古典密码——Vigenère

算法

使用表格生成密码

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

加解密公式：

$$C_i \equiv P_i + K_i \pmod{26}$$

$$P_i \equiv C_i - K_i \pmod{26}$$

破解：

- 利用Kasiski 实验获取到密钥长度
- 利用重合指数攻击破解密钥

  重合指数是指两个元素随机相同的概率之和，记作

$$CI' = \sum_{i=1}^{n} \frac{f_i}{L} \frac{f_i - 1}{L - 1}$$

  L：密文长度； fi ： 在密文中的出现次数

- 字母频率分析

  判断字母频率与统计字母频率的内积最大的情况

核心代码

数据结构

```
char M[1005],C[1005],K[1005];
char flag; //输出格式
struct Node{
    float value; //重合指数差,与我们的标准重合指数的差值越小越好
    int length;
};
vector< Node > key; //存放key可能的长度和重合指数差
set< int > key_len; //存放key可能的长度
double g[]={0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966,
0.00153, 0.00772, 0.04025,0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327,
0.09056, 0.02758, 0.00978, 0.02360, 0.00150,0.01974, 0.00074};
//英文字母使用频率表
```

加密：直接根据公式

```
for(int m=0;m<lenm;m++)
{
  if(M[m]<'a')
    C[m]=((M[m]-'A'+26)+(K[k]-'a'))%26+'A';
  else
    C[m]=((M[m]-'a'+26)+(K[k]-'a'))%26+'a';
  k++;
  if(k==lenk)
    k=0;
}
```

解密：加密的逆变换

```
for(int c=0;c<lenc;c++)
{
    if(C[c]<'a')
        M[c]=((C[c]-'A'+26)-(K[k]-'a'))%26+'A';
    else
        M[c]=((C[c]-'a'+26)-(K[k]-'a'))%26+'a';
    k++;
    if(k==lenk)
        k=0;
}
```

破解

计算所选分组的重合指数

```
float Coincidence_index(string cipher,int start,int length){
```

```
    float index=0.000;
    int sum=0;
    int num[26];
    memset(num,0,sizeof(num));
    while(start<=cipher.length()){
        num[cipher[start]-'a']++;
        start+=length;
        sum++;
    }
    for(int i=0;i<26;i++){
        if(num[i]<=1) continue;
        index+=(float)(num[i]*(num[i]-1))/(float)((sum)*(sum-1));
    }
    return index;
}
```

根据 kasiski测试法的原理获取key可能的长度

```
void Find_same(string cipher){
    for(int i=3;i<5;i++){
        for(int j=0;j<cipher.length()-i;j++){
            string p=cipher.substr(j,i);
            for(int k=j+i;k<cipher.length()-i;k++){
                string tmp=cipher.substr(k,i);
                if(tmp==p){
                    Node x;
                    x.length=k-j;
                    key.push_back(x);
                }
            }
        }
    }
}
```

求出可能的key的值的最大公因子,经过重合指数检验,对key的长度进行排序

```
void Get_key(string cipher){
    Find_same(cipher);
    for(int i=0;i<key.size();i++){
        int x=key[i].length;
        for(int j=0;j<key.size();j++){
            if(key[i].length>key[j].length)
                key_len.insert(gcd(key[i].length,key[j].length));
            else
                key_len.insert(gcd(key[j].length,key[i].length));
        }
    }
    key.clear();
    set< int >::iterator it=key_len.begin();
    while(it!=key_len.end()){
        int length=*it;

        if(length==1){
```

```
            it++;
            continue;
        }
        float sum=0.000;
        cout<<length<<" ";
        for(int i=0;i<length;i++){
            cout<<Coincidence_index(cipher,i,length)<<"  ";
            sum+=Coincidence_index(cipher,i,length);
        }
        cout<<endl;
        Node x;
        x.length=length;
        x.value=(float)fabsf(0.065-(float)(sum/(float)length));
        if(x.value<=0.1)
            key.push_back(x);
        it++;
    }
    sort(key.begin(),key.end(),Greater_sort);
}
```

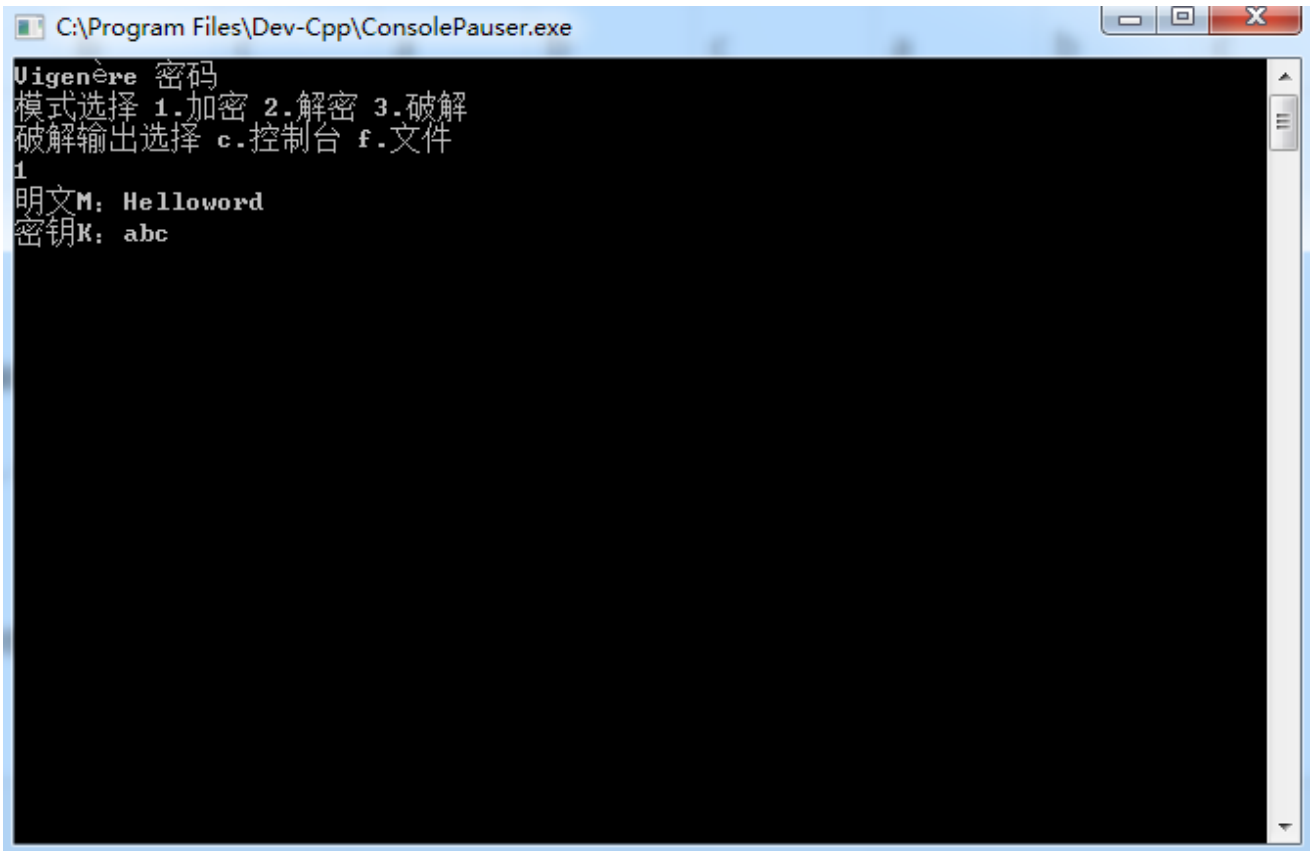
取前面10个公因子进行求解,对每个公因子的每个分子进行字母的拟重合指数分析,由Chi测试(卡方检验),获取峰值点,将结果输出到控制台或文件中

```
void Get_ans(string cipher){
    int lss=0;
    ofstream myout("temp.txt");
    while(lss<key.size()&&lss<10)
    {
        Node x=key[lss];
        int ans[cipher.length()];
        memset(ans,0,sizeof(ans));
        map< char ,int > mp;
        for(int i=0;i<x.length;i++){
            double max_pg=0.000;
            for(int k=0;k<26;k++){
                mp.clear();
                double pg=0.000;
                int sum=0;
                for(int j=i;j<cipher.length();j+=x.length){
                    char c=(char)((cipher[j]-'a'+k)%26+'a');
                    mp[c]++;
                    sum++;
                }
                for(char j='a';j<='z';j++){
                    pg+=((double)mp[j]/(double)sum)*g[j-'a'];
                }
                if(pg>max_pg){
                    ans[i]=k;
                    max_pg=pg;
                }
            }
        }
        if(flag=='c')
```

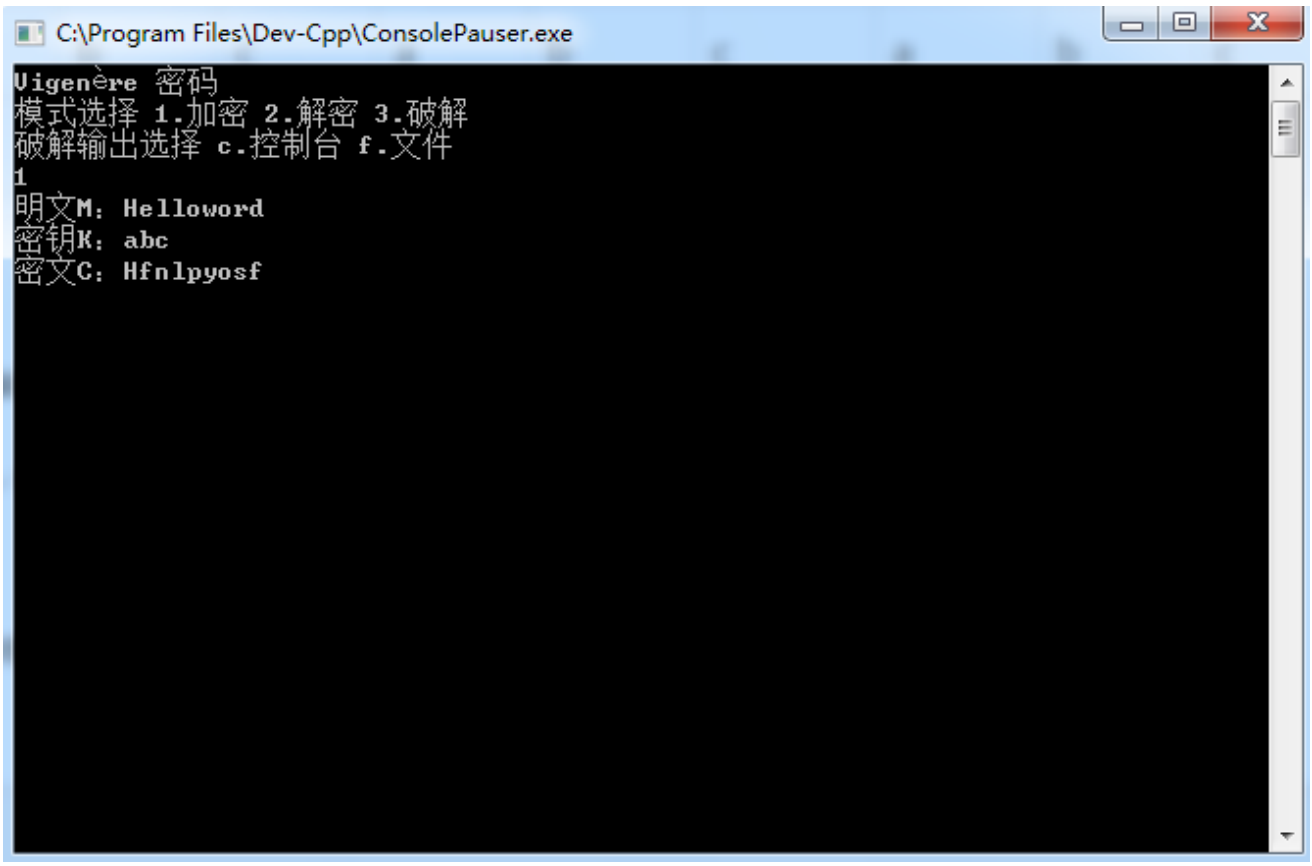
```
        {
            cout<<endl<<"key_length: "<<x.length<<endl<<"key is: ";
            for(int i=0;i<x.length;i++){
                cout<<(char)((26-ans[i])%26+'a')<<" ";
            }
            cout<<endl<<"Clear text:"<<endl;
            for(int i=0;i<cipher.length();i++){
                cout<<(char)((cipher[i]-'a'+ans[i%x.length])%26+'a');
            }
            cout<<endl;
        }
        else
        {
            myout<<endl<<"key_length: "<<x.length<<endl<<"key is: ";
            for(int i=0;i<x.length;i++){
                myout<<(char)((26-ans[i])%26+'a')<<" ";
            }
            myout<<endl<<"Clear text:"<<endl;
            for(int i=0;i<cipher.length();i++){
                myout<<(char)((cipher[i]-'a'+ans[i%x.length])%26+'a');
            }
            myout<<endl;
        }
        lss++;
    }
    myout.close();
}
```

