

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

1. 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。

1. 实验原理
2. **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＝m1m2**…**mi**…,** 则有

密文：c=c1c2**…**ci**…,** 其中 ci=(mi+key mod26)，i＝1，2，**…**。

1. **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。

1. **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：mckebw 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**。

1. **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4.03×1026。

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26×26＝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26×26×26＝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％**，2**0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。

1. 实验要求
2. 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。
3. 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。
4. 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。
5. 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 MPJEJBNA QCRRNEC GNB MBAY HC PCGMTCPCD 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 ZBNZSIMPJOCD GMBSPMA MF SIC QCRRNEC

写出获得的明文消息和置换表。

1. 实验环境
2. 实验基础环境：



1. 编译器版本：

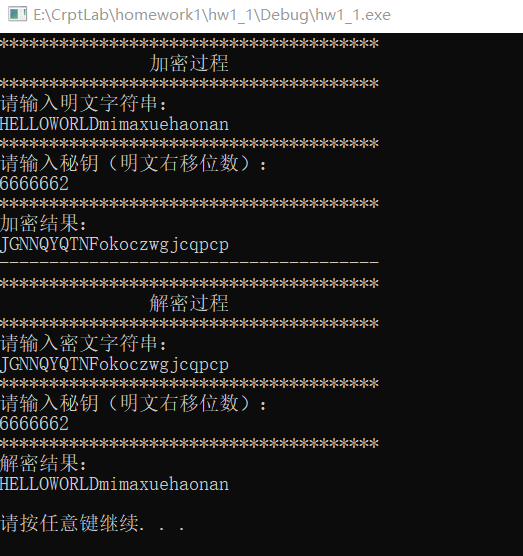
Microsoft Visual Studio 2010 版本 10.0.30319.1 RTMRel

