



南开大学  
Nankai University

# 南 开 大 学

网络空间安全学院

密码学课程报告

---

## 第一次实验报告

——古典密码算法及攻击方法

---

学号： 1611519

姓名： 周子祎

年级： 2016 级

专业： 信息安全-法学

2018 年 11 月 30 日

# 密码学第一次实验报告

## ——古典密码算法及攻击方法

### 一、实验目的

通过 C++ 编程实现移位密码和单表置换密码算法，加深对经典密码体制的了解。并通过对这两种密码实施攻击，了解对古典密码体制的攻击方法。

### 二、实验原理

#### 1. 移位密码

**移位密码**：将英文字母**向前或向后移动一个固定位置**。例如向后移动 3 个位置，即对字母表作置换（不分大小写）。

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
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

设明文为：public keys,则经过以上置换就变成了：sxeolf nhbv。

如果将 26 个英文字母进行编码：A→0, B→1, …, Z→25, 则以上加密过程可简单地写成：

明文： $m = m_1m_2\cdots m_i\cdots$ ，则有

密文： $c=c_1c_2\cdots c_i\cdots$ ，其中  $c_i=(m_i+key \bmod 26)$ ,  $i=1, 2, \cdots$ 。

#### 2. 对移位密码的攻击

移位密码是一种最简单的密码，其有效密钥空间大小为 25。因此，很容易**用穷举的方法攻破**。穷举密钥攻击是指攻击者对可能的密钥的穷举，也就是用所有可能的密钥解密密文，**直到得到有意义的明文**，由此确定出正确的密钥和明文的攻击方法。对移位密码进行穷举密钥攻击，**最多只要试译 25 次**就可以得到正

确的密钥和明文。

### 3. 单表置换密码

单表置换密码就是根据字母表的置换对明文进行变换的方法，例如，给定置换

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

明文：public keys, 则有

密文：mcke bw qxuo。

单表置换实现的一个关键问题是关于置换表的构造。置换表的构造可以有各种不同的途径，主要考虑的是记忆的方便。如**使用一个短语或句子，删去其中的重复部分，作为置换表的前面的部分，然后把没有用到的字母按字母表的顺序依次放入置换表中。**

### 4. 对单表置换密码的攻击方法

在单表置换密码中，由于置换表字母组合方式有  $26!$  种，约为  $4.03 \times 10^{26}$ 。所以采用穷举密钥的方法不是一种最有效的方法。对单表置换密码最有效的攻击方法是**利用自然语言的使用频率**：单字母、双字母组/三字母组、短语、词头/词尾等，这里仅考虑英文的情况。英文的一些显著特征如下：

**短单词**(small words)：在**英文中只有很少几个非常短的单词**。因此，如果在一个加密的文本中可以确定单词的范围，那么就能得出明显的结果。一个字母的单词只有 a 和 I。如果不计单词的缩写，在从电子邮件中选取 500k 字节的样本中，**只有两个字母的单词仅出现 35 次**，而两个字母的所有组合为  $26 \times 26 = 676$

种。而且，还是在那个样本中，只有三个字母的单词出现 196 次，而三个字母的所有组合为  $26 \times 26 \times 26 = 17576$  种。

**常用单词(common words):**再次分析 500k 字节的样本，总共有 5000 多个不同的单词出现。在这里，**9 个最常用的单词出现的总次数占总单词数的 21%**，**20 个最常用的单词出现的总次数占总单词数的 30%**，104 个最常用的单词占 50%，247 个最常用的单词占 60%。样本中最常用的 9 个单词占总词数的百分比为：

the	4.65	to	3.02	of	2.61	I	2.2	a	1.95
and	1.82	is	1.68	that	1.62	in	1.57		

**字母频率(character frequency):**在 1M 字节旧的电子文本中，对字母“A”到“Z”（忽略大小写）分别进行统计。发现近似频率（以百分比表示）：

e	11.67	t	9.53	o	8.22	i	7.81	a	7.73	n	6.71	s	6.55
r	5.97	h	4.52	l	4.3	d	3.24	u	3.21	c	3.06	m	2.8
p	2.34	y	2.22	f	2.14	g	2.00	w	1.69	b	1.58	v	1.03
k	0.79	x	0.30	j	0.23	q	0.12	z	0.09				

从该表中可以看出，最常用的单字母英文是 e 和 t，其他字母使用频率相对来说就小得多。这样，攻击一个单表置换密码，首先统计密文中最常出现的字母，并据此猜出两个最常用的字母，并根据英文统计的其他特征（如字母组合等）进行试译。

### 三、 实验要求

1. 根据实验原理部分对移位密码算法的介绍，自己创建明文信息，并选择一个密钥，编写移位密码算法实现程序，实现加密和解密操作。
2. 两个同学为一组，互相攻击对方用移位密码加密获得的密文，恢复出其明文和密钥。

3. 自己创建明文信息，并选择一个密钥，构建置换表。编写置换密码的加解密实现程序，实现加密和解密操作。
4. 用频率统计方法，试译下面用单表置换加密的一段密文：

SIC GCBSPNA XPMHACQ JB GPYXSMEPNXIY JR SINS MF SPNBRQJSSJBE  
JBFMPQNSJMB FPMQ N XMJBS N SM N XMJBS H HY QCNBR MF N XMRRJHAY  
JBRCGZPC GINBBCA JB RZGI N VNY SINS SIC MPJEBNA QCRRNEC GNB MBAY  
HC PCGMTCPCD HY SIC PJEISFZA PCGJXJCBSR SIC XNPSJGJXNBSR JB SIC  
SPNBRNGSJMB NPC NAJGC SIC MPJEBNSMP MF SIC QCRRNEC HMH SIC  
PCGCJTCP NBD MRGNP N XMRRJHAC MXXMBCBS VIM VJRICR SM ENJB  
ZBNZSIMPJOCD GMBSPMA MF SIC QCRRNEC

写出获得的明文消息和置换表。

## 四、实验环境

1. 实验基础环境：

计算机名: DESKTOP-DH2NC4P  
操作系统: Windows 10 家庭中文版 64 位 (10.0, 版本 17134)  
语言: 中文(简体) (区域设置: 中文(简体))  
系统制造商: LENOVO  
系统型号: 80Y5  
BIOS: 4KCN43WW  
处理器: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz (4 CPUs), ~2.5GHz  
内存: 12288MB RAM  
页面文件: 4143MB 已用, 9892MB 可用  
DirectX 版本: DirectX 12

2. 编译器版本：

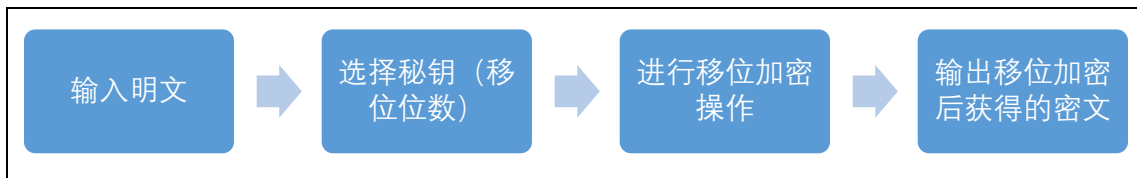
Microsoft Visual Studio 2010 版本 10.0.30319.1 RTMRel

