

第五章 加密模块

5.1 明文 or 加密

到现在，我们写的网络程序可以说是“裸奔”的，因为我们网络通过 socket 发送，完全是明文传输的，没有一点点点的加密。想象一下，如果你写的服务器，结果是明文传输，用户账号密码，甚至任何消息都会被破解，被黑客利用。可想而知，网络程序如果不加密，后果很严重，你上网浏览了什么图片，发了什么帖子逛了什么论坛，监听者都知道，你的游戏会被破解，被人写出利用它的外挂，你的游戏账号会被陌生人登陆洗号，甚至漏洞被利用刷金币。

所以网络模块非加密不可！

现在的 http 协议网页就是不安全的，登陆信息只是简单的 base64 编码了一下，ftp 也是明文传输的，smtp 早期协议也是明文传输的，现在 smtp/http 可以使用 ssl 加密传输，安全了不少。

总之，我们本章要学习密码学相关知识。

5.2 密码学浅谈

密码学是研究如何隐密地传递信息的学科。在现代特别指对信息以及其传输的数学性研究。在通信过程中，待加密的信息称为明文，已被加密的信息称为密文。

下面将介绍数据块加密方法。适用于网络数据包加密，也适用于文件加密。

5.3 加密方法简介

最古典的两个加密技巧是：

置换 (Transposition cipher)：将字母顺序重新排列，例如‘help me’变成‘ehpl em’。

替代 (substitution cipher)：有系统地将一组字母换成其他字母或符号，例如‘fly at once’变成‘gmz bu podf’（每个字母用下一个字母取代）。这两种单纯的方式都不足以提供足够的机密性。凯撒密码是最经典的替代法。

例如替代法（每个字母用下一个字母取代）用 c/c++ 代码来实现如下

- 类型 1

```

void    Crypt( char *data , int len )
{
    for (int i=0;i<len;i++)
    {
        data[i] ++;
    }
}

void    Decrypt( char *data , int len )
{
    for (int i=0;i<len;i++)
    {
        data[i] --;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char data[25]="hello123";
    Crypt(data,strlen(data));
    printf("%s \n",data);

    Decrypt(data,strlen(data));
    printf("%s \n",data);

    return 0;
}

```

运行结果如图

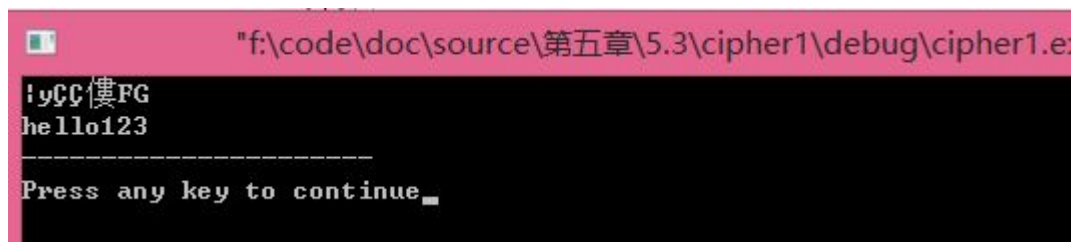


可以看出我们的数据加密后，密文依然有一些信息被暴露出来。
 可以稍微修改一下，不是++，移动下一个字节，而是加一个很大的数，2-255 都可以，解密时，减去这个数，
 代码如下

- 类型 2

```
char key=20;
void Crypt2( char *data , int len )
{
    for (int i=0;i<len;i++)
    {
        data[i] +=key;
    }
}

void Decrypt2( char *data , int len )
{
    for (int i=0;i<len;i++)
    {
        data[i] -=key;
    }
}
```



可以看出后面输出的密文强度增大了，看不出某些字符了。但依然可以分析出代码重复规律，再把代码修改一下，按长度做加密 key.

- 类型 3

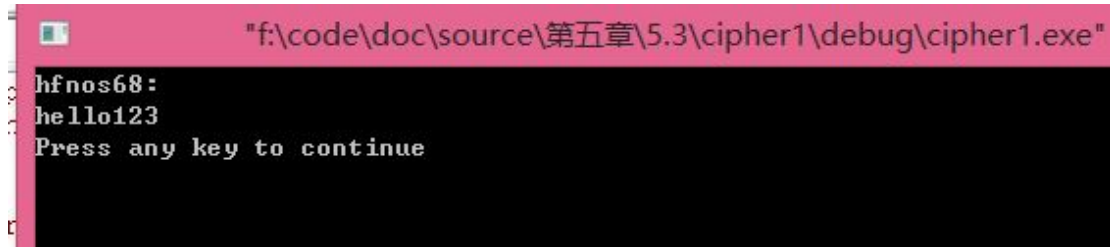
```
void Crypt3( char *data , int len )
{
    for (int i=0;i<len;i++)
    {
        data[i] +=i;
    }
}

void Decrypt3( char *data , int len )
```

```

{
    for (int i=0;i<len;i++)
    {
        data[i] -=i;
    }
}

```



这样密文稍微强度增加了点，看不出原文以及字符规律了。这种加密算法还可以变异出很多算法，大家可以自由发挥。

5.4 XOR 加密

还有一个经典的程序的加密算法：异或（XOR）

因为用同一值去异或两次就可以恢复出原来的值，所以加密和解密都严格采用同一程序。即：P XOR K = C C XOR K = P 这种方法比较简单,没有实际的保密性

代码比上一个更简单。

```

void XOR(char* data,int len)
{
    for (int i=0;i<len;i++)
    {
        data[i] ^= i;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //待加密的buffer
    char data[25]= "hello 123456";

    XOR(data, strlen(data));//加密

    printf("[%s] \n",data);
}