1. 实验内容
2. 移位密码加解密程序实现
3. 加解密流程图

加密流程图：

解密流程图：

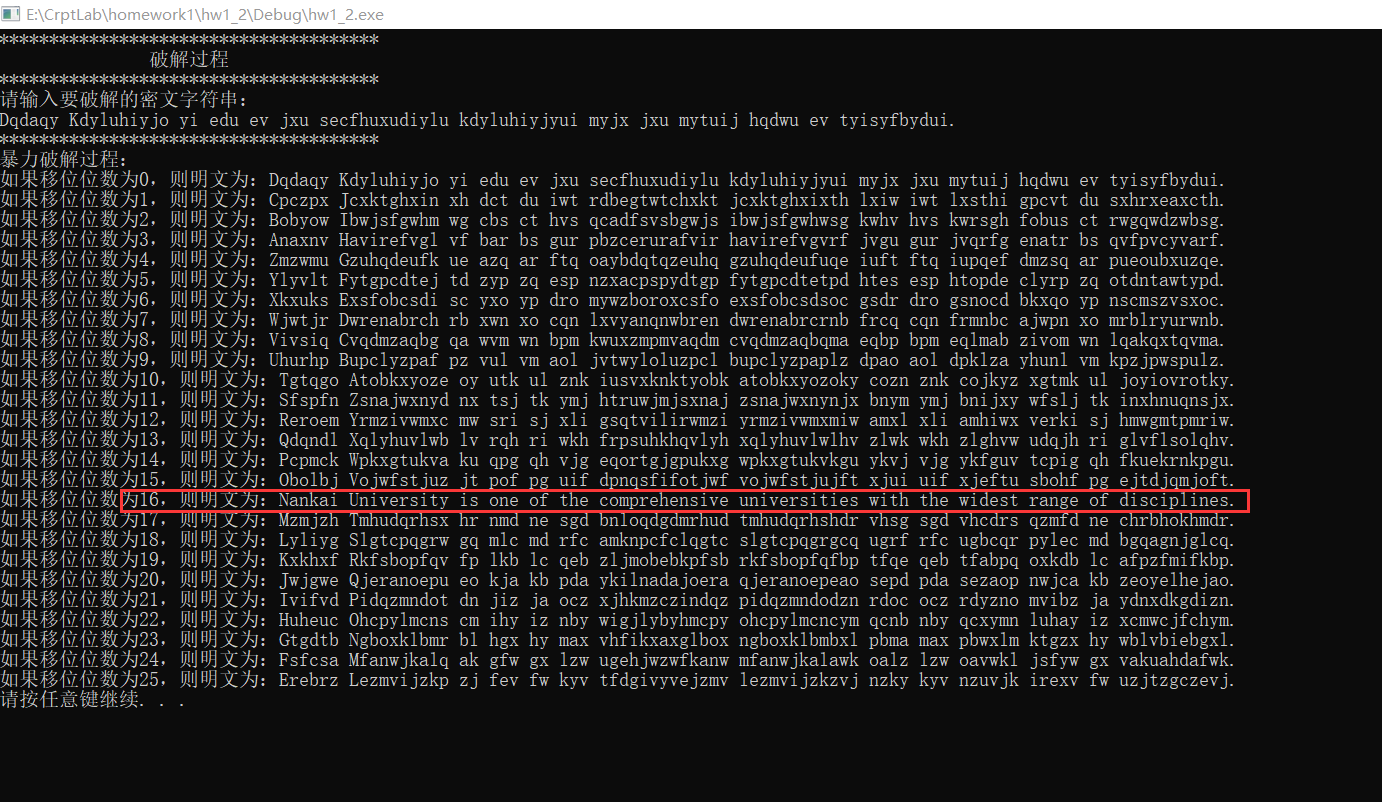
1. 程序实现结果



1. 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. //移位加密
7. vector<char> \* shift\_encrypt(vector<char> \* plaintext, int offset)
8. {
9. int real\_offset = offset % 26;
10. //密文向量指针
11. vector<char>\* ciphertext = new vector<char>;
12. for (int i = 0; i < (\*plaintext).size(); i++)
13. {
14. if ((\*plaintext)[i] >= 65 && (\*plaintext)[i] <= 90)
15. {
16. int temp = (\*plaintext)[i] + real\_offset;
17. if (temp > 90)
18. temp -= 26;
19. (\* ciphertext).push\_back((char)temp);
20. continue;
21. }
22. if ((\*plaintext)[i] >= 97 && (\*plaintext)[i] <= 122)
23. {
24. //这里temp不能定义为char，
25. //因为char型变量超过128时会自动取补码，造成异常字符
26. int temp = (\*plaintext)[i] + real\_offset;
27. if (temp >122)
28. temp -= 26;
29. (\* ciphertext).push\_back((char)temp);
30. continue;
31. }
32. //非字母内容，默认不加密
33. (\* ciphertext).push\_back((\*plaintext)[i]);
34. }
35. return ciphertext;
36. }
37. //移位解密
38. vector<char> \* shift\_decrypt(vector<char> \* ciphertext, int offset)
39. {
40. int real\_offset = offset % 26;
41. //明文向量指针
42. vector<char>\* plaintext = new vector<char>;
43. for (int i = 0; i < (\*ciphertext).size(); i++)
44. {
45. if ((\*ciphertext)[i] >= 65 && (\*ciphertext)[i] <= 90)
46. {
47. int temp = (\*ciphertext)[i] - real\_offset;
48. if (temp < 65)
49. temp += 26;
50. (\* plaintext).push\_back((char)temp);
51. continue;
52. }
53. if ((\*ciphertext)[i] >= 97 && (\*ciphertext)[i] <= 122)
54. {
55. int temp = (\*ciphertext)[i] - real\_offset;
56. if (temp < 97)
57. temp += 26;
58. (\* plaintext).push\_back((char)temp);
59. continue;
60. }
61. //非字母内容，默认不解密
62. (\* plaintext).push\_back((\*ciphertext)[i]);
63. }
64. return plaintext;
65. }
67. int main()
68. {
69. string mstring\_input;//加密时输入的明文字符串
70. string cstring\_input;//解密时输入的密文字符串