结果

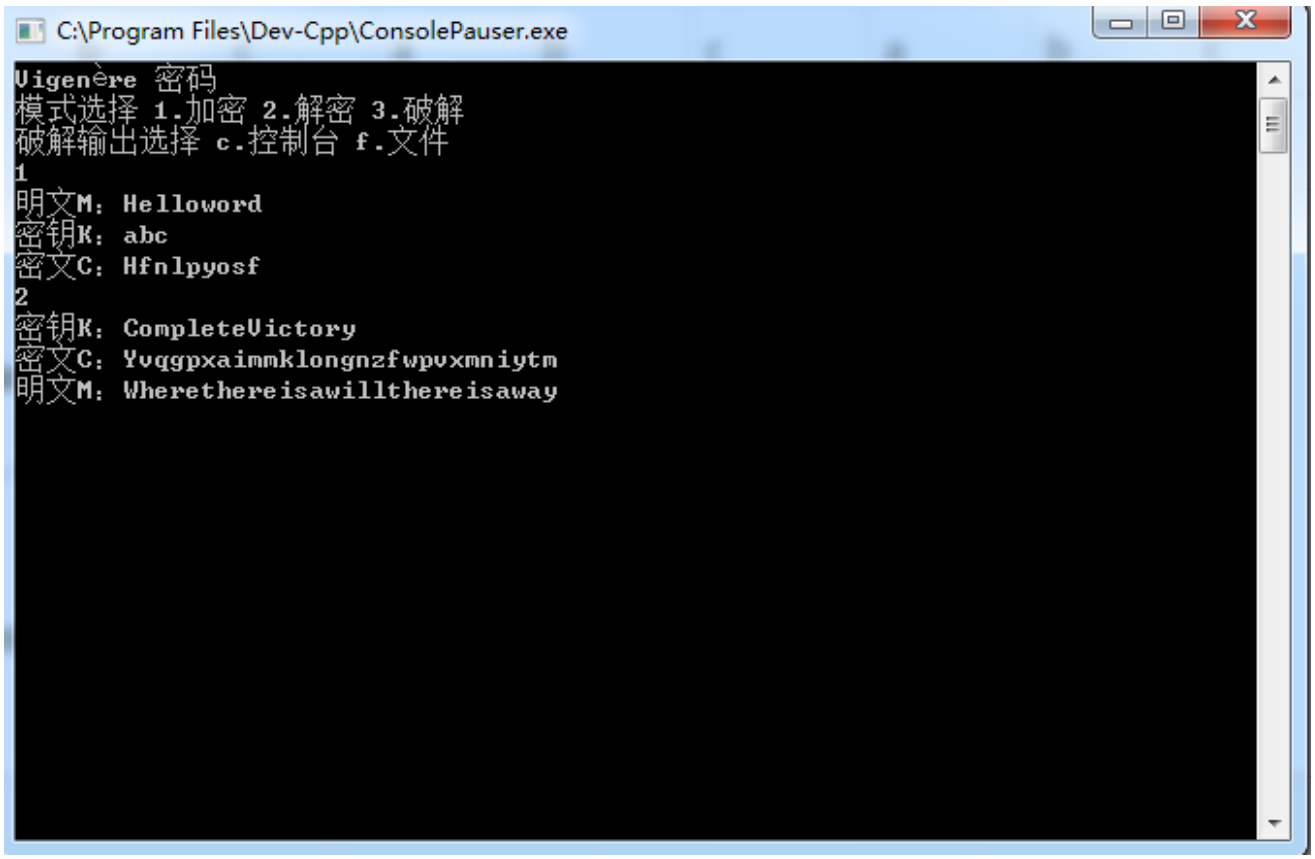
运行，选择加密模式，输入明文M和密钥K



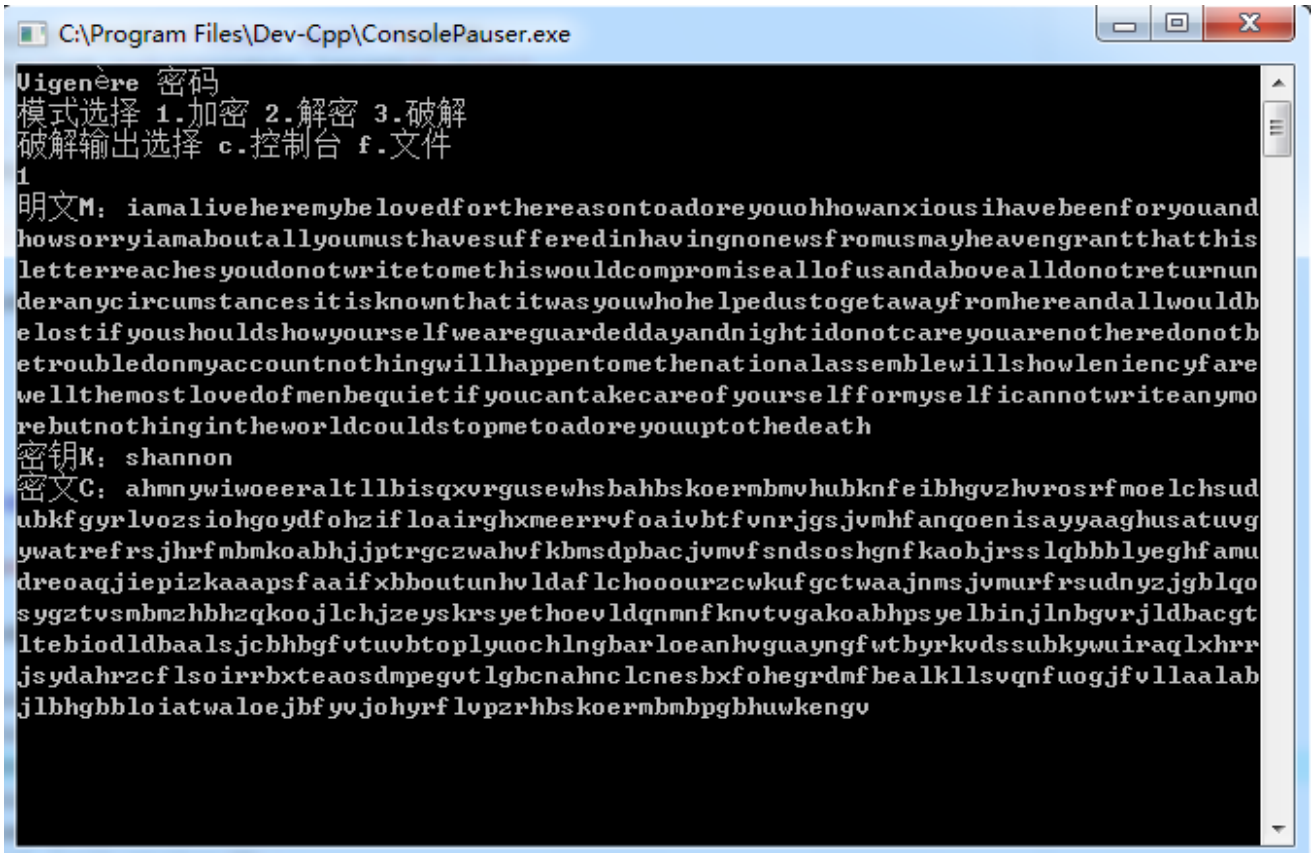
输出加密后的密文C



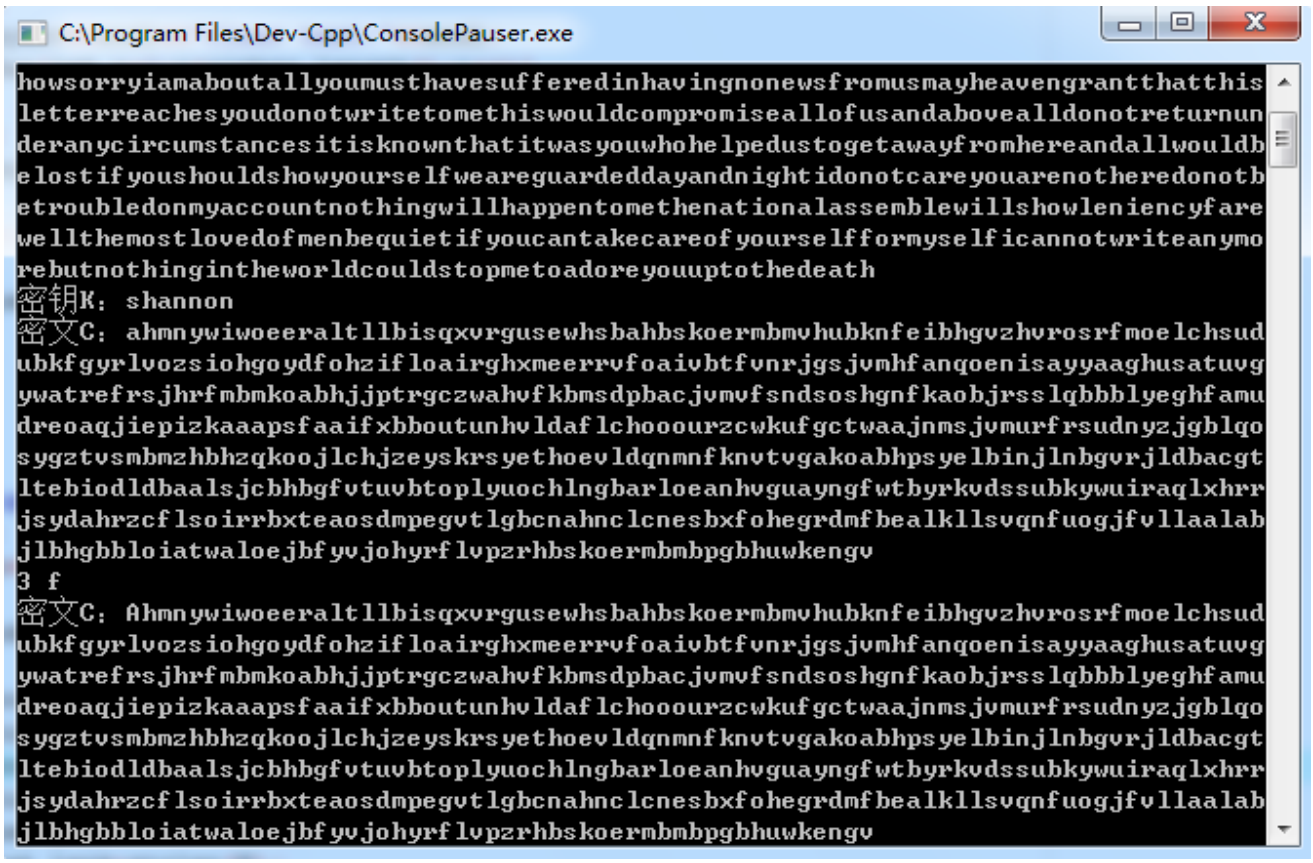
选择解密模式，输入密钥和密文



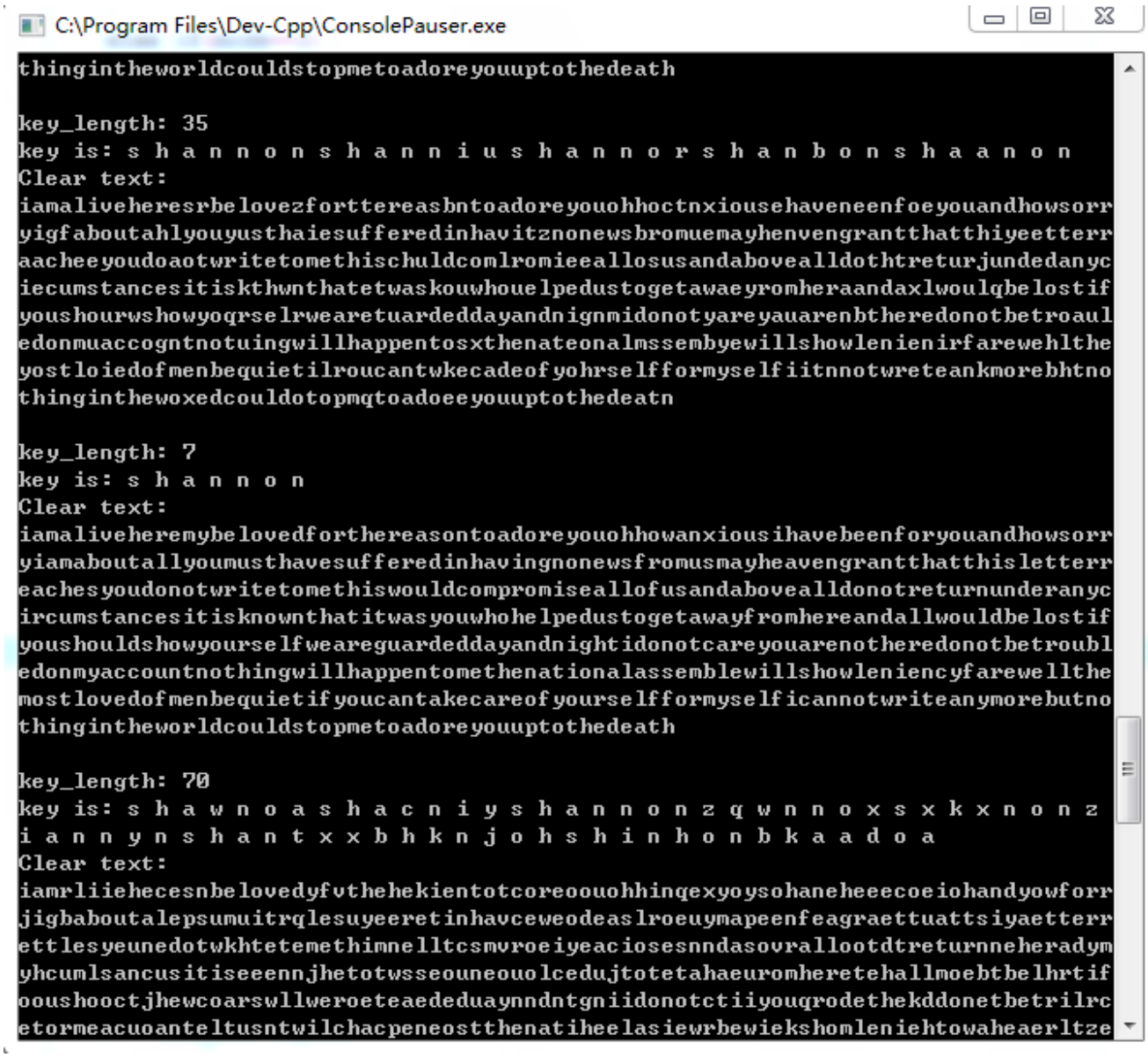
输出解密后明文



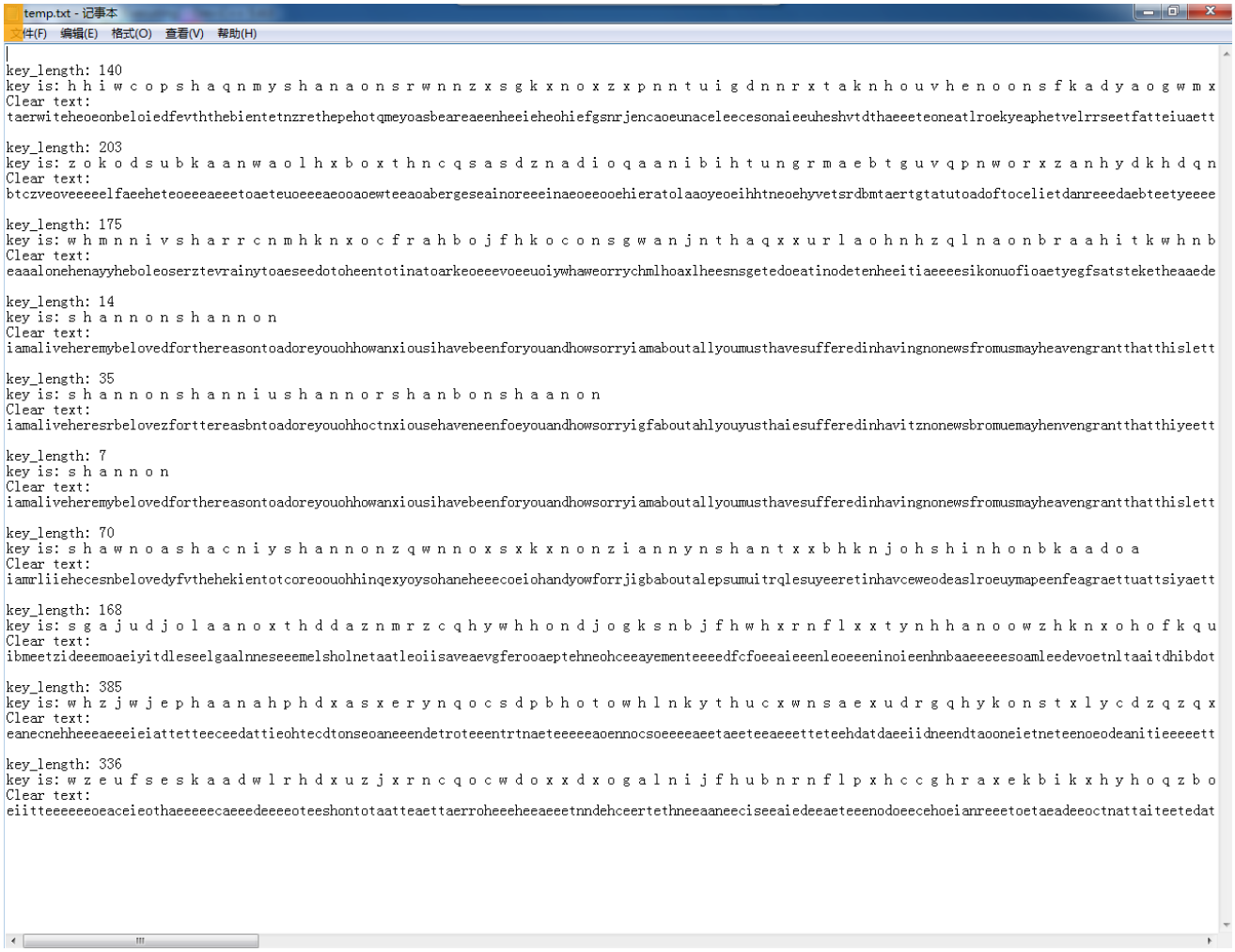
选择一个足够长的文本加密


C:\Program Files\Dev-Cpp\ConsolePauser.exe
howsorryiamaboutallyoumusthavesufferedinhavingnonewsfromusmayheavengrantthatthis
letterreachesyoudonotwritetomethiswouldcompromiseallofusandabovealldonotreturnun
deranycircumstancesitisknownthatitwasyouwhohelpedustogetawayfromhereandallwouldb
elostifyoushouldshowyourselfweareguardeddayandnightidonotcareyouarenotheredonotb
etroubledonmyaccountnothingwillhappentomethenationalassemblewillshowleniencyfare
wellthemostlovedofmenbequietifyoucantakecareofyourselfformyselficannotwriteanymo
rebutnothingintheworldcouldstopmetoadoreyouuptothedeath
密钥K：shannon
密文C：ahmnywiwoeeraltllbisqxvrgusewhsbahbskoermbmvhubknfeibhgvzhvrosrfmoelchsud
ubkfgyrlvozsiohgoydfohzifloairghxmeerrvfoaivbtfvnrjgsjvmhfanqoenisayyaaghusatuvg
ywatrefrsjhrfmbmkoabhjjptrgczwahvfkbmsdpbacjvmvfsndsoshgnfkaobjrsslqbbblyeghfamu
dreoaqjiepizkaaapsfaaifxbboutunhvldaflchooourzcwkufgctwaajnmsjvmurfrsudnyzjgblqo
sygztvsmbmzhbhzqkoojlchjzeyskrsyethoevldqnmnfknvtvgakoabhpsyelbinjlnbgvrjldbacgt
ltebiodldbaalsjcbhbgfvtuvbtoplyuochlngbarloeanhvguayngfwtbyrkvdssubkywuiraqlxhrr
jsydahrzcflsoirrbxteaosdmpegvtlgbcnahnclcnesbxfohegrdmfbealkllsvqnfuogjfvllaalab
jlbhgbbloiatwaloejbfyvjohyrflvpzrhbskoermbmbpgbhuwkengv
3 f
密文C：Ahmnywiwoeeraltllbisqxvrgusewhsbahbskoermbmvhubknfeibhgvzhvrosrfmoelchsud
ubkfgyrlvozsiohgoydfohzifloairghxmeerrvfoaivbtfvnrjgsjvmhfanqoenisayyaaghusatuvg
ywatrefrsjhrfmbmkoabhjjptrgczwahvfkbmsdpbacjvmvfsndsoshgnfkaobjrsslqbbblyeghfamu
dreoaqjiepizkaaapsfaaifxbboutunhvldaflchooourzcwkufgctwaajnmsjvmurfrsudnyzjgblqo
sygztvsmbmzhbhzqkoojlchjzeyskrsyethoevldqnmnfknvtvgakoabhpsyelbinjlnbgvrjldbacgt
ltebiodldbaalsjcbhbgfvtuvbtoplyuochlngbarloeanhvguayngfwtbyrkvdssubkywuiraqlxhrr
jsydahrzcflsoirrbxteaosdmpegvtlgbcnahnclcnesbxfohegrdmfbealkllsvqnfuogjfvllaalab
jlbhgbbloiatwaloejbfyvjohyrflvpzrhbskoermbmbpgbhuwkengv

破解结果控制台输出