```

```

XOR(data, strlen (data));//解密

printf("[%s] \n",data);

}

```

xor一次加密，xor两次就解密了

运行结果如下，可以看出密文已经看不出原文是啥东西了。



xor 加密由于算法太过简单，所以还是很脆弱的，被黑客可以轻易破解掉，于是有了更多更强的加密算法。

5.5 收发映射表加密

收发映射表（替换）加密法

基本思想依然是替换。但是把映射的字符数量扩大到了很大的范围。
 比如我们字符一个 `char` 的范围是 `0-0xFF`。即表达的数据可以是 0 到 255，那么把 0 到 255 的数字按顺序放在 `BYTE ByteMap [256]`；然后打乱数组数据。
 代码如下

发送字节映射表 负责加密映射，明文数据作为下标查询出一个密文
 接收字节映射表 负责解密映射，密文作为下标查询出的数据就是明文数据

```

#include <Windows.h>

#include<iostream>
#include<algorithm>
using namespace std;

BYTE ByteMap[256];
BYTE ByteMap2[256];

void map_crypt(char* data,int len)
{
    for (int i=0;i<len;i++)
    {
        BYTE ch=data[i];
        data[i]=ByteMap[ch];
    }
}

void map_decrypt(char* data,int len)
{
    for (int i=0;i<len;i++)
    {
        BYTE ch=data[i];
        data[i]=ByteMap2[ch];
    }
}

void build_map()
{
    for (int i=0;i<256;i++)
    {
        ByteMap[i]=i;
    }

    int k=0;
    for (int i=0;i<256;i++,k++)
    {
        printf("0x%x ",ByteMap[i]);
    }

    printf("\n%d\n\n",k);
}

```

```

BYTE*   a_begin   =   ByteMap;
BYTE*   a_end     =   ByteMap   +   sizeof(ByteMap)/sizeof(BYTE);
random_shuffle(a_begin,a_end);
for(BYTE*   p     =   a_begin;   p!=a_end;   p++)
{
    //   cout<<*p<<"\t";
}
for (int i=0;i<256;i++,k++)
{
    printf("0x%x ",ByteMap[i]);
}

printf("\n\n\n" );


for (int i=0;i<256;i++,k++)
{
    ByteMap2[ByteMap[i]]=i;
}

for (int i=0;i<256;i++,k++)
{
    printf("0x%x ",ByteMap2[i]);
}

}

int _tmain(int argc, _TCHAR* argv[])
{

    build_map();//生成收发映射表
    //待加密的buffer
    char data[55]= "收发映射表加密 hello 123456";


    map_crypt(data, strlen(data));//加密
    printf("map_crypt[%s] \n",data);

    map_decrypt(data, strlen (data));//解密
    printf("decrypt[%s] \n",data);

    return 0;
}

```

运行如下



```
"b:\map\debug\map.exe"
0x74 0x81 0xa4 0x25 0x63 0x8a 0x83 0x19 0xab 0xb0 0x6f 0xf1 0xd2 0x70 0xe7 0x37
0x65 0x13 0xa8 0x2f 0xfc 0x34 0xee 0x45 0x2c 0x24 0xb5 0xe8 0x7d 0xbe 0xbf 0xa3
0xf4 0x33 0x77 0x9e 0xf7 0xae 0x3d 0x23 0xdc 0xb2 0x4f 0xec 0x3b 0x5a 0x3e 0x15
0x80 0x62 0x78 0x96 0x51 0x53 0xc9 0x6b 0x22 0xd5 0x73 0xb4 0x75 0x28 0xfe 0xb1
0x16 0x18 0x17 0x54 0x29 0x92 0x46 0xe 0x52 0x8d 0xd6 0xc6 0x1b 0x1e 0xc5 0x94 0
x5f 0x1f 0xad 0x3c 0x40 0x6a 0xbc 0x97 0x64 0xb8 0x66 0xf6 0x9d 0x5c 0x12 0x41 0
xed 0x2e 0xe2 0xd 0x4e 0x27 0x9 0x3f 0xb3 0x71 0xfd 0xa0 0x11 0xd1 0xba 0x93 0x
c3 0xaf 0x61 0xf2 0x60 0xf9 0x7f 0x85 0x38 0xd0 0xb9 0xdd 0xb 0xd4 0xa2 0x67 0x8
7 0xf0 0xcd 0xcc 0xf5 0xa6 0x7c 0x99 0x5 0x8c 0x1c 0xcf 0xc4 0x79 0xbh 0xca 0x58
0xb7 0x59 0x5d 0x49 0x47 0xf 0x26 0x7 0x5e 0xc1 0x21 0x7b 0xf8 0x9b 0xac 0x2d 0
xd3 0x2 0x48 0xbd 0xf3 0x89 0x2b 0x9a 0xc0 0x44 0xd9 0x2a 0xe1 0xaa 0x14 0x7e 0x
72 0x6c 0x30 0x31 0x5b 0x9f 0xe0 0x95 0xe3 0x4b 0x84 0x8 0x4d 0xd7 0x50 0x20 0xa
0xef 0xcb 0x39 0x36 0x32 0xdb 0x43 0xa9 0x69 0xe9 0xde 0x0 0xfb 0x57 0xe4 0xda
0x82 0x4c 0x91 0x42 0x90 0x55 0x98 0x8f 0xea 0xd 0xc7 0x6d 0x1a 0xc2 0x35 0xfa 0
x3 0x6 0xce 0x6e 0xeb 0xb6 0x88 0xa7 0x76 0xa1 0x8b 0x4 0xc8 0x3a 0x1 0xe6 0x7a
0x86 0xc 0x56 0x10 0xdf 0xe5 0x9c 0xd8 0x4a 0xff 0x8e 0xa5 0x68 map_encrypt[? 6
?`U <?膊
来?恩]
decrypt[收发映射表加密 hello 123456]
```