## 五、 实验内容

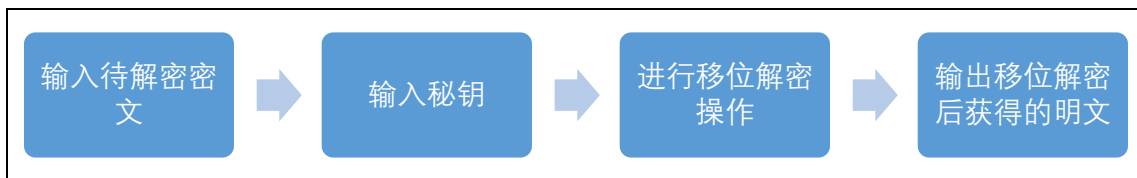
### 1. 移位密码加解密程序实现

#### (1) 加解密流程图

加密流程图：



解密流程图：



#### (2) 程序实现结果

E:\CrptLab\homework1\hw1\_1\Debug\hw1\_1.exe

```
*****
                        加密过程
*****
请输入明文字符串：
HELLOWORLDmimaxuehaonan
*****
请输入密钥（明文右移位数）：
6666662
*****
加密结果：
JGNNQYQTNFokoczwgjcqpcp
-----
*****
                        解密过程
*****
请输入密文字符串：
JGNNQYQTNFokoczwgjcqpcp
*****
请输入密钥（明文右移位数）：
6666662
*****
解密结果：
HELLOWORLDmimaxuehaonan
请按任意键继续. . .
```

### (3) 程序实现代码

(相关代码位于工程文件 hw1\_1 中)

```
1. #include<iostream>
2. #include<string>
3. #include<vector>
4. #include<stdlib.h>
5. using namespace std;
6.
7. //移位加密
8. vector<char> * shift_encrypt(vector<char> * plaintext, int offset)
9. {
10.     int real_offset = offset % 26;
11.
12.     //密文向量指针
13.     vector<char>* ciphertext = new vector<char>;
14.
15.     for (int i = 0; i < (*plaintext).size(); i++)
16.     {
17.         if ((*plaintext)[i] >= 65 && (*plaintext)[i] <= 90)
18.         {
19.             int temp = (*plaintext)[i] + real_offset;
20.             if (temp > 90)
21.                 temp -= 26;
22.             (* ciphertext).push_back((char)temp);
23.             continue;
24.         }
25.
26.         if ((*plaintext)[i] >= 97 && (*plaintext)[i] <= 122)
27.         {
28.             //这里 temp 不能定义为 char,
29.             //因为 char 型变量超过 128 时会自动取补码, 造成异常字符
30.             int temp = (*plaintext)[i] + real_offset;
31.             if (temp > 122)
32.                 temp -= 26;
33.             (* ciphertext).push_back((char)temp);
34.             continue;
35.         }
36.
37.         //非字母内容, 默认不加密
38.         (* ciphertext).push_back((*plaintext)[i]);
39.
40.     }
41.     return ciphertext;
42. }
43.
44. //移位解密
45. vector<char> * shift_decrypt(vector<char> * ciphertext, int offset)
46. {
47.     int real_offset = offset % 26;
48.
49.     //明文向量指针
50.     vector<char>* plaintext = new vector<char>;
51.
52.     for (int i = 0; i < (*ciphertext).size(); i++)
```

```
53.     {
54.         if ((*ciphertext)[i] >= 65 && (*ciphertext)[i] <= 90)
55.         {
56.             int temp = (*ciphertext)[i] - real_offset;
57.             if (temp < 65)
58.                 temp += 26;
59.             (*plaintext).push_back((char)temp);
60.             continue;
61.         }
62.
63.         if ((*ciphertext)[i] >= 97 && (*ciphertext)[i] <= 122)
64.         {
65.             int temp = (*ciphertext)[i] - real_offset;
66.             if (temp < 97)
67.                 temp += 26;
68.             (*plaintext).push_back((char)temp);
69.             continue;
70.         }
71.         //非字母内容，默认不解密
72.         (*plaintext).push_back((*ciphertext)[i]);
73.
74.     }
75.     return plaintext;
76. }
77.
78. int main()
79. {
80.     string mstring_input; //加密时输入的明文字符串
81.     string cstring_input; //解密时输入的密文字符串
82.     int offset;
83.
84.     cout << "*****" << endl;
85.     cout << "          加密过程          " << endl;
86.     cout << "*****" << endl;
87.     cout << "请输入明文字符串: \n";
88.     getline(cin, mstring_input);
89.     cout << "*****" << endl;
90.     cout << "请输入密钥（明文右移位数）: \n";
91.     cin >> offset;
92.     cout << "*****" << endl;
93.
94.     //使用 STL 中的 vector，便于后续操作
95.
96.     //加密过程明文向量指针 enc_plaintext
97.     vector<char> * enc_plaintext = new vector<char>
(mstring_input.begin(), mstring_input.end());
98.
99.     //加密过程密文向量指针 enc_ciphertext
100.    vector<char> * enc_ciphertext;
101.
102.    //移位加密
103.    enc_ciphertext = shift_encrypt(enc_plaintext, offset);
104.
105.    //输出加密后的密文
106.    cout << "加密结果: \n";
```



```
107.     for(int i = 0; i < (*enc_ciphertext).size(); i++)
108.     {
109.         cout<<(*enc_ciphertext)[i];
110.     }
111.     cout<<endl;
112.     cout << "-----"<<endl;
113.
114.     cout << "*****"<<endl;
115.     cout << "          解密过程          "<<endl;
116.     cout << "*****"<<endl;
117.     cout << "请输入密文字符串: \n";
118.     //防止之前的 cin 输入造成影响
119.     cin.ignore();
120.     getline(cin, cstring_input);
121.     cout << "*****"<<endl;
122.     cout << "请输入密钥（明文右移位数）: \n";
123.     cin >> offset;
124.     cout << "*****"<<endl;
125.     //使用 STL 中的 vector，便于后续操作
126.
127.     //解密过程密文向量指针 dec_ciphertext
128.     vector<char> * dec_ciphertext = new vector<char>
(cstring_input.begin(), cstring_input.end());
129.
130.     //解密过程明文向量指针 dec_plaintext
131.     vector<char> * dec_plaintext;
132.
133.     //移位解密
134.     dec_plaintext=shift_decrypt(dec_ciphertext, offset);
135.
136.     //输出解密后的明文
137.     cout << "解密结果: \n";
138.     for(int i = 0; i < (*dec_plaintext).size(); i++)
139.     {
140.         cout<<(*dec_plaintext)[i];
141.     }
142.     cout<<endl<<endl;
143.
144.     system("pause");
145.     return 0;
146. }
```

