //定义一个字节数组
            int MinNum = 0; //设置进度条最小值
            long length = FStream.Length; //记录文件长度
            int MaxNum = (int)(length / ((long)0x400)); //计算进度条最大值
            PBar.Maximum = MaxNum; //设置进度条的最大值
            DES myDES = new DESCryptoServiceProvider(); //创建 DES 加/解密对象
            //创建 CryptoStream 加/解密流对象
            CStream = new CryptoStream(NewFStream, myDES.CreateDecryptor(btRKey, btRKey), CryptoStreamMode.Write);
            while (MinNum < length)
            {
                int count = FStream.Read(buffer, 0, 0x400); //对要解密的文件进行流读取
                CStream.Write(buffer, 0, count); //向解密流中写入数据
                MinNum += count; //记录已经读取的数据位置
                try
                {
                    if (PBar.Value < PBar.Maximum)
                    {
                        PBar.Value++; //增加进度条的值
                    }
                }
                catch
                {
                }
            }
        }
    }
}

```

```

    }
    }
    MessageBox.Show("文件解密成功!", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch
{
    MessageBox.Show("文件解密失败!", "错误", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
finally
{
    CStream.Close();
    FStream.Close();
    NewFStream.Close();
}
}
}

```

秘笈心法

心法领悟 582: 如何去掉字符串中的所有空格?

由于空格的 ASCII 码值是 32, 因此在去掉字符串中所有的空格时, 只需循环访问字符串中的所有字符, 并判断它们的 ASCII 码值是否为 32 即可。去掉字符串中所有空格的代码如下:

```

CharEnumerator CEnumerator = textBox1.Text.GetEnumerator();
while (CEnumerator.MoveNext())
{
    byte[] array = new byte[1];
    array = System.Text.Encoding.ASCII.GetBytes(CEnumerator.Current.ToString());
    int asciiicode = (short)(array[0]);
    if (asciiicode != 32)
    {
        textBox2.Text += CEnumerator.Current.ToString();
    }
}
}

```

实例 583

使用口令加密可执行文件

光盘位置: 光盘\MR\19\583

高级

趣味指数: ★★★★★

实例说明

EXE 可执行文件的加密是软件加密技术的一个重要部分, 本实例讲解如何在 C# 中使用口令加密 EXE 可执行文件。运行本实例, 如果用户选择的是 EXE 文件, 则输入加密口令, 单击“加密”按钮, 加密选择的 EXE 文件; 如果用户选择的是 mrexec 文件, 则输入加密时用的口令, 单击“打开”按钮, 打开加密过的 EXE 文件。实例运行效果如图 19.13 所示。

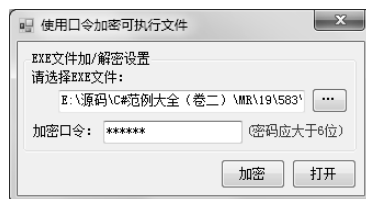


图 19.13 使用口令加密可执行文件

关键技术

使用口令加密可执行文件, 顾名思义就是在打开加密过的可执行文件时, 需要先输入口令, 然后才能打开, 这时就需要将使用口令加密过的可执行文件的菜单项写入到注册表中, 以便打开使用口令加密过的可执行文件时直接弹出“输入口令”对话框。将使用口令加密过的可执行文件的菜单项写入到注册表中时用到 RegistryKey 类的 CreateSubKey 和 SetValue 方法, 下面分别对它们进行详细讲解。

(1) CreateSubKey 方法


RegistryKey 类表示 Windows 注册表中的项级节点, 它位于 Microsoft.Win32 命名空间下, 其 CreateSubKey 方法主要用来创建一个新子项或打开一个现有子项以进行写访问, 语法格式如下:

```
public RegistryKey CreateSubKey(string subkey)
```

参数说明

❶ subkey: 要创建或打开的子项的名称或路径。

❷ 返回值: RegistryKey 对象, 表示新建的子项或 NULL (如果操作失败)。如果为 subkey 指定了零长度字符串, 则返回当前的 RegistryKey 对象。

 技巧: 要获取 RegistryKey 的实例, 可使用 Registry 类的静态成员之一。

(2) SetValue 方法

SetValue 方法用来设置注册表项中的名称/值对的值, 其语法格式如下:

```
public void SetValue(string name, Object value)
```

参数说明

❶ name: 要存储的值的名称。

❷ value: 要存储的数据。

设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 并将其命名为 RevolveJPG。

(2) 更改默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加一个 OpenFileDialog 控件, 用来显示“打开”对话框; 添加两个 TextBox 控件, 分别用来显示选择的 EXE 文件路径和输入加密、解密密码; 添加 3 个 Button 控件, 分别用来执行选择加密或解密的 EXE 文件、加密 EXE 文件和打开加密后的 EXE 文件操作。

(3) 程序主要代码如下。

Frm_Main 窗体加载时, 首先将 EXE 加密文件菜单写入到注册表中, 然后判断系统中是否有打开的 EXE 加密文件, 如果有, 则将其显示到“请选择 EXE 文件”文本框中, 实现代码如下:

```
private void Frm1_Load(object sender, EventArgs e)
{
    FileMenu(Application.ExecutablePath + ",0", Application.ExecutablePath); //在注册表中创建 EXE 加密文件菜单项
    string[] str = Environment.GetCommandLineArgs(); //记录系统中的打开项
    try
    {
        string strFile = ""; //定义一个字符串, 用来记录文件名
        for (int i = 2; i < str.Length; i++)
            strFile += str[i]; //为文件名赋值
        FileInfo FInfo = new FileInfo(strFile); //创建 FileInfo 对象
        if (FInfo.Extension.ToLower() == ".mrexe")
            textBox1.Text = strFile; //显示文件名
    }
    catch {}
}
```

向注册表中写入 EXE 加密文件菜单时用到 FileMenu 方法, 该方法为自定义的、无返回值类型的方法, 主要用来向注册表中写入 EXE 加密文件菜单, 它有两个 string 类型的参数, 分别用来表示 EXE 加密程序环境变量及 EXE 加密程序路径。FileMenu 方法的实现代码如下:

```
public static void FileMenu(string strPath, string strName) //创建快捷菜单
{
    try
    {
        Registry.ClassesRoot.CreateSubKey(".mrexe"); //在注册表中创建 EXE 加密文件子键
        RegistryKey RKey1 = Registry.ClassesRoot.OpenSubKey(".mrexe", true); //打开新创建的子键
        RKey1.SetValue("", "mrexefile"); //为新创建的子键赋值
        RKey1.Close(); //关闭子键
        Registry.ClassesRoot.CreateSubKey("mrexefile"); //创建 mrexefile 子键
        RegistryKey RKey2 = Registry.ClassesRoot.OpenSubKey("mrexefile", true); //打开 mrexefile 子键
        RKey2.CreateSubKey("DefaultIcon"); //创建 DefaultIcon 子键
        RKey2.CreateSubKey("shell"); //创建 shell 子键
        RKey2.Close(); //关闭子键
        RegistryKey RKey3 = Registry.ClassesRoot.OpenSubKey("mrexefile\\DefaultIcon", true); //打开子键
        RKey3.SetValue("", strPath); //为子键设置值
        RKey3.Close(); //关闭子键
    }
}
```

```

RegistryKey RKey4 = Registry.ClassesRoot.OpenSubKey("mrexefile\shell", true); //打开子键
RKey4.CreateSubKey("使用口令打开"); //创建子键
RKey4.Close(); //关闭子键
RegistryKey RKey5 = Registry.ClassesRoot.OpenSubKey("mrexefile\shell\使用口令打开", true); //打开子键
RKey5.CreateSubKey("command"); //创建子键
RKey5.Close(); //关闭子键
//打开子键
RegistryKey RKey6 = Registry.ClassesRoot.OpenSubKey("mrexefile\shell\使用口令打开\command", true);
RKey6.SetValue("", strName + "\F %1"); //为子键设置值
RKey6.Close(); //关闭子键
}
catch
{
}
}

```

单击“加密”按钮，首先判断是否选择要加密的 EXE 文件，如果已经选择，则使用用户输入的加密密码对选择的 EXE 文件进行加密，执行成功后弹出信息提示。“加密”按钮的 Click 事件的代码如下：