开始输出加密映射表，然后输出的是解密映射表，最后 decrypt 打印出我们加密前的明文，说明收发映射加密工作正常。

其实，这个加密方法也容易被破解，只是破解成本增大了，黑客需要绘制出收发表，或是通过破解找到你的收发表数组定义。

5.6 对称加密 AES/DES

前面几个经典的加密算法非常小儿科，可以轻易被高手给破解掉，现在开始介绍现代工业级别的标准加密算法。

高级加密标准（英语：Advanced Encryption Standard，缩写：AES），在密码学中又称 Rijndael 加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一。（摘自网络）

DES 全称为 Data Encryption Standard，即数据加密标准，是一种使用密钥加密的块算法，1976 年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS），随后在国际上广泛流传开来。（摘自网络）

可以看出 DES 出现的最早，AES 加密强度大于 DES。

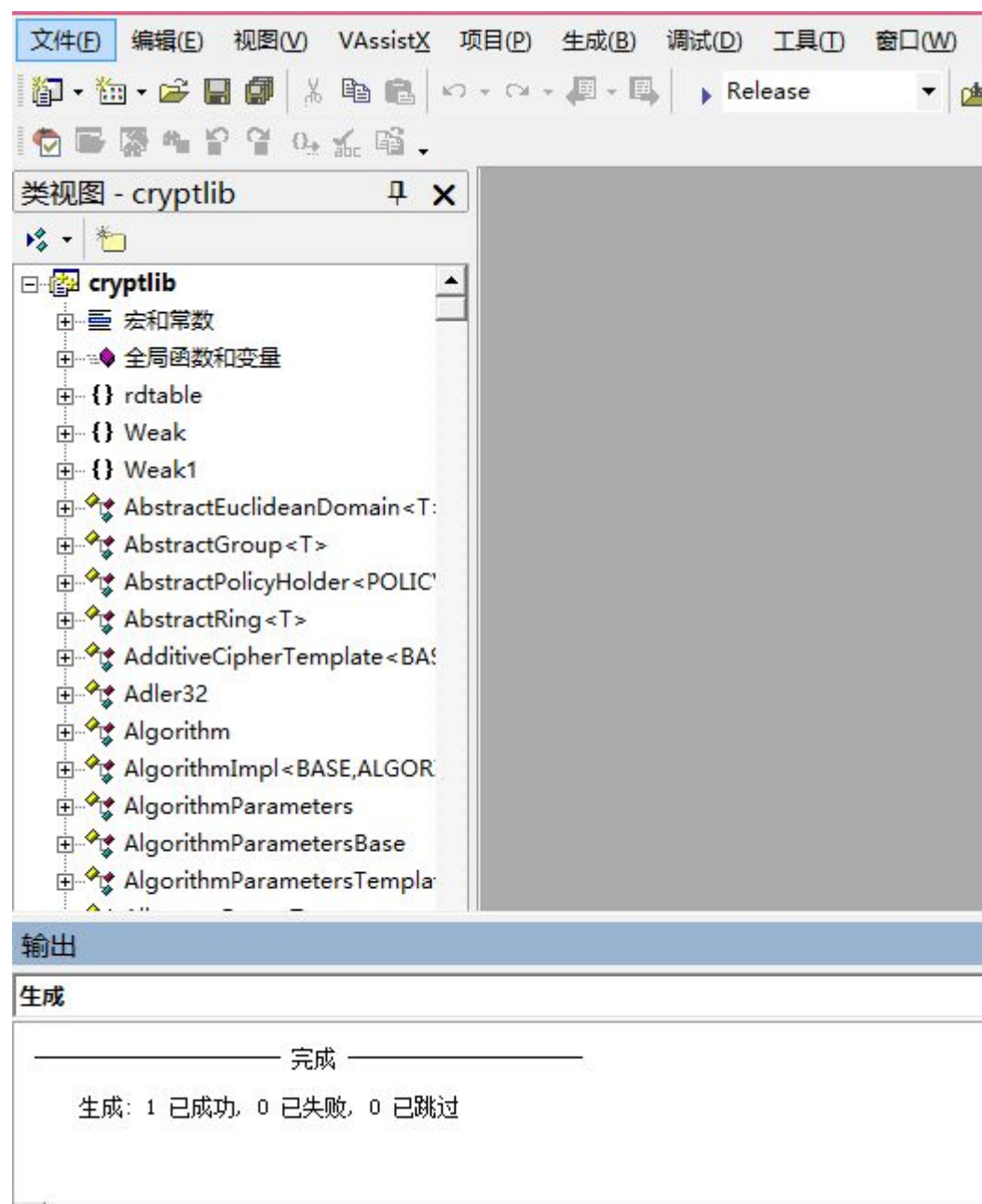
对称加密（AES/DES），加密、解密用的密匙相同。就像日常生活中的钥匙，开门和锁门都是同一把。