71. int offset;
72. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
73. cout << " 加密过程 "<<endl;
74. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
75. cout << "请输入明文字符串：\n";
76. getline(cin, mstring\_input);
77. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
78. cout << "请输入秘钥（明文右移位数）：\n";
79. cin >> offset;
80. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
81. //使用STL中的vector，便于后续操作
83. //加密过程明文向量指针enc\_plaintext
84. vector<char> \* enc\_plaintext = new vector<char> (mstring\_input.begin(), mstring\_input.end());
85. //加密过程密文向量指针enc\_ciphertext
86. vector<char> \* enc\_ciphertext;
87. //移位加密
88. enc\_ciphertext=shift\_encrypt(enc\_plaintext, offset);
90. //输出加密后的密文
91. cout << "加密结果：\n";
92. for(int i = 0; i < (\*enc\_ciphertext).size(); i++)
93. {
94. cout<<(\*enc\_ciphertext)[i];
95. }
96. cout<<endl;
97. cout << "--------------------------------------"<<endl;
98. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
99. cout << " 解密过程 "<<endl;
100. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
101. cout << "请输入密文字符串：\n";
102. //防止之前的cin输入造成影响
103. cin.ignore();
104. getline(cin, cstring\_input);
105. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
106. cout << "请输入秘钥（明文右移位数）：\n";
107. cin >> offset;
108. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
109. //使用STL中的vector，便于后续操作
111. //解密过程密文向量指针dec\_ciphertext
112. vector<char> \* dec\_ciphertext = new vector<char> (cstring\_input.begin(), cstring\_input.end());
113. //解密过程明文向量指针dec\_plaintext
114. vector<char> \* dec\_plaintext;
115. //移位加密
116. dec\_plaintext=shift\_decrypt(dec\_ciphertext, offset);
117. //输出解密后的明文
118. cout << "解密结果：\n";
119. for(int i = 0; i < (\*dec\_plaintext).size(); i++)
120. {
121. cout<<(\*dec\_plaintext)[i];
122. }
123. cout<<endl<<endl;
124. system("pause");
125. return 0;
126. }
127. 移位密码攻击
128. 暴力破解实现结果