```

private void button2_Click(object sender, EventArgs e) //加密 EXE 文件
{
    string strPwd = textBox2.Text; //记录加密密码
    byte[] btRKey = new byte[0]; //定义一个字节数组，用来记录加密密码
    if (strPwd.Length == 6) //判断密码位数
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[0], (byte)strPwd[1] }; //记录加密密码
    }
    if (strPwd.Length == 7)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[0] };
    }
    if (strPwd.Length >= 8)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6], (byte)strPwd[7] };
    }
    FileStream FStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read); //创建 FileStream 对象
    //创建一个 FileStream 对象，用来创建加密文件
    FileStream NewFStream = new FileStream(textBox1.Text + ".mrexe", FileMode.OpenOrCreate, FileAccess.Write);
    NewFStream.SetLength((long)0); //设置加密文件初始长度为 0
    byte[] buffer = new byte[0x400]; //定义一个字节数组
    int MinNum = 0; //设置加密位置从 0 开始
    long length = FStream.Length; //记录文件长度
    int MaxNum = (int)(length / ((long)0x400)); //计算文件最大值
    DES myDES = new DESCryptoServiceProvider(); //创建 DES 加密对象
    //创建 CryptoStream 加密流对象
    CryptoStream CStream = new CryptoStream(NewFStream, myDES.CreateEncryptor(btRKey, btRKey), CryptoStreamMode.Write);
    while (MinNum < length)
    {
        int count = FStream.Read(buffer, 0, 0x400); //对要加密的文件进行流读取
        CStream.Write(buffer, 0, count); //向加密流中写入数据
        MinNum += count; //记录已经读取的数据位置
    }
    CStream.Close();
    NewFStream.Close();
    FStream.Close();
    File.Delete(textBox1.Text); //删除旧文件
    MessageBox.Show("使用口令加密可执行文件成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

单击“打开”按钮，首先使用用户输入的解密密码对加密过的 EXE 文件进行解密，然后使用 Process 类的 Start 方法打开解密过的 EXE 文件。“打开”按钮的 Click 事件的代码如下：

```

private void button3_Click(object sender, EventArgs e) //解密 EXE 文件
{
    string strPwd = textBox2.Text; //记录解密密码

```

```

FileStream FStream = null; //声明 FileStream 对象, 用来记录旧文件
FileStream NewFStream = null; //声明 FileStream 对象, 用来记录新文件
CryptoStream CStream = null; //声明 CryptoStream 对象, 用来解密文件
try
{
    try
    {
        byte[] btRKey = new byte[0]; //定义一个字节数组, 用来记录解密密码
        if (strPwd.Length == 6) //判断密码位数
        {
            btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[0], (byte)strPwd[1] }; //记录解密密码
        }
        if (strPwd.Length == 7)
        {
            btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[6], (byte)strPwd[0] };
        }
        if (strPwd.Length >= 8)
        {
            btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[6], (byte)strPwd[7] };
        }
        FStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read); //创建 FileStream 对象
        string strNewFile = textBox1.Text.Substring(0, textBox1.Text.Length - 6); //获取新文件名
        NewFStream = new FileStream(strNewFile, FileMode.OpenOrCreate, FileAccess.Write); //创建解密文件
        NewFStream.SetLength((long)0); //设置解密文件初始长度为 0
        byte[] buffer = new byte[0x400]; //定义一个字节数组
        int MinNum = 0; //设置解密位置从 0 开始
        long length = FStream.Length; //记录文件长度
        int MaxNum = (int)(length / ((long)0x400)); //计算文件最大值
        DES myDES = new DESCryptoServiceProvider(); //创建 DES 加/解密对象
        //创建 CryptoStream 加/解密流对象
        CStream = new CryptoStream(NewFStream, myDES.CreateDecryptor(btRKey, btRKey), CryptoStreamMode.Write);
        while (MinNum < length)
        {
            int count = FStream.Read(buffer, 0, 0x400); //对要解密的文件进行流读取
            CStream.Write(buffer, 0, count); //向解密流中写入数据
            MinNum += count; //记录已经读取的数据位置
        }
        CStream.Close();
        FStream.Close();
        NewFStream.Close();
        File.Delete(textBox1.Text); //删除加密文件
        System.Diagnostics.Process.Start(strNewFile); //打开解密后的 EXE 文件
    }
    catch
    {
        MessageBox.Show("口令错误!", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Error);
        textBox2.Focus();
    }
}
finally
{
    CStream.Close();
    FStream.Close();
    NewFStream.Close();
}
}