我不打算讲解这个的内部复杂的加密算法原理，只讲解实际使用。这里我采用 crypto++ 算法库

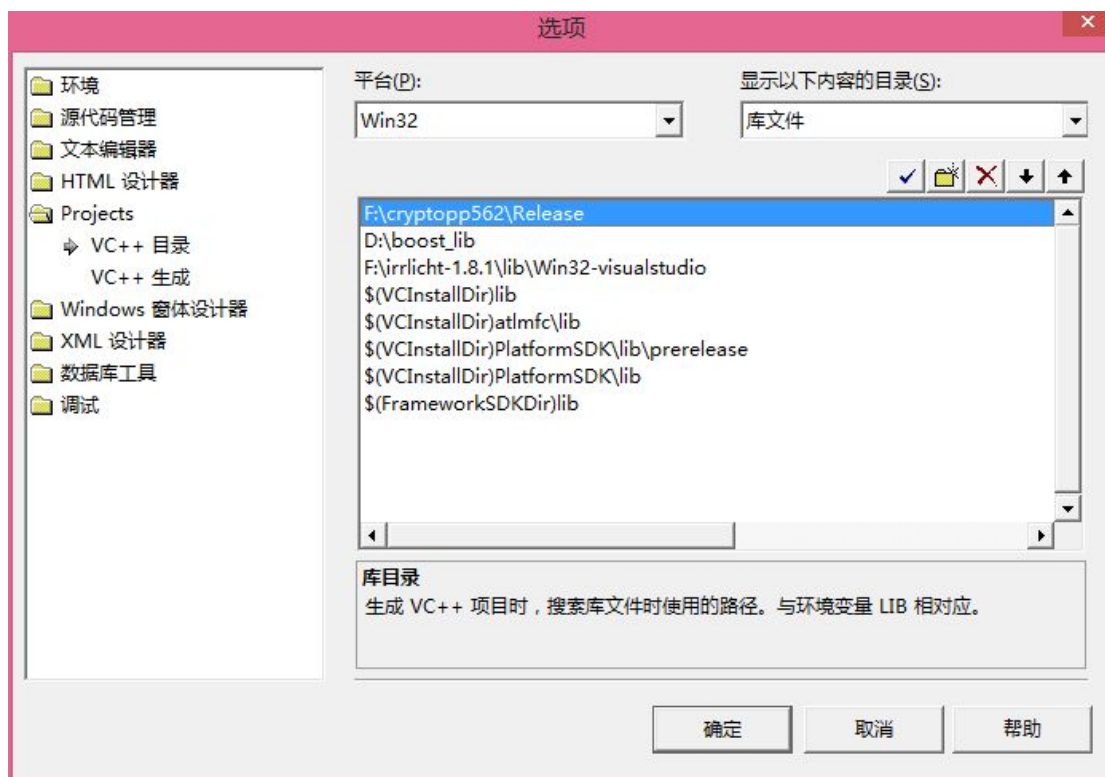
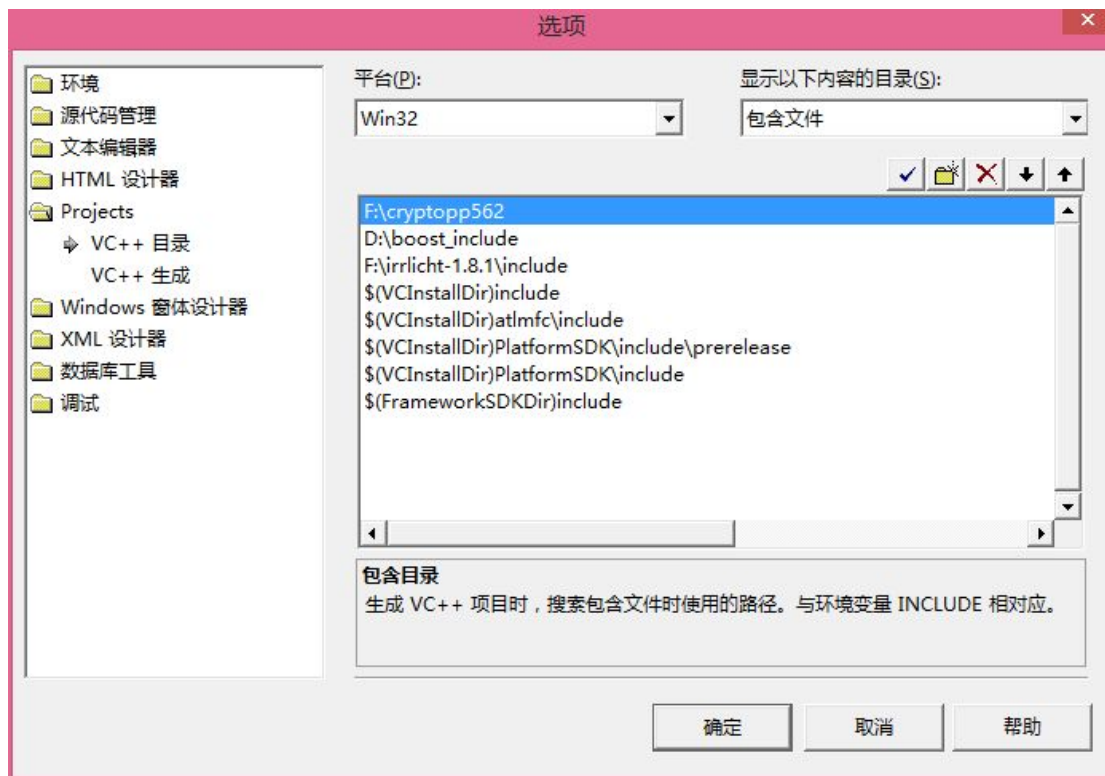
Crypto++ 库是一个免费的 C++ 类库的加密方案。该库包含以下算法：MD2, MD4, MD5, Panama Hash, DES, ARC4, RSA, DSA, SHA-1, SHA-2, AES (Rijndael), RC6 等等。

官网 <http://www.cryptopp.com/> 当前最新版本 5.6.2

去官网下载安装包，解压到某目录，比如 F:\cryptopp562，进入目录，打开 cryptlib.dsp，选择 release，F7 编译。一会就会编译出静态库。