C:\Program Files\Dev-Cpp\ConsolePauser.exe
thinginthe worldcouldstopmetoadoreyouuptothedeath
key_length: 35
key is: s h a n n o n s h a n n i u s h a n n o r s h a n b o n s h a a n o n
Clear text:
iamaliveheresrbelovezforttereasbntoadoreyouohhoctnxiousehaveneenfoeyouandhowsorr
yigfaboutahlyouyusthaiesufferedinhavitznonewsbromuemayhenvengrantthatthiyeetterr
aacheeyoudoaotwritetomethischuldcomlromieeallosusandabovealldothtreturjundedanyc
iecumstancesitiskthwnthatetwaskouwhouelpedustogetawaeyromheraandaxlwoulqbelostif
youshourwshowyoqrselrwearetuardeddayandnignmidonotyareyauarenbtheredonotbetroaul
edonmuaccogntnotuingwillhappentosxthenateonalmssembyewillshowlenienirfarewehlthe
yostloiedofmenbequietilroucantwkecadeofyohrselfformyselfiitnnotwreteankmorebhtno
thinginthewoxedcouldotopmqtoadoeeyouuptothedeatn
key_length: 7
key is: s h a n n o n
Clear text:
iamaliveheremybelovedforthereasontoadoreyouohhowanxiousihavebeenforyouandhowsorr
yiamaboutallyoumusthavesufferedinhavingnonewsfromusmayheavengrantthatthisletterr
eachesyoudonotwritetomethiswouldcompromiseallofusandabovealldonotreturnunderanyc
ircumstancesitisknownthatitwasyouwhohelpedustogetawayfromhereandallwouldbelostif
youshouldshowyourselfweareguardeddayandnightidonotcareyouarenotheredonotbetroubl
edonmyaccountnothingwillhappentomethenationalassemblewillshowleniencyfarewellthe
mostlovedofmenbequietifyoucantakecareofyourselfformyselficannotwriteanymorebutno
thinginthe worldcouldstopmetoadoreyouuptothedeath
key_length: 70
key is: s h a w n o a s h a c n i y s h a n n o n z q w n n o x s x k x n o n z
i a n n y n s h a n t x x b h k n j o h s h i n h o n b k a a d o a
Clear text:
iamrliiehecesnbelovedyfvthehekientotcoreoouohhinqexyoysohaneheeecoeiohandyowforr
jigbaboutalepsumuitrqlesuyeeretinhavceweodeaslroeuymapeenfeagraettuattsiyaetterr
ettlesyeunedotwkhtetemethimnelltcsmvroeiyeaciosesnndasovrallootdtreturnneheradym
yhcumlsancusitiseeennjhetotwsseouneouolcedujtotetahaeuromheretehallmoebtbelhrtif
ooushooctjhewcoarswllweroeteaededuaynndntgniidonotctiiyouqrodethekddonetbetrilrc
etormeacuoanteltusntwilchacpeneostthenatiheelasiewrbewiekshomleniehtowaheaerltze