```

秘笈心法

心法领悟 583: 如何区别 0、空字符串、NULL、Empty 和 Nothing?

对于声明后未赋值的数值类型变量, 它们的默认值为 0; 对于声明后未赋值的字符串变量, 则默认值为空字符串; NULL 关键字说明变量不包含有效数据, 它是将 NULL 值显式地赋值给变量的结果, 也可能是包含 NULL

的表达式之间进行运算的结果; Empty 关键字表示未初始化的变量的默认值; Nothing 关键字用于将对象变量从实际对象中分离出来。

实例 584

使用对称算法加密解密文件

光盘位置: 光盘\MR\19\584

高级

趣味指数: ★★★★★

实例说明

本实例使用对称算法实现加密和解密文件, 在本实例的对称算法加密窗体中, 首先选择原文件路径, 然后输入加密密码和加密后的文件路径, 最后单击“加密”按钮实现对原文件加密, 如图 19.14 所示。在本实例的对称算法解密窗体中, 首先选择要解密的源文件路径, 然后输入解密密码和解密后文件路径, 最后单击“解密”按钮实现对源文件解密, 如图 19.15 所示。

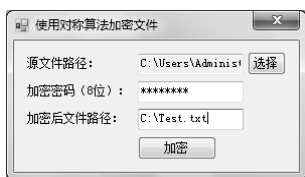


图 19.14 对称算法加密窗体



图 19.15 对称算法解密窗体

关键技术

本实例主要通过调用 DESCryptoServiceProvider 类的 CreateDecryptor 方法和 CryptoStream 类的相关方法, 实现使用对称算法加密和解密文件的功能。

 说明: 关于 DESCryptoServiceProvider 类的 CreateDecryptor 方法和 CryptoStream 类的详细讲解, 请参见实例 575 中的关键技术。

设计过程

(1) 打开 Visual Studio 2008 开发环境, 新建一个 Windows 窗体应用程序, 解决方案命名为 SymmetricalEncrypt, 项目名称命名为 EncryptFile, 作为实现加密的项目; 然后添加一个新项目, 命名为 UnEncryptFile, 作为实现解密的项目。

(2) 更改 EncryptFile 项目中默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加 3 个 TextBox 控件, 分别用来显示源文件路径、输入加密密码和加密后的文件路径; 添加两个 Button 控件, 分别用来选择源文件和实现对文件的加密。

(3) 更改 UnEncryptFile 项目中默认窗体 Form1 的 Name 属性为 Frm_Main, 在该窗体中添加 3 个 TextBox 控件, 分别用来显示源文件路径、输入解密密码和解密后的文件路径; 添加两个 Button 控件, 分别用来选择源文件和实现对文件的解密。