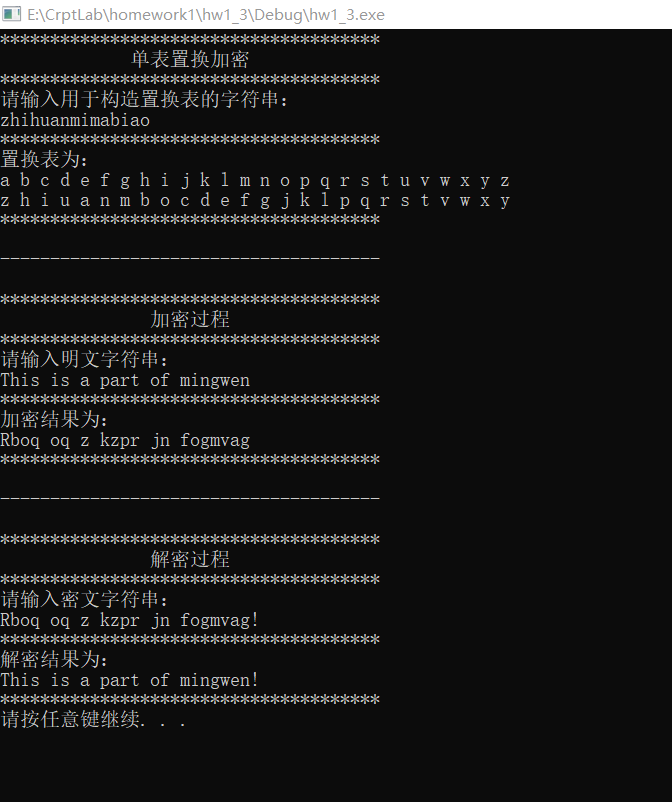
可以通过观察结果，得到秘钥(偏移量)为16

1. 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;
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. //破解过程密文向量指针crk\_ciphertext
14. vector<char> \* crk\_ciphertext = new vector<char> (cstring\_crack.begin(), cstring\_crack.end());
15. //破解过程明文向量指针crk\_plaintext
16. vector<char> \* crk\_plaintext = new vector<char>;
17. for(;offset<=25;offset++ )
18. {
19. cout << "如果移位位数为"<<offset<<"，则明文为：";
20. crk\_plaintext=shift\_decrypt(crk\_ciphertext, offset);
22. for(int i = 0; i < (\*crk\_plaintext).size(); i++)
23. {
24. cout<<(\*crk\_plaintext)[i];
25. }
26. cout<<endl;
27. }
28. system("pause");
29. return 0;
30. }
31. 单表置换密码
32. 加密流程图
33. 程序实现结果