## 2. 移位密码攻击

### (1) 暴力破解实现结果

```

E:\CrptLab\homework1\hw1_2\Debug\hw1_2.exe
*****
          破解过程
*****
请输入要破解的密文字符串:
Dqdaqy Kdyluhijyo yi edu ev jxu secfhuxdiylu kdyluhijyui myjx jxu mytuij hqdwu ev tyisyfbydui.
*****
暴力破解过程:
如果移位位数为0, 则明文为: Dqdaqy Kdyluhijyo yi edu ev jxu secfhuxdiylu kdyluhijyui myjx jxu mytuij hqdwu ev tyisyfbydui.
如果移位位数为1, 则明文为: Cpczpx Jcxktghxin xh dct du iwt rdbegtwtchxkt jcxktghxixth lxw iwt lxsthi gpcvt du sxhrxeaxcth.
如果移位位数为2, 则明文为: Bobyow Ibwjsfgwhm wg cbs ct hvs qcadfsvsbgwjs ibwjsfgwhwsg kwhv hvs kwrsgh fobus ct rwgqwdzwbgs.
如果移位位数为3, 则明文为: Anaxnv Havirefvgf vf bar bs gur pbzcerurafvir havirefvgvrf jvgu gur jvqrfg enatr bs qvfpvcyvarf.
如果移位位数为4, 则明文为: Zmzwmu Gzuhqdeufk ue azq ar ftq oaybdqtqzeuhq gzuhqdeufuqe iuft ftq iupqef dmzsq ar pueoubxuzqe.
如果移位位数为5, 则明文为: Ylyvlt Fytpcdtejt td zyp zq esp nzxacpspydtgp fytgpcdtetpd htes esp htopde clyrp zq otdntawtypd.
如果移位位数为6, 则明文为: Xkxuks Exsfobcsdi sc yxo yp dro mywzboroxcsfo exsfobcsdsoc gsdr dro gsnocd bkxqo yp nscmszvsxoc.
如果移位位数为7, 则明文为: Wjwtrj Dwrenabrch rb xwn xo cqn lxvyanqnbwren dwrenabrcrbn frqz cqn frmnbc ajwpn xo mrblrlyurwnb.
如果移位位数为8, 则明文为: Vvisiq Cvqdmzaqbg qa wvm wn bpm kwuxzmpmvaqdm cvqdmzaqbqma eqbp bpm eqlmab zivom wn lqakqztqvma.
如果移位位数为9, 则明文为: Uuhurhp Bupclyzpafl pz vul vm aol jvtwyloolzpcpl bupclyzpaflz dpaol aol dplza yhuul vm kpzjwspulz.
如果移位位数为10, 则明文为: Tgtqgo Atobkxyoze oy utk ul znk iusvxnkntyobk atobkxyozoky cozn znk cojkzy xgtmk ul joyiovrotky.
如果移位位数为11, 则明文为: Sfspfn Zsnaixnynd nx tsj tk ymj htruwjmxsxnaj zsnaixnyynjx bnyx ymj bnijxy wfsjl tk inxhnuqnsjx.
如果移位位数为12, 则明文为: Reroem Yrmziwmxnc mw sri sj xli gsqtviliirwnzj yrmziwmxmxiw amxl xli amhiwx verki sj hmwgmtpmriw.
如果移位位数为13, 则明文为: Qdqndf Xqlyhuvlwb lv rgh ri wkh frpsuhkhqvlyh xqlyhuvlwlhv zlw wkh zlghev udqjh ri glvlfsolghv.
如果移位位数为14, 则明文为: Pcpmck Wpkxgtukva ku qpg qh vjg eqortgjpukxg wpkxgtukvkgu ykvj vjg ykfguv tcpig qh fkuokrnrkpgu.
如果移位位数为15, 则明文为: Obolbj Vojwfstjuz jt pof pg uif dnpgsfifotjwf vojwfstjuift xjuj uif xjeftu sbohf pg ejtdjqmjoft.
如果移位位数为16, 则明文为: Nankai University is one of the comprehensive universities with the widest range of disciplines.
如果移位位数为17, 则明文为: Mzmjzh Tmhudqrhsx hr nmd ne sgd bnloqgdgmrhud tmhudqrhshdr vhsq sgd vhcdrs qzmfd ne chrbhokhmdr.
如果移位位数为18, 则明文为: Llylyg Slgtcpqgrw gq mlc md rfc amknpcfcfgtc slgtcpqgrgcq ugrf rfc ugbcqr pylec md bgqagnjlcq.
如果移位位数为19, 则明文为: Kkxhxf Rkfsbopfqv fp lkb lc qeb zljmobeekpfsb rkfsbopfqfqb tfqe qeb tfabpq oxkdb lc afpzmifkbp.
如果移位位数为20, 则明文为: Jwlgwe Qjeranoepu eo kja kb pda ykilnadajoera qjeranoepao sepd pda sezaop nwjca kb zeoyelhejao.
如果移位位数为21, 则明文为: Iviyvd Pidqzmdot dn jiz ja ocz xjhkmzcizndqz pidqzmdodzn rdoo ocz rdyzno mwibz ja ydnxdkgidzn.
如果移位位数为22, 则明文为: Huheuc Ohcpylmcns cm ihy iz nby wigjlybyhmcpv ohcpylmcncvm qcnb nby qcxywn luhay iz xcnwcfjchym.
如果移位位数为23, 则明文为: Gtgdtb Ngboxklbmr bl hgx hy max vfhikxaglbbox ngboxklbmbxl pbma max pbwxml ktgzz hy wblvbiebgxl.
如果移位位数为24, 则明文为: Fsfcsa Mfanwjkalq ak gfw gx lzw ugehjwzfwkanw mfanwjkalaw oalz lzw oawwkl jsfyw gx vakuahdafwk.
如果移位位数为25, 则明文为: Erebrz Lezmvijskp zj fev fw kyv tfdgivyvejzmv lezmvijskzvz nzky kyv nzuvjk irexv fw uzjtzgzevj.
请按任意键继续. . .