文件输出结果



分析

维吉尼亚密码和凯撒密码相比安全性略有提升，因为等于是每个字母都使用不同的密钥进行一次凯撒变换，可以有效防止穷举攻击。

相比更复杂的以数学为基础的密码，维吉尼亚密码作为最基础的多表代换密码有非常实用的简单性和易操作性，十分方便好用。加解密也非常容易，一个加减公式就能实现。

但是攻击依旧存在，利用Kasiski 实验、字母频率分析和重合指数攻击的结合可以轻易地实现破解一个足够长的、原文有统计学规律的密文。

第一次实验使用了C++作为编程语言，主要原因是第一次课没带电脑直接用了机房的，但是Python环境安装不了==，其次C++的数据结构比较熟悉，而且作为底层语言运行速度比Python快好多，可以更迅速地实现破解中的运算和穷举。

线性反馈移位寄存器

算法

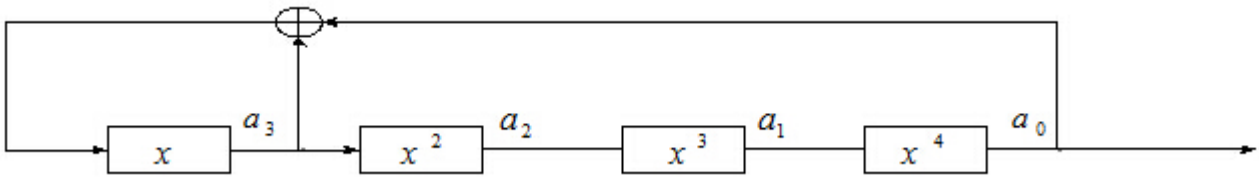
一个n级的移位寄存器产生的序列的最大周期为

$$2^n - 1$$

，当然这个最大周期跟反馈函数有很大关系，线性反馈函数实际上就是这个n级的移位寄存器选取“某些位”进行异或后得到的结果，这里的“某些位”的选取很重要，得到线性反馈函数之后，把这个移位寄存器的每次向右移动一位，把最右端的作为输出，把“某些位”的异或结果作为输入放到最左端的那位，这样所有的输出对应一个序列，这个序列叫做M序列，是最长线性移位寄存器序列的简称。

上面“某些位”的选取问题还没有解决，那么应该选取哪些位来进行异或才能保证最长周期为 2^n-1，这是一个很重要的问题。选取的“某些位”构成的序列叫做抽头序列，理论表明，要使**LFSR**得到最长的周期，这个抽头序列构成的多项式加**1**必须是一个本原多项式，也就是说这个多项式不可约，比如 $f(x)=x^4+x+1$。

例如：



如果 a_3,a_2,a_1,a_0 的值分别是**1 0 0 0**，反馈函数选取 $f(x)=x^4+x+1$，那么得到如下序列

a_3	a_2	a_1	a_0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

可以看出周长为**15**。在这一个周期里面涵盖了开区间 $[1,2^4-1]$ 内的所有整数，并且都是没有固定顺序出现的，有很好的随机性。

核心代码

不断将输入位和所有选择位异或后输出到输出位（此处直接相加看奇偶性判断异或结果）,当重复时结束程序

```
def lfsr(seed, taps):
    sr, xor = seed, 0
    while 1:
        for t in taps:
            xor += int(sr[t-1])
        if xor%2 == 0.0:
            xor = 0
        else:
            xor = 1
        #print xor
        sr, xor = str(xor) + sr[:-1], 0
        print sr
        if sr == seed:
            break
```

结果

选择a0-a4为11010，抽头位为2,5，运行`lfsr('11010', (2,5))`得到

```
PS C:\Users\lenovo> & C:/Python27/python.exe d:/==/zuoye/cry/lfsr/lfsr.py
11101
01110
10111
11011
01101
00110
00011
10001
11000
11100
11110
11111
01111
00111
10011
11001
01100
10110
01011
00101
10010
01001
00100
00010
00001
10000
01000
10100
01010
```

还有的就不截了，当再次等于11010时停止循环

分析

通过本原多项式，线性反馈移位寄存器可以生成看起来是随机的且循环周期非常长的序列，具有速度和面积优势，可产生伪随机序列、数据压缩、计数器、数据编码解码等等。

产生伪随机序列的最大长度：2^n-1，反馈多项式项数越多越难破译，安全性越高。

本次实验采用了Python2，后面的实验将都使用py2，因为py真的是很简洁又好用各种包还很强大:)

序列密码——RC4

算法

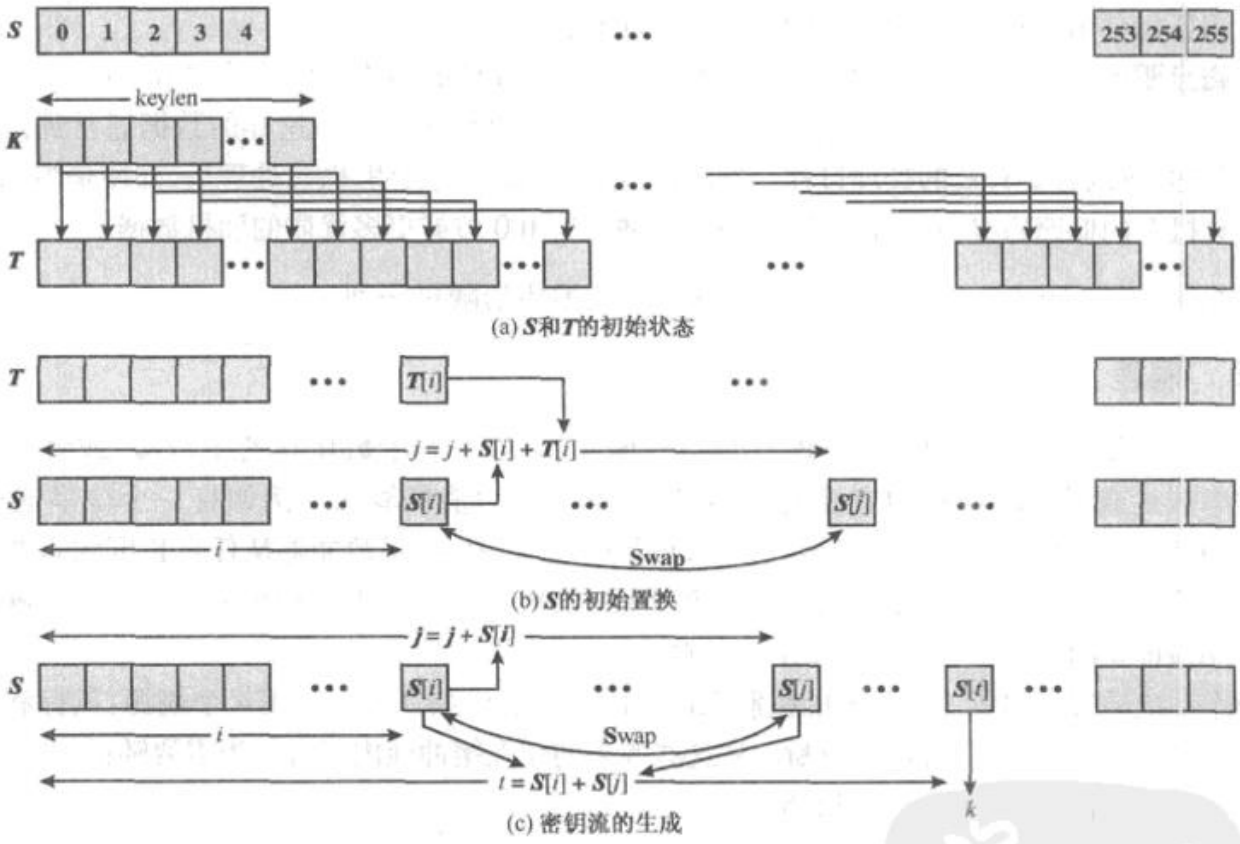


图 7.6 RC4

- 先初始化状态向量S（256个字节，用来作为密钥流生成的种子1）

  按照升序，给每个字节赋值0,1,2,3,4,5,6.....,254,255
- 初始密钥（由用户输入），长度任意

  如果输入长度小于256个字节，则进行轮转，直到填满

  由上述轮转过程得到256个字节的向量T（用来作为密钥流生成的种子2）
- 开始对状态向量S进行置换操作（用来打乱初始种子1）

  按照下列规则进行

  从第零个字节开始，执行256次，保证每个字节都得到处理

```
j = 0;
for (i = 0 ; i < 256 ; i++){
    j = (j + S[i] + T[i]) mod 256;
    swap(S[i] , S[j]);
}
```

这样处理后的状态向量S几乎是带有一定的随机性了

- 最后是秘钥流的生成与加密

  假设我的明文字节数是datalength=1024个字节（当然可以是任意个字节）

```
i=0;
j=0;
while(datalength--){//相当于执行1024次，这样生成的秘钥流也是1024个字节
      i = (i + 1) mod 256;
        j = (j + S[i]) mod 256;
        swap(S[i] , S[j]);
        t = (S[i] + S[j]) mod 256;
        k = S[t];//这里的K就是当前生成的一个秘钥流中的一位
        //可以直接在这里进行加密，当然也可以将密钥流保存在数组中，最后进行异或就ok
        data[]=data[]^k;
}
```

核心代码

S盒

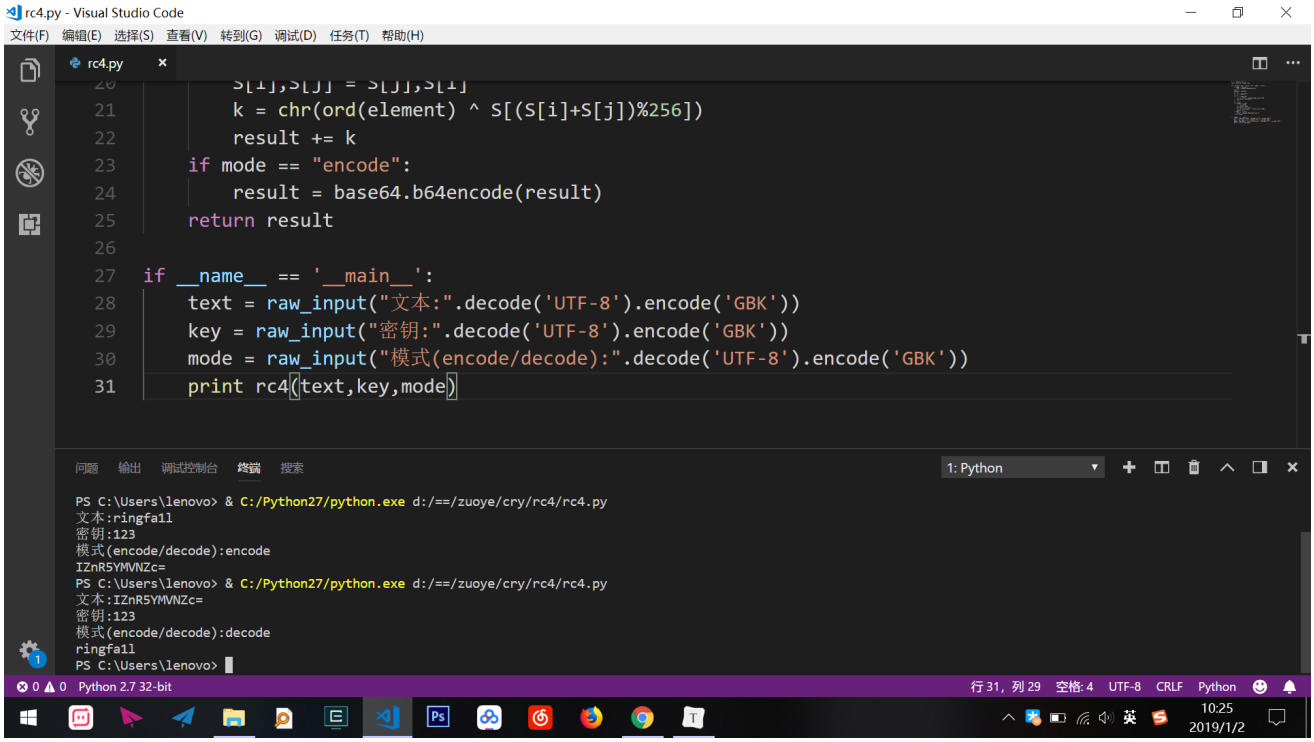
```
S = list(range(256))
j = 0
for i in range(256):
j = (j + S[i] + ord(key[i%key_len]))%256
S[i],S[j] = S[j],S[i]
```

秘钥流

```
for element in text:
i = (i+1)%256
j = (j+S[i])%256
S[i],S[j] = S[j],S[i]
k = chr(ord(element) ^ S[(S[i]+S[j])%256])
result += k
```