设置编译器头文件目录，添加 F:\cryptopp562 ， lib 目录添加 F:\cryptopp562\release 文件夹。



先看 DES 加密的例子

```
void DES_Test(){

    //待加密的明文
    char data[255] = "hello 123456";
    //加密长度
    int dataLen =  strlen(data) ;

    //////////////////////////////////////
    //DES  Encrypt

    MyDES a;
    a.init("kkkey", "iviv");
    char enc[321]="";
    a.Enc(data,dataLen,enc);

    printf("Enc:");
    printf("[%s]\n",enc);
    for (int i=0;i<dataLen;i++)
    {
        printf("%d ",( BYTE)enc[i]);
    }

    printf("\n");

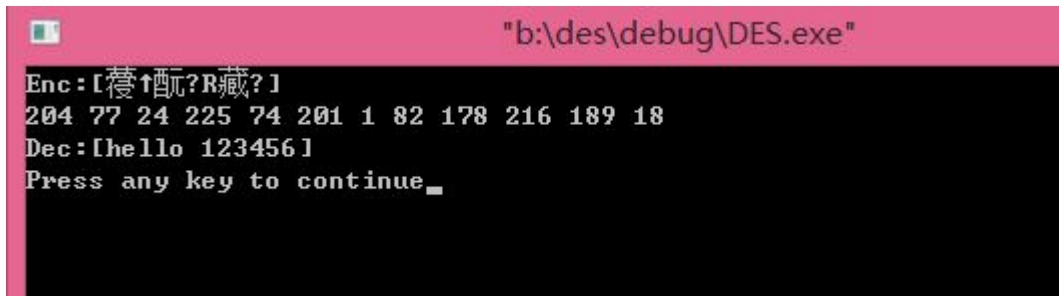
    //////////////////////////////////////
    // DES  Decrypt

    char dec[321];
    a.Dec(enc,dataLen,dec);

    printf("Dec:[]");
    for (int i=0;i<dataLen;i++)
    {
        printf("%c",dec[i]);
    }
}
```

```
printf("]\n");  
  
}
```

运行结果如下



Enc 输出的字符串已经是乱码了。完全看不出明文是什么。第二行输出的数字是加密后的每一个 byte 的数值。

MyDES.h 源码如下

```
#pragma once  
  
#include "osrng.h"  
using CryptoPP::AutoSeededRandomPool;  
  
#include "cryptlib.h"  
using CryptoPP::Exception;  
  
#include "hex.h"  
using CryptoPP::HexEncoder;  
using CryptoPP::HexDecoder;  
  
#include "filters.h"  
using CryptoPP::StringSink;  
using CryptoPP::StringSource;  
using CryptoPP::StreamTransformationFilter;  
  
#include "des.h"  
using CryptoPP::DES;  
  
#include "modes.h"
```

```
using CryptoPP::CFB_Mode;
```

```
#pragma comment(lib,"cryptlib.lib")
```

```
/*  
*/
```

DES

对称加密.

key iv 固定，则每次加密相同的数据的结果密文都一样.虽然别人看不到明文，但可以猜到是相同的数据包。

```
*/  
/*
```

```
class MyDES
```

```
{
```

```
public:
```

```
    MyDES()//默认key
```

```
{
```

```
        memset( key ,0, sizeof(key) );
```

```
        memset( iv ,0,  sizeof(iv) );
```

```
}
```

```
~MyDES(){
```

```
}
```

```
//key[16]
```

```
void init( const char* k, const char* v )
```

```
{
```

```
    //使用某一组key
```

```
    memcpy(key,k, CryptoPP::DES::DEFAULT_KEYLENGTH );
```

```
    memcpy(iv,v, CryptoPP::DES::DEFAULT_KEYLENGTH );
```

```
}
```

```
//随机生成key
```

```
void GenerateRandomKey()
```

```

{
    rnd.GenerateBlock(key, DES::DEFAULT_KEYLENGTH);
    rnd.GenerateBlock(iv, DES::BLOCKSIZE);
}

//加密
void Enc( const char* src,int size,char* outdata )
{
    memset(outdata,0,size);

    //////////////////////////////////////
    // Encrypt

    CFB_Mode<DES>::Encryption cfbEncryption(key, sizeof(key), iv);
    cfbEncryption.ProcessData((byte*)outdata, (byte*)src, size);

}

//解密
void Dec( const char* src,int size,char* outdata )
{
    memset(outdata,0,size);

    //////////////////////////////////////
    // Decrypt

    CFB_Mode<DES>::Decryption cfbDecryption(key, sizeof(key), iv);
    cfbDecryption.ProcessData((byte*)outdata, (byte*)src, size);

}

// Generate a random key
byte key[DES::DEFAULT_KEYLENGTH];

// Generate a random IV
byte iv[DES::BLOCKSIZE];

private:

    AutoSeededRandomPool rnd;

```

$$\};$$

代码很简单，实际编码中我们的 key 和 iv 要保存好，不然被黑客获得，程序就赤裸了。

下面看 AES 加密

```
void AES_Test(){  
  
    //待加密的明文  
    char data[255] = "hello 123456";  
    //加密长度  
    int dataLen = (int)strlen(data);  
  
    ///////////////////////////////////////  
    // Encrypt  
  
    MyAES a;  
    a.init("kkkey","iviv");  
    char enc[321]="";  
    a.Enc(data,dataLen,enc);  
  
  
    printf("Enc:");  
    printf("[%s]\n",enc);  
    for (int i=0;i<dataLen;i++)  
    {  
        printf("%d ",( BYTE)enc[i]);  
    }  
  
    printf("\n");  
  
    ///////////////////////////////////////  
    // Decrypt  
  
    char dec[321];
```



```

        a.Dec(enc,dataLen,dec);

        printf("Dec:[");
        for (int i=0;i<dataLen;i++)
        {
            printf("%c",dec[i]);
        }

        printf("]\n");
    }
}