```

可以通过观察结果, 得到秘钥(偏移量)为 16

## (2) 暴力破解程序代码

(相关代码位于工程文件 hw1\_2 中)

其中加解密函数 shift\_decrypt、shift\_encrypt 代码不变, 不再重复展示。

```

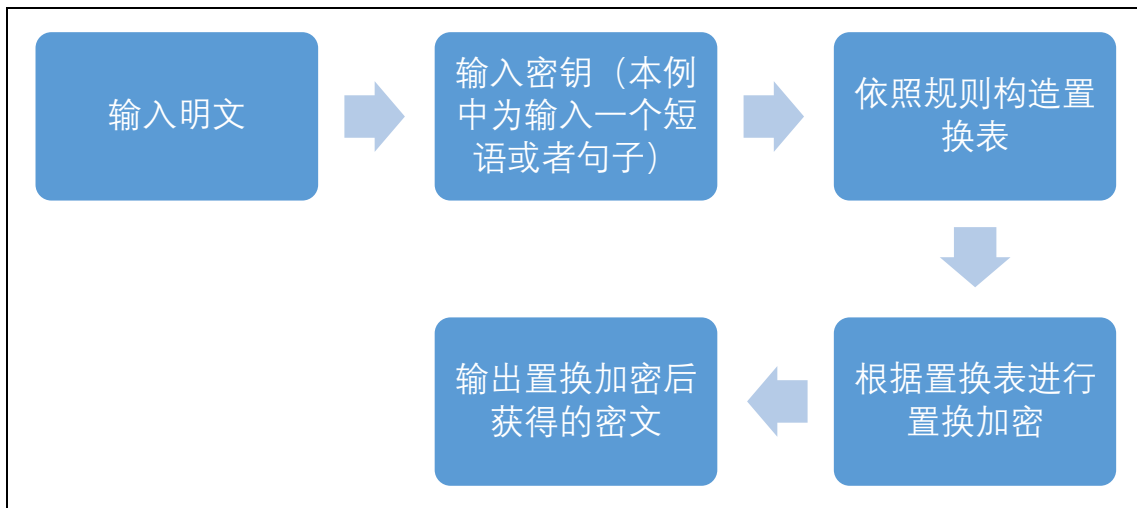
1. int main()
2. {
3.     string cstring_crack; //破解时输入的密文字符串
4.     int offset=0;
5.
6.     cout << "*****" << endl;
7.     cout << "          破解过程          " << endl;
8.     cout << "*****" << endl;
9.     cout << "请输入要破解的密文字符串: \n";
10.    getline(cin, cstring_crack);
11.    cout << "*****" << endl;
12.    cout << "暴力破解过程: \n";
13.
14.    //破解过程密文向量指针 crk_ciphertext
15.    vector<char> * crk_ciphertext = new vector<char>
(cstring_crack.begin(), cstring_crack.end());
16.
17.    //破解过程明文向量指针 crk_plaintext
18.    vector<char> * crk_plaintext = new vector<char>;
19.
20.    for(; offset<=25; offset++)
21.    {
22.        cout << "如果移位位数为"<<offset<<" , 则明文为: ";
23.
24.        crk_plaintext=shift_decrypt(crk_ciphertext, offset);

```

```
25.  
26.     for(int i = 0; i < (*crk_plaintext).size(); i++)  
27.     {  
28.         cout<<(*crk_plaintext)[i];  
29.     }  
30.     cout<<endl;  
31. }  
32.  
33.     system("pause");  
34.     return 0;  
35. }
```

### 3. 单表置换密码

#### (1) 加密流程图



#### (2) 程序实现结果

```
E:\CrptLab\homework1\hw1_3\Debug\hw1_3.exe
*****
                        单表置换加密
*****
请输入用于构造置换表的字符串:
zhihuanmimabiao
*****
置换表为:
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
z h i u a n m b o c d e f g j k l p q r s t v w x y
*****

-----

*****
                        加密过程
*****
请输入明文字符串:
This is a part of mingwen
*****
加密结果为:
Rboq oq z kzpr jn fogmvag
*****

-----

*****
                        解密过程
*****
请输入密文字符串:
Rboq oq z kzpr jn fogmvag!
*****
解密结果为:
This is a part of mingwen!
*****
请按任意键继续. . .
```