结果

输入文本和密钥，选择加解密得出结果（加密结果可能是无法显示的字符，所以做base64处理）



分析

不够安全：

- 偏移

  在密钥随机的前提下，如果密钥流是均匀的话，每个字节出现的概率应该是1/256=0.3906%,但从理论分析和实验结果来看，密钥流某些位置上的某些字节出现的概率要明显高于(或低于)其他字节，即偏移（biase）。

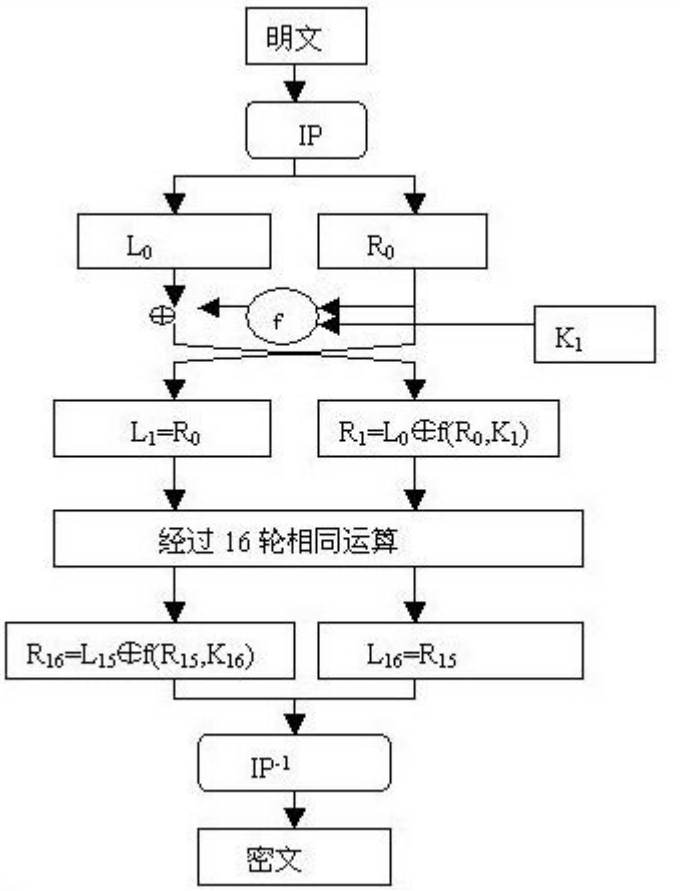
  在S盒初始化结束，生成密钥流的过程中，假设S2=0,S1=X≠2,S[X]=Y,根据伪随机数生成算法，第一轮，S[X]和S1互换，生成的密钥字节是S[X+Y];第二轮，S2和S[X]互换，生成的密钥字节是S[X]=0，即Z2=0。由条件概率公式，计算出z2=0的概率约为1/128。

- 概率

  基于概率论的相关公式，我们就可以利用RC4密钥流中的偏移特性，对明文某一字节出现的概率进行计算，从而攻破RC4。在预处理阶段，通过大量的实验，生成随机的key，统计密钥流中各字节出现的概率，类似求出前面多项展开式中的p1，p2。在获取密文阶段，通过一些手段，不断用RC4加密同样的明文，记录出现的密文.类似求出前面多项展开式中的n1，n2。最后，计算明文各个字节的概率，从概率较高的候选字节中恢复明文。

分组密码——DES

算法



- IP置换：

  将输入的64位数据块按位重新组合，并把输出分为L0、R0两部分，每部分各长32位。

- 密钥置换：

  不考虑每个字节的第8位，DES的密钥由64位减至56位，每个字节的第8位作为奇偶校验位。

  在DES的每一轮中，从56位密钥产生出不同的48位子密钥：将56位的密钥分成两部分，每部分28位；根据轮数，这两部分分别循环左移1位或2位。移动后，从56位中选出48位。这个过程中，既置换了每位的顺序，又选择了子密钥，因此称为压缩置换。

- 扩展置换：

  IP置换后获得的右半部分R0，将32位输入扩展为48位(分为4位×8组)输出。

  目的是生成与密钥相同长度的数据以进行异或运算；提供更长的结果，在后续的替代运算中可以进行压缩。

  扩展置换之后，右半部分数据R0变为48位，与密钥置换得到的轮密钥进行异或。

- S盒代替：

  压缩后的密钥与扩展分组异或以后得到48位的数据，将这个数据送人S盒，进行替代运算。替代由8个不同的S盒完成，每个S盒有6位输入4位输出。48位输入分为8个6位的分组，一个分组对应一个S盒，对应的S盒对各组进行代替操作。

  一个S盒就是一个4行16列的表，盒中的每一项都是一个4位的数。S盒的6个输入确定了其对应的输出在哪一行哪一列，输入的高低两位做为行数H，中间四位做为列数L，在S-BOX中查找第H行L列对应的数据(<32)。

  代替过程产生8个4位的分组，组合在一起形成32位数据。

- P盒置换：

  S盒代替运算的32位输出按照P盒进行置换。该置换把输入的每位映射到输出位，任何一位不能被映射两次，也不能被略去。

最后，P盒置换的结果与最初的64位分组左半部分L0异或，然后左、右半部分交换，接着开始另一轮。

- IP-1末置换：

  末置换是初始置换的逆过程，DES最后一轮后，左、右两半部分并未进行交换，而是两部分合并形成一个分组做为末置换的输入。

  置换后得到密文。

核心代码

初始化

```
def __init__(self, key, mode=ECB, IV=None, pad=None, padmode=PAD_NORMAL):
    if len(key) != 8:
        raise ValueError("Invalid DES key size. Key must be exactly 8 bytes long.")
    _baseDes.__init__(self, mode, IV, pad, padmode)
    self.key_size = 8

    self.L = []
    self.R = []
    self.Kn = [ [0] * 48 ] * 16 # 16 48-bit keys (K1 - K16)
    self.final = []

    self.setKey(key)
```

字符串转01

```
def __String_to_BitList(self, data):
    data = [ord(c) for c in data]
    l = len(data) * 8
    result = [0] * l
    pos = 0
    for ch in data:
        i = 7
        while i >= 0:
            if ch & (1 << i) != 0:
                result[pos] = 1
            else:
                result[pos] = 0
            pos += 1
            i -= 1

    return result
```

01转字符串

```
def __BitList_to_String(self, data):
    result = []
    pos = 0
    c = 0
    while pos < len(data):
        c += data[pos] << (7 - (pos % 8))
        if (pos % 8) == 7:
            result.append(c)
            c = 0
        pos += 1

    return ''.join([ chr(c) for c in result ])
```

置换变化

```
def __permutate(self, table, block):
    return list(map(lambda x: block[x], table))
```

创建子密钥

```
def __create_sub_keys(self):
    # 子密钥
    key = self.__permutate(des.__pc1, self.__String_to_BitList(self.getKey()))
    i = 0
    # 左右块
    self.L = key[:28]
    self.R = key[28:]
    while i < 16:
        j = 0
        # 循环左移
        while j < des.__left_rotations[i]:
            self.L.append(self.L[0])
            del self.L[0]

            self.R.append(self.R[0])
            del self.R[0]

            j += 1

        self.Kn[i] = self.__permutate(des.__pc2, self.L + self.R)

        i += 1
```

加密算法

```
def __des_crypt(self, block, crypt_type):
    block = self.__permutate(des.__ip, block)
    self.L = block[:32]

    self.R = block[32:]
```

```
    if crypt_type == des.ENCRYPT:  #加密从Kn[1]到Kn[16]
        iteration = 0
        iteration_adjustment = 1
    else:                          #解密相反
        iteration = 15
        iteration_adjustment = -1

    i = 0
    while i < 16:
        tempR = self.R[:]

        self.R = self.__permutate(des.__expansion_table, self.R)

        self.R = list(map(lambda x, y: x ^ y, self.R, self.Kn[iteration]))
        B = [self.R[:6], self.R[6:12], self.R[12:18], self.R[18:24], self.R[24:30],
self.R[30:36], self.R[36:42], self.R[42:]]

        # B[1] to B[8]
        j = 0
        Bn = [0] * 32
        pos = 0
        while j < 8:
            m = (B[j][0] << 1) + B[j][5]
            n = (B[j][1] << 3) + (B[j][2] << 2) + (B[j][3] << 1) + B[j][4]

            v = des.__sbox[j][(m << 4) + n]

            Bn[pos] = (v & 8) >> 3
            Bn[pos + 1] = (v & 4) >> 2
            Bn[pos + 2] = (v & 2) >> 1
            Bn[pos + 3] = v & 1

            pos += 4
            j += 1

        self.R = self.__permutate(des.__p, Bn)

        self.R = list(map(lambda x, y: x ^ y, self.R, self.L))

        self.L = tempR

        i += 1
        iteration += iteration_adjustment

    self.final = self.__permutate(des.__fp, self.R + self.L)
    return self.final
```

数据处理

```
def crypt(self, data, crypt_type):
```

```
    #检验
    if not data:
        return ''
    if len(data) % self.block_size != 0:
        if crypt_type == des.DECRYPT:
            raise ValueError("Invalid data length, data must be a multiple of " +
str(self.block_size) + " bytes\n.")
        if not self.getPadding():
            raise ValueError("Invalid data length, data must be a multiple of " +
str(self.block_size) + " bytes\n. Try setting the optional padding character")
        else:
            data += (self.block_size - (len(data) % self.block_size)) * self.getPadding()
        # print "Len of data: %f" % (len(data) / self.block_size)

    if self.getMode() == CBC:
        if self.getIV():
            iv = self.__String_to_BitList(self.getIV())
        else:
            raise ValueError("For CBC mode, you must supply the Initial Value (IV) for
ciphering")

    # 分别加密块
    i = 0
    dict = {}
    result = []
    while i < len(data):
        block = self.__String_to_BitList(data[i:i+8])

        if self.getMode() == CBC:
            if crypt_type == des.ENCRYPT:
                block = list(map(lambda x, y: x ^ y, block, iv))

            processed_block = self.__des_crypt(block, crypt_type)

            if crypt_type == des.DECRYPT:
                processed_block = list(map(lambda x, y: x ^ y, processed_block, iv))
                iv = block
            else:
                iv = processed_block
        else:
            processed_block = self.__des_crypt(block, crypt_type)


        result.append(self.__BitList_to_String(processed_block))
        i += 8

    # print "Lines: %d, cached: %d" % (lines, cached)
    return ''.join(result)
```

结果

将要加密的文本放入data.txt，密钥放入key.txt，运行脚本，选择模式为CBC或EBC，选择CBC的话继续输入IV，运行结束加密文件为encrypt.txt，解密文件为decrypt.txt。

```
550     else:
551         des = des(key, mode)
552     res = des.encrypt(data)
553     f2.write(base64.b64encode(res))
554     de = des.decrypt(res)
555     f4.write(de)
```

```
问题  输出  调试控制台  终端  搜索

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

PS C:\Users\lenovo> & C:/Python27/python.exe d:/==/zuoye/cry/des/des.py
Key       : 'secret_k'
Data      : 'Hello wo'
模式(ECB/CBC):CBC
IV:12345678
PS C:\Users\lenovo>
```

data.txt	2018/12/21 10:45	文本文档	1 KB
decrypt.txt	2019/1/2 22:41	文本文档	1 KB
des.py	2019/1/2 22:40	PY 文件	16 KB
encrypt.txt	2019/1/2 22:41	文本文档	1 KB
key.txt	2018/12/21 10:39	文本文档	1 KB
1.PNG	2019/1/2 22:46	PNG 文件	28 KB

data.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Hello wo

key.txt - 记事本
文件(F) 编辑(E) 格式(O)
secret_k

encrypt.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
JqTS1Jt8ekc=

decrypt.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Hello wo

分析

安全性：

S盒代替时DES算法的关键步骤，所有的其他的运算都是线性的，易于分析，而S盒是非线性的，相比于其他步骤，提供了更好安全性，实现较好的混淆。

互补对称性的缺点:

利用DES算法的互补对称性,利用选择明文进行穷举攻击时可将密钥的加密测试量降低一半。

- 选择两个明密对.
- 令K(0)是最低位为0的所有密钥构成的集合
- 对K(0)中的每个元k,计算c的补=Ek(m),并检验c的补=c1是否成立.若成立,则判定k为候选密钥

加强：多重DES

```
def encrypt(self, data, pad=None, padmode=None):
    ENCRYPT = des.ENCRYPT
    DECRYPT = des.DECRYPT
    data = self._guardAgainstUnicode(data)
    if pad is not None:
        pad = self._guardAgainstUnicode(pad)
    data = self._padData(data, pad, padmode)
    if self.getMode() == CBC:
        self.__key1.setIV(self.getIV())
        self.__key2.setIV(self.getIV())
        self.__key3.setIV(self.getIV())
        i = 0
        result = []
        while i < len(data):
            block = self.__key1.crypt(data[i:i+8], ENCRYPT)
            block = self.__key2.crypt(block, DECRYPT)
            block = self.__key3.crypt(block, ENCRYPT)
            self.__key1.setIV(block)
            self.__key2.setIV(block)
            self.__key3.setIV(block)
            result.append(block)
            i += 8
        return ''.join(result)
    else:
        data = self.__key1.crypt(data, ENCRYPT)
        data = self.__key2.crypt(data, DECRYPT)
        return self.__key3.crypt(data, ENCRYPT)
```

- 密钥长度增加到112位或168位，可以有效克服穷举搜索攻击
- 相对于DES，增强了抗差分分析和线性分析的能力
- 具备继续使用现有的DES实现的可能
- 处理速度相对较慢，特别是对于软件实现
- 明文分组的长度仍为64位，就效率和安全性而言，与密钥的增长不相匹配

哈希函数实现消息认证

算法

基于sha1

- 把原始消息（字符串，文件等）转换成位字符串
- 消息必须进行补位，先补一个1，然后再补0，直到长度满足对512取模后余数是448。总而言之，补位是至少补一位，最多补512位。
- 将原始数据的长度补到已经进行了补位操作的消息后面。如果原始的消息长度超过了512，我们需要将它补成512的倍数。然后我们把整个消息分成一个一个512位的数据块，分别处理每一个数据块，从而得到消息摘要。
- 按顺序处理 Mi,进行下面的步骤

  (1) 将 Mi 分成 16 个字 W0, W1, ... , W15, W0 是最左边的字

  (2) 对于 t = 16 到 79 令 Wt = S1(Wt-3 XOR Wt-8 XOR Wt- 14 XOR Wt-16).