(4) 程序主要代码如下。

在 EncryptFile 项目的 Frm_Main 主窗体中, 单击“加密”按钮实现对源文件的加密功能, 实现代码如下:

```
private void button2_Click(object sender, EventArgs e)
{
    string myFile = textBox1.Text;           //获取源文件路径
    string myPassword = textBox2.Text;       //获取加密密码
    string myEnFile = textBox3.Text;         //获取加密后的文件路径
    try
    {
        byte[] myIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF }; //设置向量
        byte[] myKey = System.Text.Encoding.UTF8.GetBytes(myPassword); //设置密钥
    }
}
```

```

//源文件的文件流
FileStream myInStream = new FileStream(myFile, FileMode.Open, FileAccess.Read);
//加密后文件的文件流
FileStream myOutStream = new FileStream(myEnFile, FileMode.OpenOrCreate, FileAccess.Write);
myOutStream.SetLength(0); //初始文件流的长度
byte[] myBytes = new byte[100]; //定义缓冲区
long myInLength = 0; //定义不断变化的流的长度
long myLength = myInStream.Length; //获取源文件的文件流的长度
DES myProvider = new DESCryptoServiceProvider(); //定义标准的加密算法实例
//实现将数据流链接到加密转换的流
CryptoStream myCryptoStream = new CryptoStream(myOutStream, myProvider.CreateEncryptor(myKey, myIV), CryptoStreamMode.Write);
//从源文件流中每次读取 100 个字节，然后写入加密转换的流
while (myInLength < myLength)
{
    int mylen = myInStream.Read(myBytes, 0, 100); //读取源文件流
    myCryptoStream.Write(myBytes, 0, mylen); //写入加密转换的流
    myInLength += mylen; //计算写入的流长度
}
myCryptoStream.Close(); //关闭资源
myInStream.Close();
myOutStream.Close();
MessageBox.Show("加密文件成功!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
}

```

在 UnEncryptFile 项目的 Frm_Main 主窗体中，单击“解密”按钮实现对文件的解密功能，实现代码如下：

```

private void button2_Click(object sender, EventArgs e)
{
    string str1 = textBox1.Text; //获取源文件路径
    string strPwd = textBox2.Text; //获取解密密码
    string str2 = textBox3.Text; //获取解密后的文件路径
    try
    {
        byte[] myIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF }; //设置向量
        byte[] myKey = System.Text.Encoding.UTF8.GetBytes(strPwd); //设置密钥
        //源文件的文件流
        FileStream myFileIn = new FileStream(str1, FileMode.Open, FileAccess.Read);
        //解密后文件的文件流
        FileStream myFileOut = new FileStream(str2, FileMode.OpenOrCreate, FileAccess.Write);
        myFileOut.SetLength(0); //初始化文件流的长度
        byte[] myBytes = new byte[100]; //定义缓冲区
        long myLength = myFileIn.Length; //获取源文件流的长度
        long myInLength = 0; //定义不断变化的流的长度
        DES myProvider = new DESCryptoServiceProvider(); //定义标准的加密算法实例
        //实现将数据流链接到解密转换的流
        CryptoStream myDeStream = new CryptoStream(myFileOut, myProvider.CreateDecryptor(myKey, myIV), CryptoStreamMode.Write);
        //从源文件流中每次读取 100 个字节，然后写入解密转换的流
        while (myInLength < myLength)
        {
            int mylen = myFileIn.Read(myBytes, 0, 100); //读取源文件流
            myDeStream.Write(myBytes, 0, mylen); //写入解密转换的流
            myInLength += mylen; //计算写入的流长度
        }
        myDeStream.Close(); //关闭资源
        myFileOut.Close();
        myFileIn.Close();
        MessageBox.Show("解密文件成功!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

■ 秘笈心法

心法领悟 584: 如何批量替换某一类字符串?

实际应用中经常遇到批量替换字符串的问题,如 Microsoft Visual Studio 2008 开发环境中的“编辑”菜单下的“全部替换”功能。用 C#实现批量替换字符串的代码如下:

```
public int M_int_index = -1;
private int M_int_start;
private int M_int_end;
M_int_index = 0;
while (M_int_index != -1)
{
    M_int_start = 0;
    M_int_end = richTextBox1.Text.Trim().Length;
    M_int_index = richTextBox1.Find(this.textBox1.Text.Trim(), M_int_start, M_int_end, RichTextFind.None);
    if (M_int_index == -1)
    {
        MessageBox.Show(this, "全部" + this.textBox1.Text + "已替换完毕。", "未找到", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        richTextBox1.SelectedText = textBox2.Text;
        M_int_index += this.textBox1.Text.Length;
    }
}
```