### (3) 程序实现代码

(相关代码位于工程文件 hw1\_3 中)

```
1. #include<iostream>
2. #include<string>
3. #include<vector>
4. #include<stdlib.h>
5. #include <algorithm>
6. using namespace std;
7.
8. //加密过程: 根据置换表 rpltbl_vector 对明文 plaintext 进行加密
9. vector<char>* encrypt(vector<char>* plaintext, vector<char>*
   rpltbl_vector)
10. {
11.     vector<char> * ciphertext= new vector<char>;
12.
13.     for(int i = 0; i < (*plaintext).size(); i++)
14.     {
15.         //小写字母
```

```
16.         if((*plaintext)[i]>= 97 && (*plaintext)[i]<= 122)
17.         {
18.             (* ciphertext).push_back( (*rpltbl_vector)
19. [((*plaintext)[i]-97)] );
20.             continue;
21.         }
22.         //大写字母
23.         if((*plaintext)[i]>= 65 && (*plaintext)[i]<= 90)
24.         {
25.             (*
26. ciphertext).push_back( (*rpltbl_vector)[(*plaintext)[i]-65] - 32 );
27.             continue;
28.         }
29.         //非字母内容，默认不加密
30.         (* ciphertext).push_back((*plaintext)[i]);
31.
32.     }
33.     return ciphertext;
34. }
35.
36. //解密过程：根据置换表 rpltbl_vector 对密文 ciphertext 进行解密
37. vector<char>* decrypt(vector<char>* ciphertext, vector<char>*
    rpltbl_vector)
38. {
39.     vector<char> * plaintext= new vector<char>;
40.
41.     vector<char>::iterator result;
42.
43.     vector<char>::iterator first = (*rpltbl_vector).begin();
44.
45.     for(int i = 0; i < (*ciphertext).size(); i++)
46.     {
47.         //小写字母
48.         if((*ciphertext)[i]>= 97 && (*ciphertext)[i]<= 122)
49.         {
50.             // 在*rpltbl_vector 中 查找值为 (*ciphertext)[i]的那一项的下标
51.             result =
52. find((*rpltbl_vector).begin(),(*rpltbl_vector).end(),(*ciphertext)[i]);
53.
54.             //使用 distance 根据迭代器获得数组下标
55.             (*plaintext).push_back( (char) ( 97+
56. distance(first,result) ) );
57.             continue;
58.         }
59.         //大写字母
60.         if((*ciphertext)[i]>= 65 && (*ciphertext)[i]<= 90)
61.         {
62.             // 在*rpltbl_vector 中 查找值为 (*ciphertext)[i]+32 的那一项的
63. 下标
64.             result =
65. find((*rpltbl_vector).begin(),(*rpltbl_vector).end(),((*ciphertext)[i]+
66. 32) );
67.
68.             //使用 distance 根据迭代器获得数组下标
```

```
66.         (*plaintext).push_back( (char) ( 65+
distance(first,result) ) );
67.
68.         continue;
69.     }
70.
71.     //非字母内容，默认不解密
72.     (* plaintext).push_back((*ciphertext)[i]);
73.
74. }
75. return plaintext;
76. }
77.

78. //根据用户输入字符串构造置换表
79. vector<char>* mk_replace_tbl(vector<char>* mktbl_vector)
80. {
81.     vector<char> * rpltbl_vector= new vector<char>;
82.
83.     //用于存储哪些字母被输入字符串所包含
84.     //0 代表未被包含
85.     int flag[26]={0};
86.
87.     for (int i = 0; i < (*mktbl_vector).size(); i++)
88.     {
89.         //大写字母转为小写
90.         if ((*mktbl_vector)[i] >= 65 && (*mktbl_vector)[i] <= 90)
91.         {
92.             (*mktbl_vector)[i] += 32;
93.         }
94.
95.         //(*mktbl_vector)[i]是字母
96.         if ( (*mktbl_vector)[i] >= 97 && (*mktbl_vector)[i] <= 122 )
97.         {
98.             //查找(*mktbl_vector)[i]是否在*rpltbl_vector 中
99.             vector<char>::iterator result =
find((*rpltbl_vector).begin(),(*rpltbl_vector).end(),(*mktbl_vector)[i]
);
100.
101.             //如果不在*rpltbl_vector 中的话
102.             if(result==(*rpltbl_vector).end())
103.             {
104.                 //插入到 rpltbl_vector 向量中
105.                 (* rpltbl_vector).push_back((*mktbl_vector)[i]);
106.
107.                 //更新 flag 表
108.                 flag[(*mktbl_vector)[i]-97]=1;
109.             }
110.         }
111.     }
112.
113.     for(int i=0;i<26;i++)
114.     {
115.         if(flag[i]==0)
116.             (* rpltbl_vector).push_back((char)(i+97));
117.     }
```

```
118.     return rpltbl_vector;
119. }
120.
121. int main()
122. {
123.     string string_mktbl; //构造置换表时输入的字符串
124.     string mstring_input; //加密过程用户输入的明文字符串
125.     string cstring_input; //解密过程用户输入的密文字符串
126.
127.     cout << "*****" << endl;
128.     cout << "          单表置换加密          " << endl;
129.     cout << "*****" << endl;
130.     cout << "请输入用于构造置换表的字符串: \n";
131.     getline(cin, string_mktbl);
132.     cout << "*****" << endl;
133.     //构造置换表所使用的向量指针 crk_ciphertext
134.     vector<char> * mktbl_vector= new vector<char>
        (string_mktbl.begin(), string_mktbl.end());
135.
136.     //存储置换表的向量指针 rpltbl_vector
137.     vector<char> * rpltbl_vector= new vector<char>;
138.
139.     rpltbl_vector = mk_replace_tbl(mktbl_vector);
140.
141.     cout << "置换表为: \n";
142.
143.     for(int i = 0; i < (*rpltbl_vector).size(); i++)
144.     {
145.         cout<<(char)(97+i)<<" ";
146.     }
147.     cout<<endl;
148.
149.     for(int i = 0; i < (*rpltbl_vector).size(); i++)
150.     {
151.         cout<<(*rpltbl_vector)[i]<<" ";
152.     }
153.     cout<<endl;
154.     cout << "*****" << endl;
155.     cout << endl;
156.     cout << "-----" << endl;
157.     cout<<endl;
158.     cout << "*****" << endl;
159.     cout << "          加密过程          " << endl;
160.     cout << "*****" << endl;
161.     cout << "请输入明文字符串: \n";
162.     getline(cin,mstring_input);
163.     cout << "*****" << endl;
164.     cout << "加密结果为: \n";
165.
166.     //存储加密过程的明文
167.     vector<char> * enc_plaintext= new vector<char>
        (mstring_input.begin(), mstring_input.end()) ;
168.
169.     //存储加密过程的密文
170.     vector<char> * enc_ciphertext= new vector<char>;
171.
172.     //加密
```

```

173.     enc_ciphertext = encrypt(enc_plaintext, rpltbl_vector);
174.
175.     //输出加密结果
176.     for(int i = 0; i < (*enc_ciphertext).size(); i++)
177.     {
178.         cout<<(*enc_ciphertext)[i];
179.     }
180.     cout<<endl;
181.     cout << "*****"<<endl;
182.     cout <<endl;
183.     cout << "-----"<<endl;
184.     cout <<endl;
185.     cout << "*****"<<endl;
186.     cout << "          解密过程          "<<endl;
187.     cout << "*****"<<endl;
188.     cout << "请输入密文字符串: \n";
189.     getline(cin,cstring_input);
190.     cout << "*****"<<endl;
191.     cout << "解密结果为: \n";
192.
193.     //存储解密过程的密文
194.     vector<char> * dec_ciphertext= new vector<char>
(cstring_input.begin(), cstring_input.end()) ;
195.
196.     //存储解密过程的明文
197.     vector<char> * dec_plaintext= new vector<char>;
198.
199.     //解密
200.     dec_plaintext = decrypt(dec_ciphertext, rpltbl_vector);
201.
202.     //输出解密结果
203.     for(int i = 0; i < (*dec_plaintext).size(); i++)
204.     {
205.         cout<<(*dec_plaintext)[i];
206.     }
207.     cout<<endl;
208.     cout << "*****"<<endl;
209.
210.     system("pause");
211.
212.     return 0;
213. }

```

#### 4. 移位密码攻击

攻击的思路是：

首先将密文中每个字母出现的频率进行统计，并与实验文档中给出的自然语言中的字母统计频率进行比较匹配。

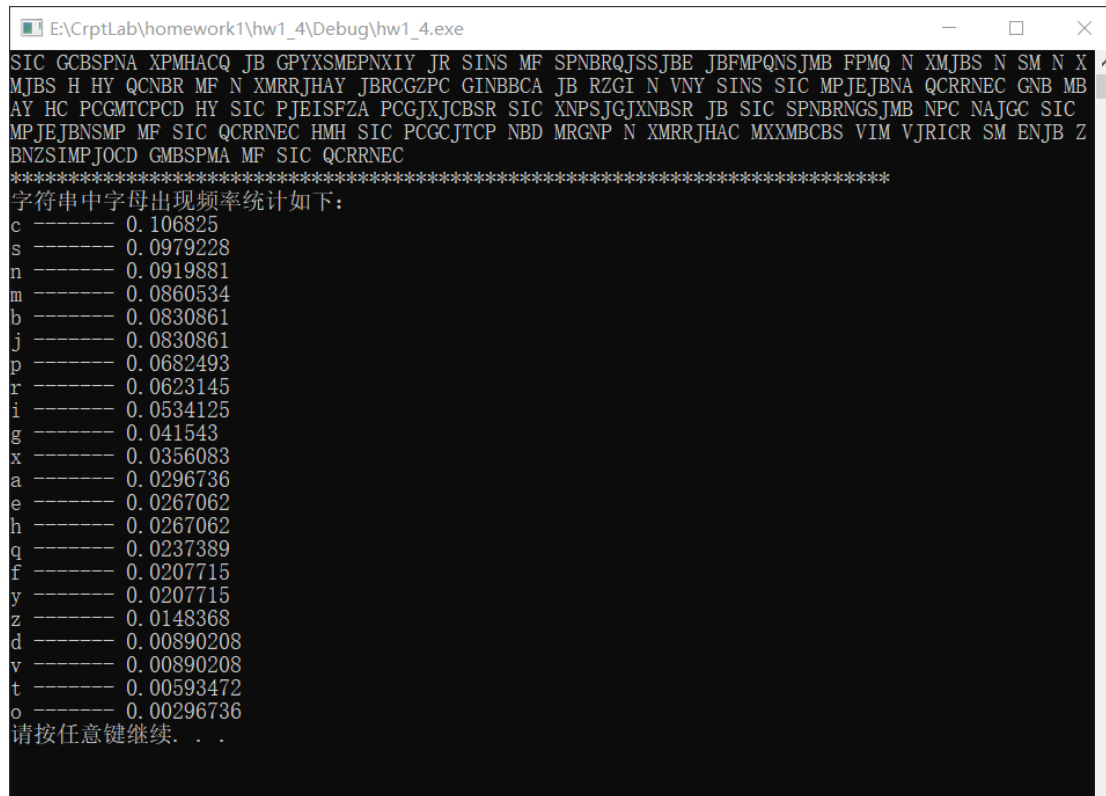
但是，由于所给定的密文段较短，因此仅依靠自然语言的统计规律，难以准确地构造正确的置换表。因此，根据字母频率构造置换表完成置换之后，还需要