(3) 令 A = H0, B = H1, C = H2, D = H3, E = H4.

(4) 对于 t = 0 到 79，执行下面的循环

TEMP = S5(A) + ft(B,C,D) + E + Wt + Kt;

E = D; D = C; C = S30(B); B = A; A = TEMP;

(5) 令 H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E. 在处理完所有的 Mi, 后，消息摘要是一个160位的字符串，以下面的顺序标识

H0 H1 H2 H3 H4

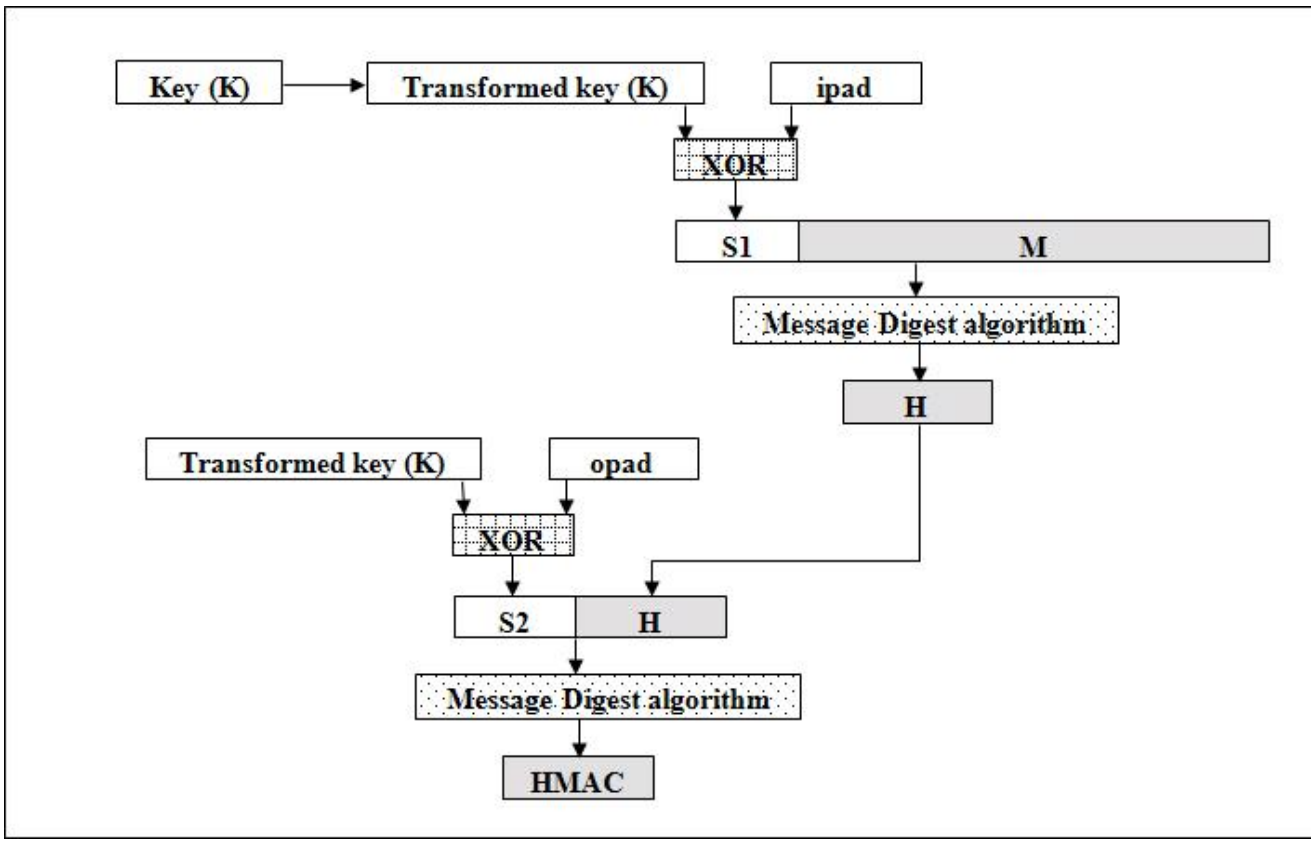
定义HMAC需要一个加密用散列函数（表示为H）和一个密钥K

假设H是一个将数据块用一个基本的迭代压缩函数来加密的散列函数。

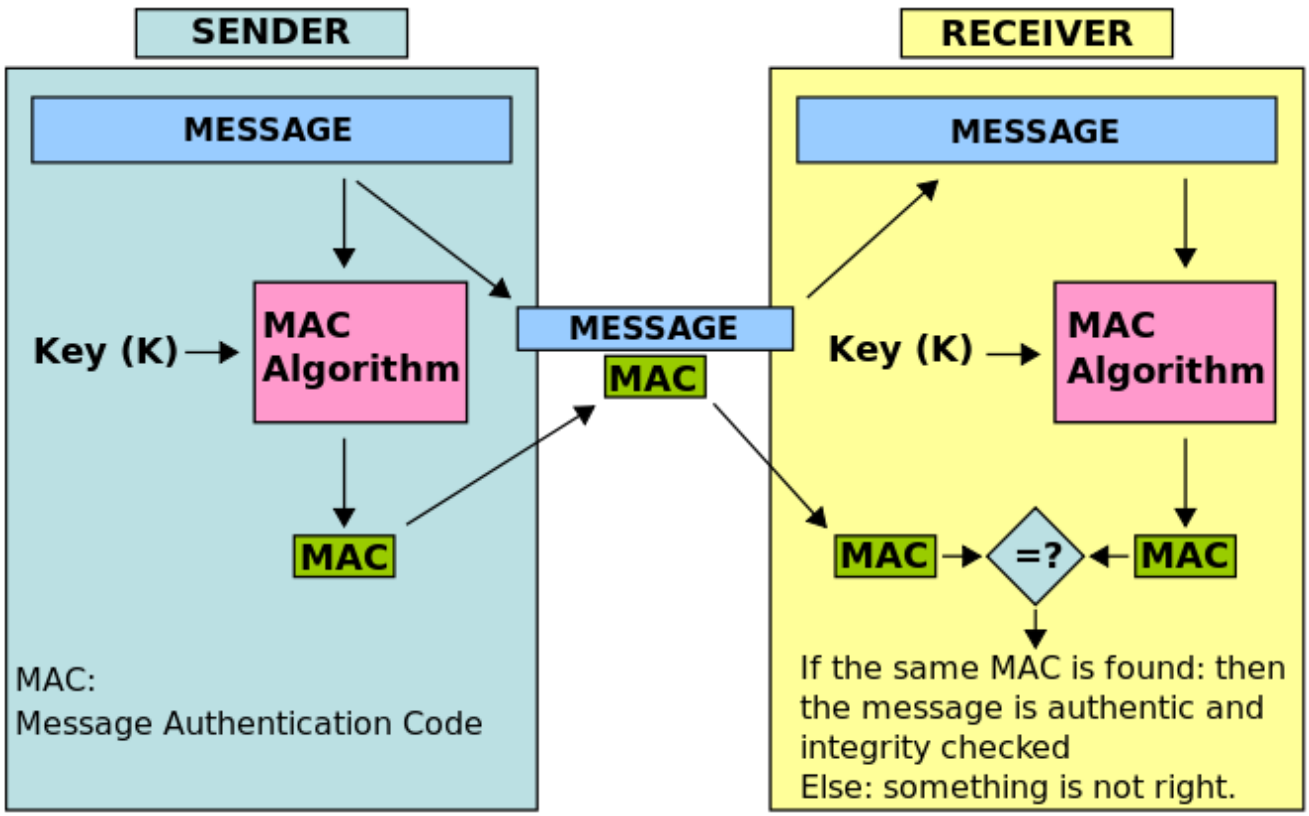
用B来表示数据块的字长。（以上说提到的散列函数的分割数据块字长B=64），用L来表示散列函数的输出数据字长（MD5中L=16(128位),SHA—1中L=20(160位)）。鉴别密钥的长度可以是小于等于数据块字长的任何正整数值。应用程序中使用的密钥长度若是比B大，则首先用使用散列函数H作用于它，然后用H输出的L长度字符串作为在HMAC中实际使用的密钥。一般情况下，推荐的最小密钥K长度是L个字长。（与H的输出数据长度相等）。

我们将定义两个固定且不同的字符串ipad,opad：（'i','o'标志内部与外部） `ipad = the byte 0x36 repeated B times` `opad = the byte 0x5C repeated B times` 计算'text'的HMAC：`H( K XOR opad, H(K XOR ipad, text))` 即为以下步骤:

- 在密钥K后面添加0来创建一个子长为B的字符串。(例如，如果K的字长是20字节，B = 64字节，则K后会加入44个零字节0x00)
- 将上一步生成的B字长的字符串与ipad做异或运算。
- 将数据流text填充至第二步的结果字符串中。
- 用H作用于第三步生成的数据流。
- 将第一步生成的B字长字符串与opad做异或运算。
- 再将第四步的结果填充进第五步的结果中。
- 用H作用于第六步生成的数据流，输出最终结果



通过散列算法可实现数字签名实现，数字签名的原理是将要传送的明文通过一种函数运算（Hash）转换成报文摘要（不同的明文对应不同的报文摘要），报文摘要加密后与明文一起传送给接收方，接收方将接收的明文产生新的报文摘要与发送方的发来报文摘要解密比较，比较结果一致表示明文未被改动，如果不一致表示明文已被篡改。



核心代码

sha1基础类

```
class Hasha1(object):
    dig = 20
    block = 64
    def __init__(self):
        self._h = (
           0x67452301,
           0xEFCDAB89,
           0x98BADCFE,
           0x10325476,
           0xC3D2E1F0,
          )
        self.remaining = b''
        self.bytes = 0

    def hash(self, arg):
        chunk = self.remaining + arg[:64]
        while len(chunk) == 64:
            self._h = process_chunk(chunk, *self._h)
            self.bytes += 64
            chunk = arg[:64]
            arg = arg[64:]
        self.remaining = chunk
        return self

    def getHex(self):
        return'%08x%08x%08x%08x%08x' % self.hashBlock()

    def hashBlock(self):
        message = self.remaining
        fullBytes = self.bytes + len(message)
        message += b'\x80'
        message += b'\x80' * ((56 - (fullBytes + 1) % 64) % 64)
        message_bit_length = fullBytes * 8
        message += struct.pack(b'>Q', message_bit_length)
        h = process_chunk(message[:64], *self._h)
        if len(message) == 64:
            return h
        return process_chunk(message[64:], *h)
```