```

MyAES.h

```

#pragma once

#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;

#include "cryptlib.h"
using CryptoPP::Exception;

#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;

#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;

#include "aes.h"
using CryptoPP::AES;

#include "modes.h"
using CryptoPP::CFB_Mode;

#pragma comment(lib,"cryptlib.lib")

```

```
/*  
/*
```

AES

对称加密.

key iv 固定, 则每次加密相同的数据的结果密文都一样. 虽然别人看不到明文, 但可以猜到是相同的数据包。

```
*/  
/*
```

```
class MyAES  
{  
public:  
    MyAES()//默认key  
    {  
        memset( key ,0, sizeof(key) );  
        memset( iv ,0, sizeof(iv) );  
    }  
    ~MyAES(){  
  
    }  
    //key[16]  
    void init( const char* k, const char* v )  
    {  
        //使用某一组key  
        memcpy(key,k, CryptoPP::AES::DEFAULT_KEYLENGTH );  
        memcpy(iv,v, CryptoPP::AES::DEFAULT_KEYLENGTH );  
    }  
  
    //随机生成key  
    void GenerateRandomKey()  
    {  
        rnd.GenerateBlock(key, AES::DEFAULT_KEYLENGTH);  
        rnd.GenerateBlock(iv, AES::BLOCKSIZE);  
    }  
  
    //加密
```

```

void Enc(  const char* src,int size,char* outdata )
{

    memset(outdata,0,size);

    //////////////////////////////////////
    // Encrypt

    CFB_Mode<AES>::Encryption cfbEncryption(key, sizeof(key), iv);
    cfbEncryption.ProcessData((byte*)outdata, (byte*)src, size);

}

//解密
void Dec(  const char* src,int size,char* outdata  )
{

    memset(outdata,0,size);

    //////////////////////////////////////
    // Decrypt

    CFB_Mode<AES>::Decryption cfbDecryption(key, sizeof(key), iv);
    cfbDecryption.ProcessData((byte*)outdata, (byte*)src, size);

}

// Generate a random key
byte key[AES::DEFAULT_KEYLENGTH];

// Generate a random IV
byte iv[AES::BLOCKSIZE];

private:

    AutoSeededRandomPool rnd;

};

```

运行如下

```
"b:\Aes\Debug\AES.exe"
Enc:[J4枪甌其3j櫟]
74 52 199 185 174 84 221 189 51 106 152 197
Dec:[hello 123456]
Press any key to continue
```

和 DES 类似，代码也类似。其实 AES 就是 DES 的加强版。增强了加密位。

5.7 非对称加密 RSA

AES/DES 是对称加密，即采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。单个密钥肯定不够安全，加密双方都有这个密钥，一旦黑客获取了，通讯就被破解了。

有对称加密，自然有非对称加密。

非对称加密算法需要两个密钥来进行加密和解密，这两个密钥是公开密钥（public key，简称公钥）和私有密钥（private key，简称密钥）

如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。

非对称密码体制的特点：算法强度复杂、安全性依赖于算法与密钥但是由于其算法复杂，而使得加密解密速度没有对称加密解密的速度快。对称密码体制中只有一种密钥，并且是非公开的，如果要解密就得让对方知道密钥。所以保证其安全性就是保证密钥的安全，而非对称密钥体制有两种密钥，其中一个是公开的，这样就可以不需要像对称密码那样传输对方的密钥了。这样安全性就大了很多。（摘自网络）

主要算法有 RSA、Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）。使用最广泛的是 RSA 算法，Elgamal 是另一种常用的非对称加密算法。

下面讲解 RSA 的使用

```
MyRSA rsa;
rsa.init("PriK","PK","seed");

char msg[1024]="hello 123456";
```

```
string enc= rsa.RSAEncryptString(msg);
cout<< "RSAEncryptString :\t" << enc << endl << endl;
```

```
string dec= rsa.RSADecryptString(enc.c_str());
cout<< "RSADecryptString :\t" << dec << endl << endl;
```

调用很简单。、

运行结果如下



```
"b:\rsa\debug\RSA.exe"
priKey :      PriK
pubKey :      PK
seed :  seed

RSAEncryptString :      940E06CB96FBF01E9753EF046F91852B2282874BA1DC717465089F9F
29B382770E4E9E00C0113ED759E54DC0C1DE41C4933A8DDC7310B5A9B20608CA17161BA6E1F7F0D2
C43B05D269E2E8FFE80B1A23A135BF883E70D2A02050357757A39CADB1F582C0303C8FE081AFADF8
3392EC8D3736B68E004BDF624A05E88C980F3100

RSADecryptString :      hello 123456
Press any key to continue_
```

Debug	2015/9/6 17:19	文件夹	
main.cpp	2015/9/6 17:19	CPP 文件	1 KB
MyRSA.cpp	2013/4/29 15:50	CPP 文件	3 KB
MyRSA.h	2013/4/29 15:56	H 文件	2 KB
PK	2015/9/6 17:19	文件	1 KB
PriK	2015/9/6 17:19	文件	2 KB
ReadMe.txt	2015/9/6 17:18	文本文档	1 KB
RSA.ncb	2015/9/6 17:21	Visual C++ Intell...	27 KB
RSA.sln	2015/9/6 17:18	Microsoft Visual...	1 KB
RSA.suo	2015/9/6 17:21	Visual Studio Sol...	8 KB
RSA.vcproj	2015/9/6 17:19	VC++ Project	4 KB