手工进行相应的调整。

## 4.1 明文字母频率统计

### (1) 频率统计结果



```
E:\CrptLab\homework1\hw1_4\Debug\hw1_4.exe
SIC GCBSPNA XPMHACQ JB GPYXSMEPNXIY JR SINS MF SPNBRQJSSJBE JBFMPQNSJMB FPMQ N XMJBS N SM N X
MJBS H HY QCNR MF N XMRRJHAY JBRCGZPC GINBBCA JB RZGI N VNY SINS SIC MPJEJBNA QCRRNEC GNB MB
AY HC PCGMTCPD HY SIC PJEISFZA PCGJXJCBSR SIC XNPSJGJXNBSR JB SIC SPNBRNGSJMB NPC NAJGC SIC
MPJEJBNSMP MF SIC QCRRNEC HMH SIC PCGCJTCP NBD MRGNP N XMRRJHAC MXXMBCBS VIM VJRICR SM ENJB Z
BNZSIMPJOC D GMBSPMA MF SIC QCRRNEC
*****
字符串中字母出现频率统计如下：
c ----- 0.106825
s ----- 0.0979228
n ----- 0.0919881
m ----- 0.0860534
b ----- 0.0830861
j ----- 0.0830861
p ----- 0.0682493
r ----- 0.0623145
i ----- 0.0534125
g ----- 0.041543
x ----- 0.0356083
a ----- 0.0296736
e ----- 0.0267062
h ----- 0.0267062
q ----- 0.0237389
f ----- 0.0207715
y ----- 0.0207715
z ----- 0.0148368
d ----- 0.00890208
v ----- 0.00890208
t ----- 0.00593472
o ----- 0.00296736
请按任意键继续. . .
```

### 明文中字母出现频率统计结果：

c ----- 0.106825	r ----- 0.0623145
s ----- 0.0979228	i ----- 0.0534125
n ----- 0.0919881	g ----- 0.041543
m ----- 0.0860534	x ----- 0.0356083
b ----- 0.0830861	a ----- 0.0296736
j ----- 0.0830861	e ----- 0.0267062
p ----- 0.0682493	h ----- 0.0267062

q ----- 0.0237389	d ----- 0.00890208
f ----- 0.0207715	v ----- 0.00890208
y ----- 0.0207715	t ----- 0.00593472
z ----- 0.0148368	o ----- 0.00296736

## (2) 程序实现代码

(相关代码位于工程文件 hw1\_4 中)

```
1. #include<iostream>
2. #include<string>
3. #include<vector>
4. #include<stdlib.h>
5. #include <algorithm>
6. using namespace std;
7.
8. int main()
9. {
10.     string input_str;
11.     vector<int>* freq_tbl= new vector<int>;
12.
13.     vector<int>* letter_tbl= new vector<int>;
14.
15.     //记录(*freq_tbl)按频次由低到高排序后,每个频次对应的字母
16.     char letter_record[26] = {'a'};
17.
18.     for(int i=1;i<=26;i++)
19.     {
20.         (*freq_tbl).push_back(0);
21.         (*letter_tbl).push_back(0);
22.     }
23.
24.     cout<<"请输入密文字符串: \n";
25.     getline(cin, input_str);
26.     cout <<
27.     "*****"
28.     "*****" << endl;
29.
30.     cout<<"字符串中字母出现频率统计如下: \n";
31.     int total_letter_num = 0;
32.
33.     //遍历输入字符串,记录字母频次
34.     for (int i = 0; i<input_str.length(); i++)
35.     {
36.         if ( input_str[i] >= 'A' && input_str[i] <= 'Z' )
37.         {
38.             (*freq_tbl)[ input_str[i] - 'A' ]++;
39.             (*letter_tbl)[ input_str[i] - 'A' ]++;
40.             total_letter_num++;
41.             continue;
42.         }
43.     }
```

```

40.
41.     if ( input_str[i] >= 'a' && input_str[i] <= 'z' )
42.     {
43.         (*freq_tbl)[ input_str[i] - 'a']++;
44.         (*letter_tbl)[ input_str[i] - 'a']++;
45.         total_letter_num++;
46.         continue;
47.     }
48. }
49.
50. //freq_tbl 按字母频次重排(默认升序排列)
51. sort((*freq_tbl).begin(),(*freq_tbl).end());
52.
53. for(int f=25;f>=0;f--)
54. //遍历(*freq_tbl), (*freq_tbl)按频率值由低到高顺序存储字母频次值
55. {
56.     for(int l=0;l<26;l++)
57. //遍历(*letter_tbl), (*letter_tbl)按字母顺序存储字母频次值
58.     {
59.         if( (*letter_tbl)[l] == (*freq_tbl)[f] )
60.         {
61.             letter_record[f] = 'a'+l;
62.
63.             //频率值相同的字母匹配哪个频次都可以, 但匹配上一次之后就不能
再匹配了, 否则 letter_record 会重复记录
64.             (*letter_tbl)[l] = -1;
65.             break;
66.         }
67.     }
68. }
69.
70. for (int k = 25; k >= 0; k--)
71. {
72.     if((*freq_tbl)[k]!=0)
73.         cout << letter_record[k]<< " ----- "
<<double((*freq_tbl)[k])/double(total_letter_num)<< endl;
74. }
75. system("pause");
76.
77. return 0;
78. }

```

## 4.2 置换表构造

### (1) 比照统计结果, 初步构造置换表

将根据此明文得到的字母频率高低排序结果, 与自然语言中字母出现的频率进行匹配, 可得到如下初步的匹配结果:

明文字母	a	b	c	d	e	f	g	h
密文字母	b	v	e	x	c	y	z	i
明文字母	i	j	k	l	m	n	o	p
密文字母	m	l	o	g	h	j	n	q
明文字母	q	r	s	t	u	v	w	x
密文字母	u	r	p	s	a	t	d	k
明文字母	y	z						
密文字母	f	w						