算法实现

```
def process_chunk(chunk, h0, h1, h2, h3, h4):
    assert len(chunk) == 64
    w = [0] * 80
    for i in range(16):
        w[i] = struct.unpack(b'>I', chunk[i * 4:i * 4 + 4])[0]
    for i in range(16, 80):
        w[i] = (w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16])
        w[i] = left_rotate(w[i], 1)

    a = h0
```

```
    b = h1
    c = h2
    d = h3
    e = h4
    for i in range(80):
        if 0 <= i <= 19:
            f = d ^ (b & (c ^ d))
            k = 0x5A827999
        elif 20 <= i <= 39:
            f = b ^ c ^ d
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
            f = (b & c) | (b & d) | (c & d)
            k = 0x8F1BBCDC
        elif 60 <= i <= 79:
            f = b ^ c ^ d
            k = 0xCA62C1D6
        a, b, c, d, e = ((left_rotate(a, 5) + f + e + k + w[i]) & 0xffffffff,
                         a, left_rotate(b, 30), c, d)
    h0 = (h0 + a) & 0xffffffff
    h1 = (h1 + b) & 0xffffffff
    h2 = (h2 + c) & 0xffffffff
    h3 = (h3 + d) & 0xffffffff
    h4 = (h4 + e) & 0xffffffff
    return h0, h1, h2, h3, h4
```

实现hmac函数

```
def hmac(key, m):
    key2 = ""
    for i in range(0, 10):
        key2 = key2 + str((int(key)/pow(2, i)) % 2)
    block = 64
    out = 20
    if len(key2) > block:
        key2 = sha1(m)
    if len(key2) < block:
        key2 = addPad(key2,block)

    tbl1 = string.maketrans(trans_5C, trans_5C)
    tbl2 = string.maketrans(trans_36, trans_36)

    o_key_pad = key2.translate(tbl1)
    i_key_pad = key2.translate(tbl2)

    preIn = i_key_pad + m
    inner = sha1(preIn)
    preOut = o_key_pad + inner
    outer = sha1(preOut)

    return outer
```

结果

运行输入text和key，产生len为40的hmac串

```
169
170 if __name__ == "__main__":
171     main()
```

```
问题  输出  调试控制台  终端  搜索

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

PS C:\Users\lenovo> & C:/Python27/python.exe d:/==/zuoye/cry/hash/hmacsha1.py
text:ringfa11
key(number):233
HMAC:  78dc8db79bfccac555d284828bb101a368c9c602
len:  40
PS C:\Users\lenovo> & C:/Python27/python.exe d:/==/zuoye/cry/hash/hmacsha1.py
text:ringfa11
key(number):322
HMAC:  44c197e5052b572d40970f25ab9bc93e5fd37aac
len:  40
PS C:\Users\lenovo>
```

分析

sha1与md5比较

- 对强行攻击的安全性：最显著和最重要的区别是SHA-1摘要比MD5摘要长32位。使用强行技术，产生任何一个报文使其摘要等于给定报摘要的难度对MD5是2^128数量级的操作，而对SHA-1则是2^160数量级的操作。这样，SHA-1对强行攻击有更大的强度。
- 对密码分析的安全性：由于MD5的设计，易受密码分析的攻击，SHA-1显得不易受这样的攻击。
- 速度：在相同的硬件上，SHA-1的运行速度比MD5慢。

HMAC加密算法是一种安全的基于加密hash函数和共享密钥的消息认证协议．它可以有效地防止数据在传输过程中被截获和篡改，维护了数据的完整性、可靠性和安全性。HMAC加密算法是一种基于密钥的报文完整性的验证方法，其安全性是建立在Hash加密算法基础上的。它要求通信双方共享密钥、约定算法、对报文进行Hash运算，形成固定长度的认证码。通信双方通过认证码的校验来确定报文的合法性。HMAC加密算法可以用来作加密、数字签名、报文验证等。

HMAC安全性：HMAC加密算法引入了密钥，其安全性已经不完全依赖于所使用的HASH算法，安全性主要有以下几点保证：

- 使用的密钥是双方事先约定的，第三方不可能知道。由上面介绍应用流程可以看出，作为非法截获信息的第三方，能够得到的信息只有作为“挑战”的随机数和作为“响应”的HMAC结果，无法根据这两个数据推算出密钥。由于不知道密钥，所以无法仿造出一致的响应。
- 在HMAC加密算法的应用中，第三方不可能事先知道输出（如果知道，不用构造输入，直接将输出送给服务器即可）。
- HMAC加密算法与一般的加密重要的区别在于它具有“瞬时”性，即认证只在当时有效，而加密算法被破解后，以前的加密结果就可能被解密。

非对称加密——RSA

算法

数学基础：

- 如果两个正整数，除了1以外，没有其他公因子，我们就称这两个数是互质关系（coprime）。

  任意两个质数构成互质关系，比如13和61。

  一个数是质数，另一个数只要不是前者的倍数，两者就构成互质关系，比如3和10。

  如果两个数之中，较大的那个数是质数，则两者构成互质关系，比如97和57。

  1和任意一个自然数是都是互质关系，比如1和99。

  p是大于1的整数，则p和p-1构成互质关系，比如57和56。

  p是大于1的奇数，则p和p-2构成互质关系，比如17和15。

- 任意给定正整数n，请问在小于等于n的正整数之中，有多少个与n构成互质关系?

  计算这个值的方法就叫做欧拉函数，以φ(n)表示

  通用公式：

$$\phi(n)=n(1-\frac{1}{p_1})(1-\frac{1}{p_2})...(1-\frac{1}{p_r})$$

- 欧拉定理：如果两个正整数a和n互质，则n的欧拉函数 φ(n) 可以让下面的等式成立：

$$a^{\phi(n)}\equiv 1(mod\ n)$$

- 费马小定理：假设正整数a与质数p互质，因为质数p的φ(p)等于p-1，则欧拉定理可以写成

$$a^{p-1}\equiv 1\ (mod\ p)$$

- 模反元素：如果两个正整数a和n互质，那么一定可以找到整数b，使得 ab-1 被n整除，或者说ab被n除的余数是1。

$$ab\equiv 1\ (mod\ n)$$

过程

- 随机选择两个不相等的质数p和q
- 计算p和q的乘积n
- 计算n的欧拉函数φ(n)
- 随机选择一个整数e，条件是1< e < φ(n)，且e与φ(n) 互质
- 计算e对于φ(n)的模反元素d（用"扩展欧几里得算法"求解）
- 将n和e封装成公钥，n和d封装成私钥
- 加密：$m^e \equiv c\ (mod\ n)$
- 解密：$c^d \equiv m\ (mod\ n)$

核心代码

欧几里得算法确定最大公约数

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

求逆

```
def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi

    while e > 0:
        temp1 = temp_phi/e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2

        x = x2- temp1* x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y

    if temp_phi == 1:
        return d + phi
```

快速判断素数

```
def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in xrange(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True
```

产生密钥

```
def generate_keypair(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')
```

```
    n = p * q

    phi = (p-1) * (q-1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)

    d = multiplicative_inverse(e, phi)

    return ((e, n), (d, n))
```

加解密

```
def encrypt(pk, plaintext):
    key, n = pk
    cipher = [(ord(char) ** key) % n for char in plaintext]
    return cipher

def decrypt(pk, ciphertext):
    key, n = pk
    plain = [chr((char ** key) % n) for char in ciphertext]
    return ''.join(plain)
```

结果

输入两个不同素数，生成公钥和私钥，输入信息利用公钥加密输出密文，密文用私钥解密输出明文

```
75  if __name__ == '__main__':
76      p = int(raw_input("Enter a prime number: "))
77      q = int(raw_input("Enter another prime number: "))
78      public, private = generate_keypair(p, q)
79      print "Your public key is ", public ," and your private key is ", private
80      message = raw_input("Enter a message: ")
81      encrypted_msg = encrypt(public, message)
82      print "Your encrypted message is: "
83      print ''.join(map(lambda x: str(x), encrypted_msg))
84      print "Your message is:"
85      print decrypt(private, encrypted_msg)
```

```
问题  输出  调试控制台  终端  搜索                                    2: Python

PS C:\Users\lenovo> & C:/Python27/python.exe d:/==/zuoye/cry/rsa/rsa.py
Enter a prime number: 13
Enter another prime number: 17
Your public key is  (91, 221)  and your private key is  (211, 221)
Enter a message: ringfall
Your encrypted message is:
23921891031191765656
Your message is:
ringfall
PS C:\Users\lenovo>
```

分析

证明正确性：

$$c^d \equiv m \pmod{n}$$

根据加密规则 `m^e ≡ c (mod n)`

于是，c可以写成： `c = m^e - kn`

将c代入要我们要证明的那个解密规则： `(m^e - kn)^d ≡ m (mod n)`

它等同于求证 `m^ed ≡ m (mod n)`

由于 `ed ≡ 1 (mod ϕ(n))`

所以 `ed = hϕ(n)+1`

将ed代入： `m^hϕ(n)+1 ≡ m (mod n)`

分成两种情况证明上面这个式子:

- m与n互质：根据欧拉定理，此时 `m^ϕ(n) ≡ 1 (mod n)`，得到 `(m^ϕ(n))^h × m ≡ m (mod n)`，原式得到证明。
- m与n不是互质关系：此时，由于n等于质数p和q的乘积，所以m必然等于kp或kq。

  以 m = kp为例，考虑到这时k与q必然互质，则根据欧拉定理，下面的式子成立： `(kp)^(q-1) ≡ 1 (mod q)`，进一步得到 `[(kp)^(q-1)]^h(p-1) × kp ≡ kp (mod q)` 即 `(kp)^ed ≡ kp (mod q)`，将它改写成下面的等式 `(kp)^ed = tq + kp`，这时t必然能被p整除，即 t=t'p，`(kp)^ed = t'pq + kp`，因为 m=kp，n=pq，所以 `m^ed ≡ m (mod n)`，原式得到证明。

安全性：

- ed≡1 (mod φ(n))。只有知道e和φ(n)，才能算出d。
- φ(n)=(p-1)(q-1)。只有知道p和q，才能算出φ(n)。
- n=pq。只有将n因数分解，才能算出p和q。

结论：如果n可以被因数分解，d就可以算出，也就意味着私钥被破解。

可是，大整数的因数分解，是一件非常困难的事情。目前，除了暴力破解，还没有发现别的有效方法。维基百科这样写道：

> 对极大整数做因数分解的难度决定了RSA算法的可靠性。换言之，对一极大整数做因数分解愈困难，RSA算法愈可靠。
>
> 假如有人找到一种快速因数分解的算法，那么RSA的可靠性就会极度下降。但找到这样的算法的可能性是非常小的。今天只有短的RSA密钥才可能被暴力破解。到2008年为止，世界上还没有任何可靠的攻击RSA算法的方式。
>
> 只要密钥长度足够长，用RSA加密的信息实际上是不能被解破的。"

攻击：

- 暴力分解 N（在 N 的比特位数小于 512 的时候，可以采用大整数分解的策略获取 p 和 q）
- p & q 不当分解 N
  - 当 p-q 很大时，一定存在某一个参数较小，这里我们假设为 p，那么我们可以通过穷举的方法对模数进行试除，从而分解模数，得到保密参数与明文信息。
  - |p-q| 较小

首先

$$\frac{(p+q)^2}{4}-n=\frac{(p+q)^2}{4}-pq=\frac{(p-q)^2}{4}$$

既然 |p-q| 较小，那么 $\frac{(p-q)^2}{4}$ 自然也比较小，进而 $\frac{(p+q)^2}{4}$ 只是比 N 稍微大一点，所以 $\frac{p+q}{2}$ 与 $\sqrt{n}$ 相近。那么我们可以按照如下方法来分解

- 顺序检查 $\sqrt{n}$ 的每一个整数 x，直到找到一个 x 使得 x^2-n 是平方数，记为 y^2
- 那么 $x^2-n=y^2$，进而根据平方差公式即可分解 N

- 共模攻击(当两个用户使用相同的模数 N、不同的私钥时，加密同一明文消息时即存在共模攻击)

  设两个用户的公钥分别为 e_1 和 e_2，且两者互质。明文消息为 m，密文分别为：

$$c_1=m^{e_1}\bmod N$$
$$c_2=m^{e_2}\bmod N$$

  当攻击者截获 c_1 和 c_2 后，就可以恢复出明文。用扩展欧几里得算法求出 $re_1+se_2=1 \bmod n$ 的两个整数 r 和 s，由此可得：

$$\begin{aligned}c_1^r c_2^s &\equiv m^{re_1}m^{se_2} \bmod n\\ &\equiv m^{(re_1+se_2)} \bmod n\\ &\equiv m \bmod n\end{aligned}$$

- 小公钥指数攻击(e 特别小，比如 e 为 3)