查看工程目录,发现新增了两个文件,这两个就是公钥(public key)和私钥(private key)。

例如 基于用户名和密码验证，Client 要传递用户名和密码，但有个问题就是如何保证用户名和密码在通讯过程即使被截取了也难以得到用户名和密码呢？答案就是 RSA

RSA 提供 public key 加密而 private key 解密的方式，可以把 public key 提供给请求方就行了，private key 保存在服务端

Server 把公钥提供给 Client. Client 用公钥加密数据后发送给 Server，Server 接收数据然后用私钥解密。这样就可以保证加密的东西只有 Server 才有解密，即使加密信息被其他人拦截也难以获取原有信息。

通常只用 RSA 加密重要的信息，比如账号密码，银行账号密码。待加密的数据不宜过长，否则加解密时间会很长。一般都是 RSA 做登录验证，登录成功后，双方通过 RSA 加密交换对称加密用的 key 信息。然后双方切换到 AES/DES 加密。

5.8 校验/散列算法 CRC32/MD5/SHA1

前面几节我们讲解了常见的一些加密方法。可以实现网络通讯的加密，或者文件的加密。总之都是保护我们数据的。但是往往还是会被人破解，被黑客从中篡改伪造数据，为此我们需要使用一种方法来应对它，就是校验（签名）。

比如客户端发送一条很重要的消息，被黑客截取并篡改了一部分数据，然后发送到服务器，如果没有使用校验，那么服务器以为是收到客户端的完整数据，但是如果给消息增加一个签名，就算黑客篡改了数据，但是签名，由于算法问题，黑客不一定能识别破解签名算法，所以发送到服务器的数据，先校验签名，如果签名错误，说明传输中部分数据被破坏（篡改）了。这样黑客就被成功防御了。

为了信息不可逆，要选取单向散列算法，又称单向 Hash 函数、杂凑函数，就是把任意长的输入消息串变化成固定长的输出串且由输出串难以得到输入串的一种函数。这个输出串称为该消息的散列值。一般用于产生消息摘要，密钥加密等

常见校验签名算法（单向散列函数）有 CRC MD5 SHA

- **CRC** （Cyclic Redundancy Check ）循环冗余校验码 是数据通信领域中最常用的一种差错校验码，CRC 校验由于实现简单，检错能力强，被广泛使用在各种数据校验应用中。

例如 zip 格式的压缩文件，使用 CRC32 作为文件校验的算法。
如图

类型	修改时间	CRC32
PNG 图像	2015/6/15 17:39	B2276744
Data Base File	2015/7/11 10:00	CE0318...
HTML 文档	2015/6/21 2:29	1E00B077
BAK 文件	2015/6/17 13:45	F929BCF6
BAK 文件	2015/6/16 16:00	57815125
HTML 文档	2015/6/17 13:37	55C946A8
BAK 文件	2015/6/17 13:37	55C946A8

- **MD5** (Message Digest Algorithm 5) 是 RSA 数据安全公司开发的一种单向散列算法，MD5 被广泛使用，可以用来把不同长度的数据块进行暗码运算成一个 128 位的数值。

MD5 校验可以应用在多个领域，比如说机密资料的检验，下载文件的检验，明文密码的加密等。我们最熟悉的 md5 应用场景就是账号的密码保存。早期很多不负责任的程序员保存用户密码都是明文。一旦数据库被黑客盗取，造成的危害极大，近几年也不时爆出知名网站数据库被黑，用户信息泄露。为了安全性，用户的密码最好用 md5 保存。

- **SHA** 安全哈希算法 (Secure Hash Algorithm) 是一种较新的散列算法，可以对任意长度的数据运算生成一个 160 位的数值，SHA 家族的五个算法，分别是 SHA-1、SHA-224、SHA-256、SHA-384，和 SHA-512。应用于高强度的签名，比如数字证书的签名，可执行文件的证书签名。

上面的几个算法 `cryptopp` 都有实现。我们可以直接使用。

以字符串 `std::string s="abc";` 为数据源。

CRC32 使用例子

```
#include "crc32.h"
#include <hex.h>
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    std::string s="abc";
    std::cout<<std::hex<< CRC32(s.c_str(),s.length()) <<std::endl;
    //crc32 ("abc") = 352441c2
    return 0;
}
```

CRC32 abc 的输出结果



SHA1 使用例子

```
string sha1(const string & message)
{
    string digest;
    CryptoPP::SHA1 c;
    StringSource(message, true,
                  new HashFilter(c, new HexEncoder(new StringSink(digest))));
    return digest;
}

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<<sha1("abc")<<std::endl;
    //sha1 ("abc") = a9993e364706816aba3e25717850c26c9cd0d89d
    return 0;
}
```

SHA1 abc 的输出结果



MD5 使用例子

```
string MD5(const string & message)
{
    string digest;
    Weak::MD5 md5;
    StringSource(message, true,
                  new HashFilter(md5, new HexEncoder(new StringSink(digest))));
    return digest;
}

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout<<MD5("abc")<<std::endl;

    return 0;
}
```