(2) 用于构造置换表的字符串 Str1 为：

b v e x c y z i m l o g h j n q u r p s  
a t d k f w

需要破译的密文为：

SIC GCBSPNA XPMHACQ JB GPYXSMEPNXIY JR SINS MF SPNBRQJSSJBE  
JBFMPQNSJMB FPMQ N XMJBS N SM N XMJBS H HY QCNBR MF N XMRRJHAY  
JBRCGZPC GINBBCA JB RZGI N VNY SINS SIC MPJEBNA QCRRNEC GNB MBAY  
HC PCGMTCPCD HY SIC PJEISFZA PCGJXJCBSR SIC XNPSJGJXNBSR JB SIC  
SPNBRNGSJMB NPC NAJGC SIC MPJEBNSMP MF SIC QCRRNEC HMH SIC  
PCGCJTCP NBD MRGNP N XMRRJHAC MXXMBCBS VIM VJRICR SM ENJB  
ZBNZSIMPJOCD GMBSPMA MF SIC QCRRNEC

转换为小写字母表示为：

sic gcbspna xpmhacq jb gpyxsmepnxiy jr sins mf spnbrqjssjbe jbfmpqnsjmb fpmq  
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbrcgzpc ginbbca jb rzgi n vny  
sins sic mpjejbna qcrrnec gnb mbay hc pcgmtcpd hy sic pjeisfza pcgjxjbsr sic  
xnpsjgxnbsr jb sic spnbrngsjmb npc najgc sic mpjejbnsmp mf sic qcrrnec hmh sic  
pcgcjtcp nbd mrgnp n xmrrjhac mxxmbcbs vim vjricr sm enjb zbnzsimpjocd  
gmbspma mf sic qcrrnec

### 4.3 完成置换，手动调整

(1) 根据初步构造的置换表，利用之前实现的单表置换程序 hw1\_3, 完成初步置换，结果如下：

(对程序文件略有修改，修改后的代码见 hw1\_5)



```
E:\CrptLab\homework1\hw1_5\Debug\hw1_5.exe
*****
                        单表置换加密
*****
请输入用于构造置换表的字符串:
bvexcyzimloghjnrpsatdkfw
*****
置换表为:
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 v e x c y z i m l o g h j n q u r p s a t d k f w
*****

*****
                        解密过程
*****
请输入密文字符串:
sic gcbspna xpmhacq jb gpyxsmepnxiy jr sins mf spnbrqjssjbe jbfmpqnsjmb fpmq
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbrcgzpc ginbbca jb rzgi n vny
sins sic mpjejbna qcrrnec gnb mbay hc pcgmtcpd hy sic pjeisfza pcgjxjbsr si
c xnpsjgxnbsr jb sic spnbrngsjmb npc najgc sic mpjejbnsmp mf sic qcrrnec hmh
sic pcgcjtcp nbd mrgnp n xmrrjhac mxxmbcbs vim vjricr sm enjb zbnzsimpjocd g
mbspma mf sic qcrrnec
*****
解密结果为:
the leatsou dsimuep na lsfdticsodhf nr thot iy tsoarpnttnac nayispotnia ysip
o dinat o ti o dinat m mf peoar iy o dirrnmufo narelge lhoaau na rglh o bof
thot the isncnaou perroce loa iauf me selivesew mf the snchtygu selndneatr th
e dostnlndoatr na the tsoaroltnia ose ounle the isncnaotis iy the perroce mim
the selenves oaw irlos o dirrnmufo iddiaeat bhi bnrher ti cona gaogthisnkew l
iatsiu iy the perroce
*****
请按任意键继续. . .
```

## (2) 手工调整置换表

- 1) 首先观察目前得到的明文，第一行中有“thot”一词，并且可以看出，“the”已经匹配正确，因此 t、h 字母是无误的，因此根据英文中常用单词“that”。因此，对置换表中明文字母 o、a 对应的密文字母（相当于在解密过程中将原本被翻译为 o 的密文翻译为 a，将原本被翻译为 a 的密文翻译为 o）：

调整后置换表-2 如下：

明文字母	a	b	c	d	e	f	g	h
密文字母	n	v	e	x	c	y	z	i
明文字母	i	j	k	l	m	n	o	p
密文字母	m	l	o	g	h	j	b	q
明文字母	q	r	s	t	u	v	w	x
密文字母	u	r	p	s	a	t	d	k
明文字母	y	z						
密文字母	f	w						

表构造字符串-2 如下：

n v e x c y z i m l o g h j b q u r p s  
a t d k f w

翻译结果-2 如下：

```
E:\CrptLab\homework1\hw1_5\Debug\hw1_5.exe
*****
                        单表置换加密
*****

请输入用于构造置换表的字符串：
nvexcyzimloghjbqurpsatdkfw
*****

置换表为：
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
n v e x c y z i m l o g h j b q u r p s a t d k f w
*****

-----

*****
                        解密过程
*****

请输入密文字符串：
sic gcbspna xpmhacq jb gpyxsmepnxiy jr sins mf spnbrqjssjbe jbfmpqnsjmb fpmq
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbrcgzpc ginbbca jb rzgi n vn
y sins sic mpjejbna qcrnec gnb mbay hc pcgmtcpd hy sic pjeisfza pcgjxjbsr
sic xnpjsgjxnbsr jb sic spnbrngsjmb npc najgc sic mpjejbnsmp mf sic qcrnec
hnh sic pcgcjtcp nbd mrgnp n xmrrjhac mxmmbcbs vim vjricr sm enjb zbnzsimpj
ocd gmbspma mf sic qcrnec
*****

解密结果为：
the leotsau dsimuep no lsfdticsadhf nr that iy tsaorpnttnoc noyispatnio ysip
a dinot a ti a dinot m mf peaor iy a dirrmuf norelgse lhaoeu no rglh a ba
f that the isncnoau perrace lao iouf me selivesew mf the snctygu selndneotr
the dastnlndaotr no the tsaoraltnio ase aunle the isncnoatis iy the perrace
mim the selenves aow irlas a dirrmue iddioeot bhi bnrher ti cano goagthisn
kew liotsiu iy the perrace
*****

请按任意键继续. . .
```

- 2) 此时我们观察“明文”中出现了 nr that iy 的结构，第二行出现 了 iy a xxxx 的结构。根据英文的语法结构我们可以推测应将 iy 改为 of。

调整后置换表-3 如下：

明文字母	a	b	c	d	e	f	g	h
密文字母	n	v	e	x	c	f	z	i
明文字母	i	j	k	l	m	n	o	p
密文字母	b	l	o	g	h	j	m	q
明文字母	q	r	s	t	u	v	w	x
密文字母	u	r	p	s	a	t	d	k
明文字母	y	z						
密文字母	y	w						