  假设用户使用的密钥 $e=3$。考虑到加密关系满足：

$$c\equiv m^3 \bmod N$$

  则：

$$m^3=c+k\times N$$
$$m=\sqrt[3]{c+k\times n}$$

  攻击者可以从小到大枚举 k，依次开三次根，直到开出整数为止。

- d 泄露攻击(可以解密所有加密的消息,我们甚至还可以对模数 N 进行分解)

我们知道 $ed \equiv 1 \bmod \varphi(n)$，那么 $\varphi(n)|k = ed - 1$。显然 k 是一个偶数，我们可以令 $k = 2^t r$，其中 r 为奇数，t 不小于 1。那么对于任何的与 N 互素的数 g，我们都有 $g^k \equiv 1 \bmod n$。那么 $z = g^{\frac{k}{2}}$ 是模 N 的二次方根。那么我们有

$$z^2 \equiv 1 \bmod p$$
$$z^2 \equiv 1 \bmod q$$

进而我们我们知道方程有以下四个解，前两个是

$$x \equiv \pm 1 \bmod N$$

后两个是 $\pm x$，其中 x 满足以下条件

$$x \equiv 1 \bmod p$$
$$x \equiv -1 \bmod q$$

显然，$z = g^{\frac{k}{2}}$ 满足的是后面那个条件，我们可以计算 $gcd(z-1, N)$ 来对 N 进行分解。

- Wiener's Attack(在 d 比较小（d<13N14d<13N14）时，攻击者可以使用Wiener's Attack来获得私钥)
- Basic Broadcast Attack(如果一个用户使用同一个加密指数 e 加密了同一个密文，并发送给了其他 e 个用户。那么就会产生广播攻击)

  这里我们假设 e 为 3，并且加密者使用了三个不同的模数 n_1, n_2, n_3 给三个不同的用户发送了加密后的消息 m，如下

  $$c_1 = m^3 \bmod n_1$$
  $$c_2 = m^3 \bmod n_2$$
  $$c_3 = m^3 \bmod n_3$$

  这里我们假设 n_1, n_2, n_3 互素，不然，我们就可以直接进行分解，然后得到 d，进而然后直接解密。

  同时，我们假设 $m < n_i, 1 \le i \le 3$。如果这个条件不满足的话，就会使得情况变得比较复杂，这里我们暂不讨论。

  既然他们互素，那么我们可以根据中国剩余定理，可得$m^3 \equiv C \bmod n_1 n_2 n_3$。

  此外，既然 $m < n_i, 1 \le i \le 3$，那么我们知道 $m^3 < n_1 n_2 n_3$ 并且 $C < m^3 < n_1 n_2 n_3$，那么 $m^3 = C$，我们对 C 开三次根即可得到 m 的值。

  对于较大的 e 来说，我们只是需要更多的明密文对。
- Related Message Attack

当 Alice 使用同一公钥对两个具有某种线性关系的消息 M1 与 M2 进行加密，并将加密后的消息 C1，C2 发送给了 Bob 时，我们就可能可以获得对应的消息 M1 与 M2。这里我们假设模数为 N，两者之间的线性关系如下

$$M_1 \equiv f(M_2) \bmod N$$

其中 f 为一个线性函数，比如说 $f = ax + b$。

在具有较小错误概率下的情况下，其复杂度为 $O(elog^2N)$。

首先，我们知道 $C_1 \equiv M_1^e \bmod N$，并且 $M_1 \equiv f(M_2) \bmod N$，那么我们可以知道 M_2 是 $f(x)^e \equiv C_1 \bmod N$ 的一个解，即它是方程 $f(x)^e - C_1$ 在模 N 意义下的一个根。同样的，M_2 是 $x^e - C_2$ 在模 N 意义下的一个根。所以说 $x - M_2$ 同时整除以上两个多项式。因此，我们可以求得两个多项式的最大公因子，如果最大公因子恰好是线性的话，那么我们就求得了 M_2。需要注意的是，在 $e = 3$ 的情况下，最大公因子一定是线性的。

这里我们关注一下 $e = 3$，且 $f(x) = ax + b$ 的情况。首先我们有

$$C_1 \equiv M_1^3 \bmod N, M_1 \equiv aM_2 + b \bmod N$$

那么我们有

$$C_1 \equiv (aM_2 + b)^3 \bmod N, C_2 \equiv M_2^3 \bmod N$$

我们需要明确一下我们想要得到的是消息 m，所以需要将其单独构造出来。

首先，我们有式 1

$$(aM_2 + b)^3 = a^3M_2^3 + 3a^2M^2b + 3aM_2b^2 + b^3$$

再者我们构造如下式 2

$$(aM_2)^3 - b^3 \equiv (aM_2 - b)(a^2M_2^2 + aM_2b + b^2) \bmod N$$

根据式 1 我们有

$$a^3M_2^3 - 2b^3 + 3b(a^2M_2^2 + aM_2b + b^2) \equiv C_1 \bmod N$$

继而我们有式 3

$$3b(a^2M_2^2 + aM_2b + b^2) \equiv C_1 - a^3C_2 + 2b^3 \bmod N$$

那么我们根据式 2 与式 3 可得

$$(a^3C_2 - b^3) * 3b \equiv (aM_2 - b)(C_1 - a^3C_2 + 2b^3) \bmod N$$

进而我们有

$$aM_2 - b = \frac{3a^3bC_2 - 3b^4}{C_1 - a^3C_2 + 2b^3}$$

进而

$$aM_2 \equiv \frac{2a^3bC_2 - b^4 + C_1b}{C_1 - a^3C_2 + 2b^3}$$

进而

$$M_2 \equiv \frac{2a^3bC_2 - b^4 + C_1b}{aC_1 - a^4C_2 + 2ab^3} = \frac{b}{a}\frac{C_1 + 2a^3C_2 - b^3}{C_1 - a^3C_2 + 2b^3}$$

上面的式子中右边所有的内容都是已知的内容，所以我们可以直接获取对应的消息。

- 选择明密文攻击

  假设爱丽丝创建了密文 $C = P^e \bmod n$ 并且把 C 发送给鲍勃，同时假设我们要对爱丽丝加密后的任意密文解密，而不是只解密 C，那么我们可以拦截 C，并运用下列步骤求出 P：

  1. 选择任意的 $X \in Z_n^*$，即 X 与 N 互素
  2. 计算 $Y = C \times X^e \bmod n$
  3. 由于我们可以进行选择密文攻击，那么我们求得 Y 对应的解密结果 $Z = Y^d$
  4. 那么，由于 $Z = Y^d = (C \times X^e)^d = C^dX = P^{ed}X = PX \bmod n$，由于 X 与 N 互素，我们很容易求得相应的逆元，进而可以得到 P

- 侧信道攻击(攻击者可获取与加解密相关的侧信道信息，例如能量消耗、运算时间、电磁辐射等等)

能量分析攻击（侧信道攻击）是一种能够从密码设备中获取秘密信息的密码攻击方法．与其 他攻击方法不同：这种攻击利用的是密码设备的能量消耗特征，而非密码算法的数学特性．能量分析攻击是一种非入侵式攻击，攻击者可以方便地购买实施攻击所需要的设备：所以这种攻击对智能卡之类的密码设备的安全性造成了严重威胁。

能量分析攻击分为：

简单能量分析攻击（SPA），即对能量迹进行直观分析，肉眼看即可。

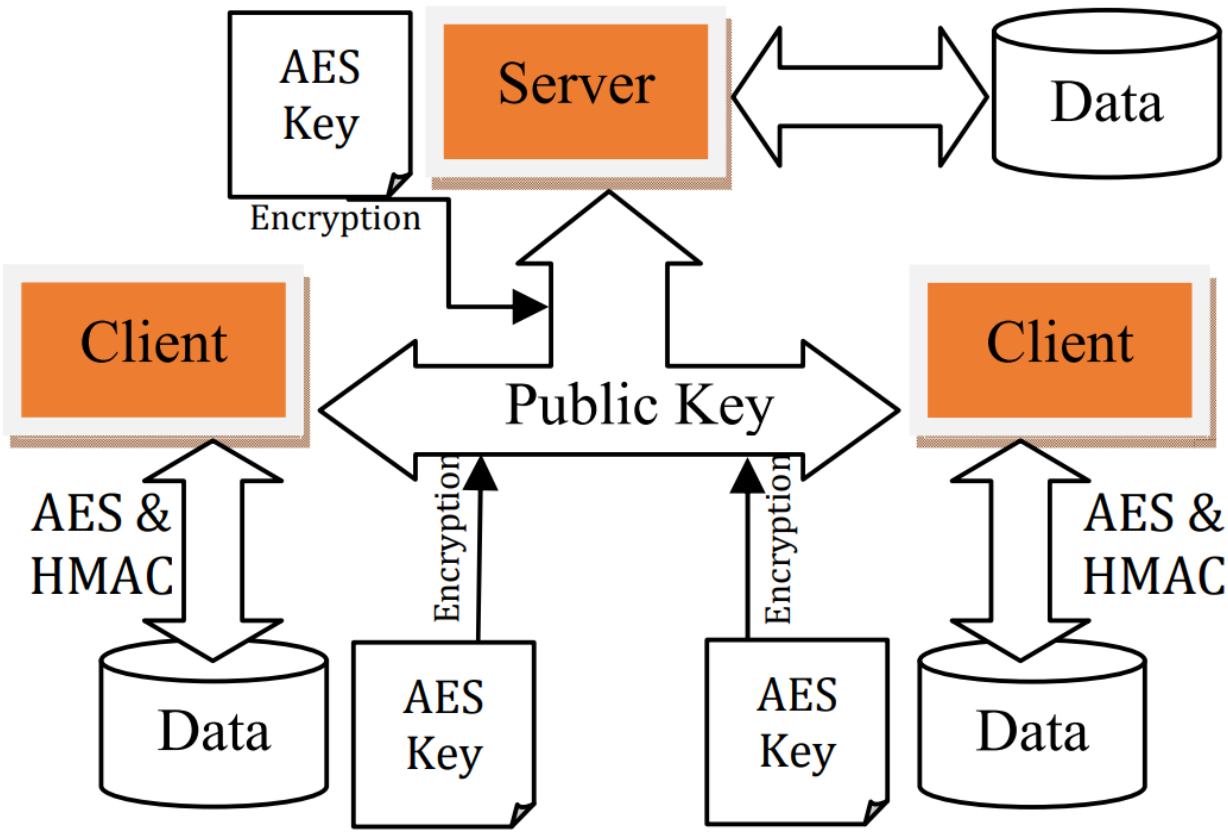
差分能量分析攻击（DPA），基于能量迹之间的相关系数进行分析。

总结：因为rsa在平时ctf中接触的比较多所以收集了各种攻击思路，也说明使用不当就很容易引起安全隐患。脚本虽然实现了简单rsa，但是当数值增加时就难以保证速度。不过主要是理解方法，平时使用可以直接使用内置库。

综合——AES+RSA+HMAC

组成：

- 对称密码AES：加密数据
- 非对称面RSA：加密AES密码
- HMAC：Alice-Bob间消息认证



过程：

- 接收方使用RSA算法生成密钥对(公钥和私钥)，并将公钥值发送给发送方。
- 发送方使用AES算法使用256位长度的密码密钥加密数据。
- 非对称密钥算法(RSA算法)使用接收方的公钥加密对称密钥。
- HMAC-SHA256用于计算包含哈希函数和密钥组合的MAC。
- 加密明文产生的密文，然后在加密的明文后追加MAC。
- 接收方使用存储密钥(私钥)解密AES密钥，然后使用AES密钥解密数据。
- 使用MAC验证信息真实可靠性

算法：

- 输入数据
- 生成RSA公私钥
- 插入加密密钥
- 使用随机salt防止弱密码攻击。salt为64位。首先创建salt1，用于AES中的加密密钥，然后创建slat2，用于HMAC的身份验证密钥。
- 使用128和256位的AES算法
- AES加密并HMAC数据认证

分析：

AES通常是高速的，需要低RAM，因此它对于加密数据非常有用，但是由于它对于加密和解密都是同一个密钥，所以密钥从加密端(发送方)传输到解密端(接收方)存在一个大问题。RSA是用来保护加密密钥的。我们没有使用昂贵的RSA提供程序，而是生成了一个独立的RSA算法，它生成两个密钥(私有密钥和公共密钥)。私钥保留在接收方，公钥发送到发送方。发送方将使用此密钥加密数据。这将有助于抵御被动攻击，但不利于主动攻击，所以MAC被用来保护加密数据。假设MAC共享的秘密没有被泄露，我们需要能够推断出给定的密文是绝对真实的还是伪造的。因此，密码方案的灵活性确保了(通过MAC代码)消除任何无效密码文本。MAC没有给出任何关于明文的信息，因为密码的输出是随机出现的。基本上，它不携带任何结构从明文到MAC代码。