MD5 abc 的输出结果



5.9 文件校验 SHA1

以上校验的数据都是字符串，其实也可以校验文件。建立一个 a.txt 文件，输入 abc，关闭保存。然后打包成 a.zip，打开 a.zip 如下

名称	大小	压缩后大小	类型	修改时间	CRC32
文件夹					
a.txt	3	5	文本文档	2015/9/7 16:13	352441C2

可以看到文件的 crc32 也是 352441C2，即 a.txt 的 crc32 也就是它的内容 abc 的

crc32 而已。

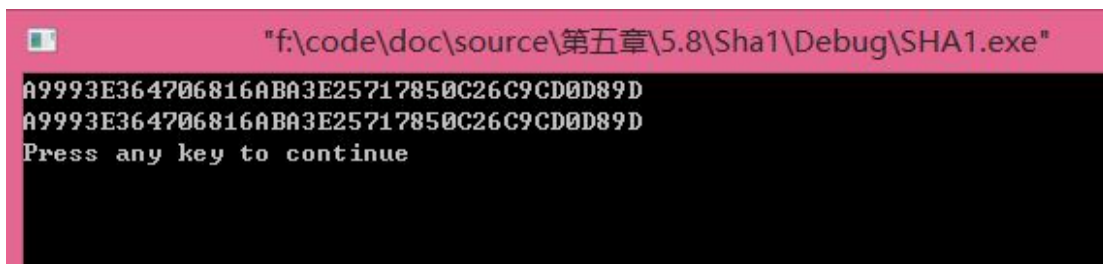
crc32 校验文件也只是多了一步打开读取文件内容而已。

Crypto++提供了 api 可以直接使用校验文件。

```
string sha1_file(const string & file)
{
    string digest;
    CryptoPP::SHA1 c;
    FileSource(file.c_str(), true,
        new HashFilter(c, new HexEncoder(new StringSink(digest))));
    return digest;
}
```

```
std::cout<<sha1_file("a.txt")<<std::endl;
```

将代码添加到 SHA1 工程里，运行如下



SHA1 校验字符串和文件输出的 hash 是一样的。

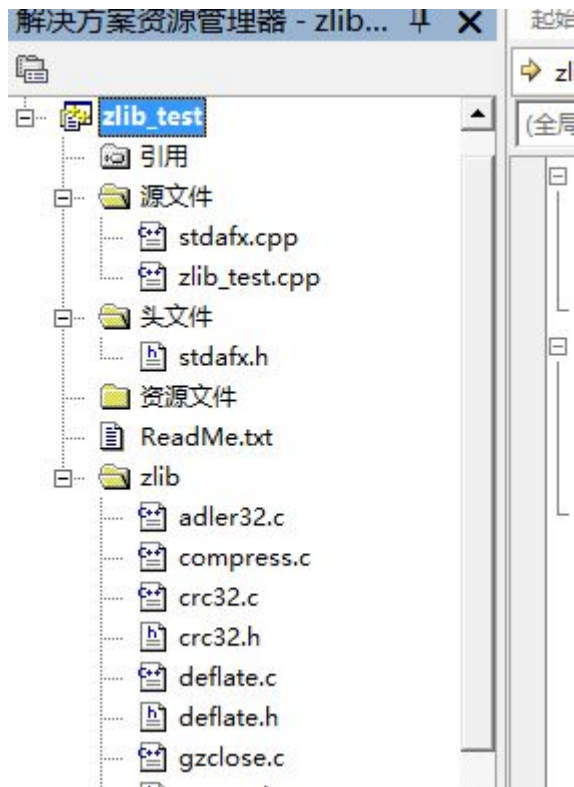
5.10 数据压缩

除了数据加密，数据压缩有时也是必要的，有时可以非常节省带宽，比如很多 web 服务器都开启了 GZIP 压缩。文本的压缩比可以达到很高。服务端如果通讯有许多大量的数据包，那么压缩一下往往会有比较好的优化效果。

压缩方法我使用的是 Zlib 库。Zlib 最新版 1.2.8。

官方网站 <http://www.zlib.net/>

下载 zlib-1.2.8.tar.gz，然后解压到新建的 zlib 目录。新建一个工程，把 zlib 目录移动到工程下，把 zlib 目录下的文件添加到工程文件里，注意不要复制文件夹。如图



如果工程编译出现 fatal error C1010: 在查找预编译头指令时遇到意外的文件结尾 就关闭预编译头

然后写 zlib 测试代码

```
const unsigned char strSrc[]="hello world! 中文测试123";
```

```
int _tmain(int argc, _TCHAR* argv[])  
{
```

```
    unsigned char buff[1024]="",strDst[1024]="";
```

```
    unsigned long srcLen=sizeof(strSrc),bufLen=sizeof(buff),dstLen=sizeof(strDst);
```

```
    printf("原文:[%s]\n大小:%d\n-----\n",strSrc,srcLen);
```

```
    //压缩
```

```
    compress(buff,&bufLen,strSrc,srcLen);
```

```
    printf("压缩后:[%s]\n压缩后大小:%d\n-----\n",buff,bufLen);
```

```
    //解压缩
```

```
    uncompress(strDst,&dstLen,buff,bufLen);
```

```
    printf("\n大小:%d\n 原文:[%s]\n",dstLen,strDst);
```

```
    return 0;  
}
```

运行后结果如下



```
"b:\zlib_test\debug\zlib_test.exe"  
原文:[hello world! 中文测试1231  
大小:25  
-----  
压缩后:[x灏H蛻菴<?麓QT誼i 慚廚1142f1  
压缩后大小:34  
-----  
大小:25  
原文:[hello world! 中文测试1231
```

其实也可以用压缩来作为加密。反正压缩后数据都乱码了（估计有很多人都这么干了）。可以再用其他加密算法把数据混合加密。强度也不错，但遇到有经验的黑客，也会查找到你用的什么算法，只是复杂度增大，破解成本增大了。

5.10 总结

本章我们学习了加密的相关技术，讲解的依然很浅显，但实用性很强，用最简单的加密或比较强的加密时，要因地制宜，看应用场景，分析需求，如果要跨平台多客户端，那么加密算法要考虑各个平台是否算法结果一致性，还要兼顾运算效率速度，数据保护安全性等等。

2016.1.3 作者：司马威
作者博客 <http://blog.csdn.net/smwhotjay>

编程交流 q 群: 316641007