表构造字符串-3 如下：

n v e x c f z i b l o g h j m q u r p s  
a t d k y w

翻译结果-3 如下：



```

E:\CrptLab\homework1\hw1_5\Debug\hw1_5.exe
*****
                                单表置换加密
*****

请输入用于构造置换表的字符串:
nvexcfzibloghjmqupsatdkyw
*****

置换表为:
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
n v e x c f z i b l o g h j m q u r p s a t d k y w
*****

-----

*****
                                解密过程
*****

请输入密文字符串:
sic gcbspna xpmhacq jb gpyxsmepnxiy jr sins mf spnbrqjssjbe jbfmpqnsjmb fpmq
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbrcgzpc ginbbca jb rzgi n vn
y sins sic mpjejbn qcrnec gnb mbay hc pcgmtcpd hy sic pjeisfza pcgjsxjbsr
sic xnpsjgxnbsr jb sic spnbrngsjmb npc najgc sic mpjejbnsmp mf sic qcrnec
hnh sic pcgcjtcp nbd mrgnp n xmrrjhac mxmmbcbs vim vjricr sm enjb zbnzsimpj
ocd gmbspma mf sic qcrnec
*****

解密结果为:
the leitsau dsomuep ni lsydtocsadhy nr that of tsairpnttnic nifospatnoi fsop
a donit a to a donit m my peair of a dorrnmuy nirelgse lhaiieu ni rglh a ba
y that the osncniau perrace lai oiuy me selovesew my the snchtfgu selndneitr
the dastnlndaitr ni the tsairaltnoi ase aunle the osncniatos of the perrace
mom the selenves aiw orlas a dorrnmue odioeit bho bnrher to cani giagthosn
kew loitsou of the perrace
*****

请按任意键继续. . .

```

- 3) 利用根据上一步的推测结果，结合此时的“明文”中多次出现 n 开头的双字母短单词，我们将 **nr** 改为 **is**，这也会在新的“明文”中形成 **is that of** 的语法结构。

调整后置换表-4 如下：

明文字母	a	b	c	d	e	f	g	h
密文字母	n	v	e	x	c	f	z	i
明文字母	i	j	k	l	m	n	o	p
密文字母	j	l	o	g	h	b	m	q
明文字母	q	r	s	t	u	v	w	x
密文字母	u	p	r	s	a	t	d	k
明文字母	y	z						
密文字母	y	w						

表构造字符串-4 如下：

n v e x c f z i j l o g h b m q u p r s  
a t d k y w

翻译结果-4 如下：

```

E:\CrptLab\homework1\hw1_5\Debug\hw1_5.exe
*****
                        单表置换加密
*****

请输入用于构造置换表的字符串：
nvexcfzijklghbmquprsatdkyw
*****

置换表为：
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
n v e x c f z i j l o g h b m q u p r s a t d k y w
*****

-----

*****
                        解密过程
*****

请输入密文字符串：
sic gcbspna xpmhacq jb gpyxsmepnxiy jr sins mf spnbrqjssjbe jbfmpqnsjmb fpmq
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbregezpc ginbbca jb rzgi n vn
y sins sic mpjejbna qcrrnec gnb mbay hc pcgmtcpd hy sic pjeisfza pcgjsxjbsr
sic xnpsjgxnbsr jb sic spnbrngsjmb npc najgc sic mpjejbnsmp mf sic qcrrnec
hnh sic pcgcjtcp nbd mrgnp n xmrrjhac mxmmbcbs vim vjricr sm enjb zbnzsimpj
ocd gmbspma mf sic qcrrnec
*****

解密结果为：
the lentrau dromuep in lrydtocradhy is that of transpittinc inforpation frop
a doint a to a doint m my peans of a dossimuy inselgre lhanneu in sglh a ba
y that the oricinau pessace lan onuy me reloverew my the richtfgu relidients
the dartilidants in the transaltion are auile the oricinator of the pessace
mom the releiver anw oslar a dossimue oddonent bho bishes to cain gnagthori
kew lontrou of the pessace
*****

请按任意键继续. . .

```

4) 使用类似的方法进行推断，并且通过修改置换表后编译程序的方式验证推断。

先后使用的推断有：

- inforpation  $\rightarrow$  information :  $p \rightarrow m$

将明文  $m$  对应的密文设置为明文  $p$  当前对应的密文

- doint  $\rightarrow$  point :  $d \rightarrow p$

将明文  $p$  对应的密文设置为明文  $d$  当前对应的密文

- $dy \rightarrow by : d \rightarrow b$

将明文 **b** 对应的密文设置为明文 **d** 当前对应的密文

- $probuem \rightarrow problem : u \rightarrow l$

将明文 **l** 对应的密文设置为明文 **u** 当前对应的密文

- $uentral \rightarrow central : u \rightarrow c$

将明文 **c** 对应的密文设置为明文 **u** 当前对应的密文

- $cryptouraphy \rightarrow cryptography : u \rightarrow g$

将明文 **g** 对应的密文设置为明文 **u** 当前对应的密文

- $day \rightarrow way : d \rightarrow w$

将明文 **w** 对应的密文设置为明文 **d** 当前对应的密文

- $unauthoriked \rightarrow unauthorized : k \rightarrow z$

将明文 **z** 对应的密文设置为明文 **k** 当前对应的密文

最终调整后的置换表如下：

明文字母	a	b	c	d	e	f	g	h
密文字母	n	h	g	d	c	f	e	i
明文字母	i	j	k	l	m	n	o	p
密文字母	j	l	w	a	q	b	m	x
明文字母	q	r	s	t	u	v	w	x
密文字母	u	p	r	s	z	t	v	k
明文字母	y	z						
密文字母	y	o						

表构造字符串如下：

n h g d c f e i j l w a q b m x u p r s  
z t v k y o

最终翻译结果如下：

```
E:\CrptLab\homework1\hw1_5\Debug\hw1_5.exe
*****
                        单表置换加密
*****

请输入用于构造置换表的字符串：
nhgdcfeijlwqbmuxprsztvkyo
*****

置换表为：
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
n h g d c f e i j l w a q b m x u p r s z t v k y o
*****

-----

*****
                        解密过程
*****

请输入密文字符串：
sic gcb spna xpm hacq jb gpy xsmepnxiy jr sins mf spn brqjssjbe jbf mpqnsjmb fpmq
n xmjbs n sm n xmjbs h hy qcnbr mf n xmrrjhay jbr cgzpc ginbbca jb rzgi n vn
y sins sic mpjebna qcrrnec gnb mbay hc pcg mtcped hy sic pjeisfza pcgjxjcbsr
sic xnp sjgjxnbsr jb sic spnbrngsjmb npc najgc sic mpjebnsmp mf sic qcrrnec
hnh sic pcgcjtcp nbd mrgnp n xmrrjhac mx xmbcbs vim vjr icr sm enjb zbnzsimpj
ocd gmb spma mf sic qcrrnec
*****

解密结果为：
the central problem in cryptography is that of transmitting information from
a point a to a point b by means of a possibly insecure channel in such a wa
y that the original message can only be recovered by the rightful recipients
the participants in the transaction are alice the originator of the message
bob the receiver and oscar a possible opponent who wishes to gain unauthori
zed control of the message
*****

请按任意键继续. . .
```