1. 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. //加密过程：根据置换表rpltbl\_vector对明文plaintext进行加密
8. vector<char>\* encrypt(vector<char>\* plaintext, vector<char>\* rpltbl\_vector)
9. {
10. vector<char> \* ciphertext= new vector<char>;
11. for(int i = 0; i < (\*plaintext).size(); i++)
12. {
13. //小写字母
14. if((\*plaintext)[i]>= 97 && (\*plaintext)[i]<= 122)
15. {
16. (\* ciphertext).push\_back( (\*rpltbl\_vector) [((\*plaintext)[i]-97)] );
17. continue;
18. }
19. //大写字母
20. if((\*plaintext)[i]>= 65 && (\*plaintext)[i]<= 90)
21. {
22. (\* ciphertext).push\_back( (\*rpltbl\_vector)[(\*plaintext)[i]-65] - 32 );
23. continue;
24. }
25. //非字母内容，默认不加密
26. (\* ciphertext).push\_back((\*plaintext)[i]);
27. }
28. return ciphertext;
29. }
30. //解密过程：根据置换表rpltbl\_vector对密文ciphertext进行解密
31. vector<char>\* decrypt(vector<char>\* ciphertext, vector<char>\* rpltbl\_vector)
32. {
33. vector<char> \* plaintext= new vector<char>;
34. vector<char>::iterator result;
35. vector<char>::iterator first = (\*rpltbl\_vector).begin();
36. for(int i = 0; i < (\*ciphertext).size(); i++)
37. {
38. //小写字母
39. if((\*ciphertext)[i]>= 97 && (\*ciphertext)[i]<= 122)
40. {
41. // 在\*rpltbl\_vector 中 查找值为 (\*ciphertext)[i]的那一项的下标
42. result = find((\*rpltbl\_vector).begin(),(\*rpltbl\_vector).end(),(\*ciphertext)[i]);
43. //使用distance根据迭代器获得数组下标
44. (\*plaintext).push\_back( (char) ( 97+ distance(first,result) ) );
45. continue;
46. }
47. //大写字母
48. if((\*ciphertext)[i]>= 65 && (\*ciphertext)[i]<= 90)
49. {
50. // 在\*rpltbl\_vector 中 查找值为 (\*ciphertext)[i]+32的那一项的下标
51. result = find((\*rpltbl\_vector).begin(),(\*rpltbl\_vector).end(),((\*ciphertext)[i]+32) );
52. //使用distance根据迭代器获得数组下标
53. (\*plaintext).push\_back( (char) ( 65+ distance(first,result) ) );
54. continue;
55. }
56. //非字母内容，默认不解密
57. (\* plaintext).push\_back((\*ciphertext)[i]);
58. }
59. return plaintext;
60. }
62. //根据用户输入字符串构造置换表
63. vector<char>\* mk\_replace\_tbl(vector<char>\* mktbl\_vector)
64. {
65. vector<char> \* rpltbl\_vector= new vector<char>;
66. //用于存储哪些字母被输入字符串所包含
67. //0代表未被包含
68. int flag[26]={0};
69. for (int i = 0; i < (\*mktbl\_vector).size(); i++)
70. {
71. //大写字母转为小写
72. if ((\*mktbl\_vector)[i] >= 65 && (\*mktbl\_vector)[i] <= 90)
73. {
74. (\*mktbl\_vector)[i] += 32;
75. }
76. //(\*mktbl\_vector)[i]是字母
77. if ( (\*mktbl\_vector)[i] >= 97 && (\*mktbl\_vector)[i] <= 122 )
78. {
79. //查找(\*mktbl\_vector)[i]是否在\*rpltbl\_vector中
80. vector<char>::iterator result = find((\*rpltbl\_vector).begin(),(\*rpltbl\_vector).end(),(\*mktbl\_vector)[i]);
82. //如果不在\*rpltbl\_vector中的话
83. if(result==(\*rpltbl\_vector).end())
84. {
85. //插入到rpltbl\_vector向量中
86. (\* rpltbl\_vector).push\_back((\*mktbl\_vector)[i]);
87. //更新flag表
88. flag[(\*mktbl\_vector)[i]-97]=1;
89. }
90. }
91. }
92. for(int i=0;i<26;i++)
93. {
94. if(flag[i]==0)
95. (\* rpltbl\_vector).push\_back((char)(i+97));
96. }
97. return rpltbl\_vector;
98. }
99. int main()
100. {
101. string string\_mktbl;//构造置换表时输入的字符串
102. string mstring\_input;//加密过程用户输入的明文字符串
103. string cstring\_input;//解密过程用户输入的密文字符串
104. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
105. cout << " 单表置换加密 "<<endl;
106. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
107. cout << "请输入用于构造置换表的字符串：\n";
108. getline(cin, string\_mktbl);
109. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
110. //构造置换表所使用的向量指针crk\_ciphertext
111. vector<char> \* mktbl\_vector= new vector<char> (string\_mktbl.begin(), string\_mktbl.end());
112. //存储置换表的向量指针rpltbl\_vector
113. vector<char> \* rpltbl\_vector= new vector<char>;
114. rpltbl\_vector = mk\_replace\_tbl(mktbl\_vector);
115. cout << "置换表为：\n";
116. for(int i = 0; i < (\*rpltbl\_vector).size(); i++)
117. {
118. cout<<(char)(97+i)<<" ";
119. }
120. cout<<endl;
121. for(int i = 0; i < (\*rpltbl\_vector).size(); i++)
122. {
123. cout<<(\*rpltbl\_vector)[i]<<" ";
124. }
125. cout<<endl;
126. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
127. cout <<endl;
128. cout << "--------------------------------------"<<endl;
129. cout<<endl;
130. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
131. cout << " 加密过程 "<<endl;
132. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
133. cout << "请输入明文字符串：\n";
134. getline(cin,mstring\_input);
135. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
136. cout << "加密结果为：\n";
137. //存储加密过程的明文
138. vector<char> \* enc\_plaintext= new vector<char> (mstring\_input.begin(), mstring\_input.end()) ;
139. //存储加密过程的密文
140. vector<char> \* enc\_ciphertext= new vector<char>;
141. //加密
142. enc\_ciphertext = encrypt(enc\_plaintext, rpltbl\_vector);
143. //输出加密结果
144. for(int i = 0; i < (\*enc\_ciphertext).size(); i++)
145. {
146. cout<<(\*enc\_ciphertext)[i];
147. }
148. cout<<endl;
149. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
150. cout <<endl;
151. cout << "--------------------------------------"<<endl;
152. cout <<endl;
153. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
154. cout << " 解密过程 "<<endl;
155. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
156. cout << "请输入密文字符串：\n";
157. getline(cin,cstring\_input);
158. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
159. cout << "解密结果为：\n";
160. //存储解密过程的密文
161. vector<char> \* dec\_ciphertext= new vector<char> (cstring\_input.begin(), cstring\_input.end()) ;
162. //存储解密过程的明文
163. vector<char> \* dec\_plaintext= new vector<char>;
164. //解密
165. dec\_plaintext = decrypt(dec\_ciphertext, rpltbl\_vector);
166. //输出解密结果
167. for(int i = 0; i < (\*dec\_plaintext).size(); i++)
168. {
169. cout<<(\*dec\_plaintext)[i];
170. }
171. cout<<endl;
172. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
173. system("pause");
174. return 0;
175. }
176. 移位密码攻击

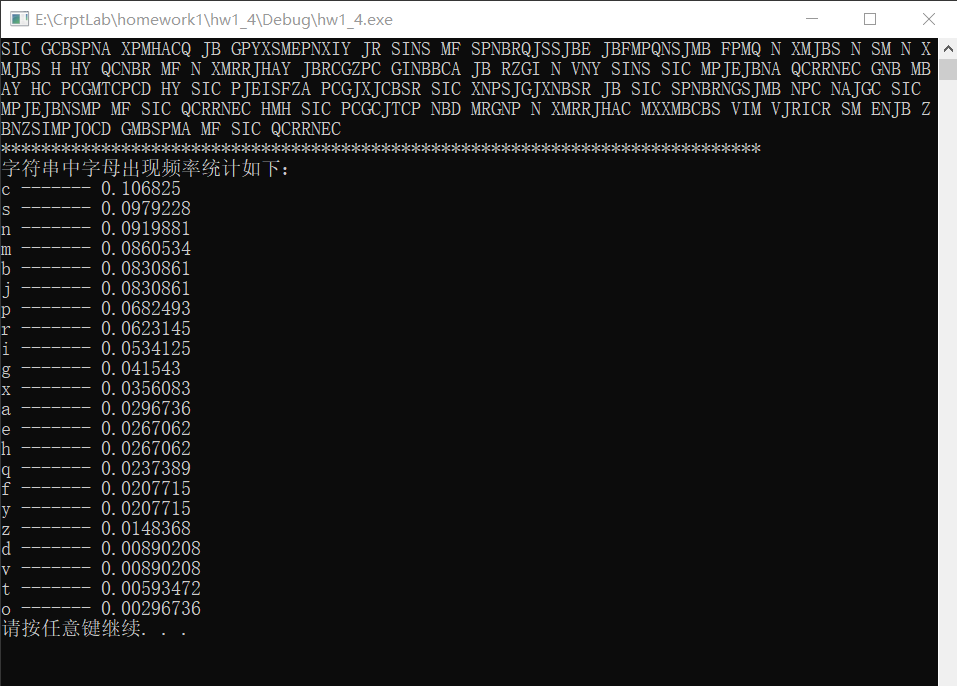
攻击的思路是：

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



**明文中字母出现频率统计结果：**

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

1. 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. int main()
8. {
9. string input\_str;
10. vector<int>\* freq\_tbl= new vector<int>;
11. vector<int>\* letter\_tbl= new vector<int>;
13. //记录(\*freq\_tbl)按频次由低到高排序后,每个频次对应的字母
14. char letter\_record[26] = {'a'};
15. for(int i=1;i<=26;i++)
16. {
17. (\*freq\_tbl).push\_back(0);
18. (\*letter\_tbl).push\_back(0);
19. }
20. cout<<"请输入密文字符串：\n";
21. getline(cin, input\_str);
22. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
23. cout<<"字符串中字母出现频率统计如下：\n";
24. int total\_letter\_num = 0;
25. //遍历输入字符串，记录字母频次
26. for (int i = 0; i<input\_str.length(); i++)
27. {
28. if ( input\_str[i] >= 'A' && input\_str[i] <= 'Z' )
29. {
30. (\*freq\_tbl)[ input\_str[i] - 'A']++;
31. (\*letter\_tbl)[ input\_str[i] - 'A']++;
32. total\_letter\_num++;
33. continue;
34. }
35. if ( input\_str[i] >= 'a' && input\_str[i] <= 'z' )
36. {
37. (\*freq\_tbl)[ input\_str[i] - 'a']++;
38. (\*letter\_tbl)[ input\_str[i] - 'a']++;
39. total\_letter\_num++;
40. continue;
41. }
42. }
43. //freq\_tbl按字母频次重排(默认升序排列)
44. sort((\*freq\_tbl).begin(),(\*freq\_tbl).end());
45. for(int f=25;f>=0;f--)
46. //遍历(\*freq\_tbl)，(\*freq\_tbl)按频率值由低到高顺序存储字母频次值
47. {
48. for(int l=0;l<26;l++)
49. //遍历(\*letter\_tbl)，(\*letter\_tbl)按字母顺序存储字母频次值
50. {
51. if( (\*letter\_tbl)[l] == (\*freq\_tbl)[f] )
52. {
53. letter\_record[f] = 'a'+l;
54. //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
55. (\*letter\_tbl)[l] = -1;
56. break;
57. }
58. }
59. }
60. for (int k = 25; k >= 0; k--)
61. {
62. if((\*freq\_tbl)[k]!=0)
63. cout << letter\_record[k]<< " ------- " <<double((\*freq\_tbl)[k])/double(total\_letter\_num)<< endl;
64. }
65. system("pause");
66. return 0;
67. }

**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** |  |  |  |  |  |  |

1. **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 MPJEJBNA QCRRNEC GNB MBAY HC PCGMTCPCD 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 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 pcgmtcpcd 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 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）



1. 手工调整置换表
2. 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，对换置换表中明文字母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：**



1. 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：**



1. 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：**



1. 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 → information ： **p → m**

**将明文m对应的密文设置为明文p当前对应的密文**

* doint → point ： **d → p**

**将明文p对应的密文设置为明文d当前对应的密文**

* dy→ by ： **d → b**

**将明文b对应的密文设置为明文d当前对应的密文**

* probuem → problem ： **u → l**

**将明文l对应的密文设置为明文u当前对应的密文**

* uentral → central ： **u → c**

**将明文c对应的密文设置为明文u当前对应的密文**

* cryptouraphy → cryptography ： **u → g**

**将明文g对应的密文设置为明文u当前对应的密文**

* day → way ： **d → w**

**将明文w对应的密文设置为明文d当前对应的密文**

* unauthoriked → unauthorized ： **k → 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：**

