# 密码学作业

**目录**

[第二章作业 1](#_Toc121608511)

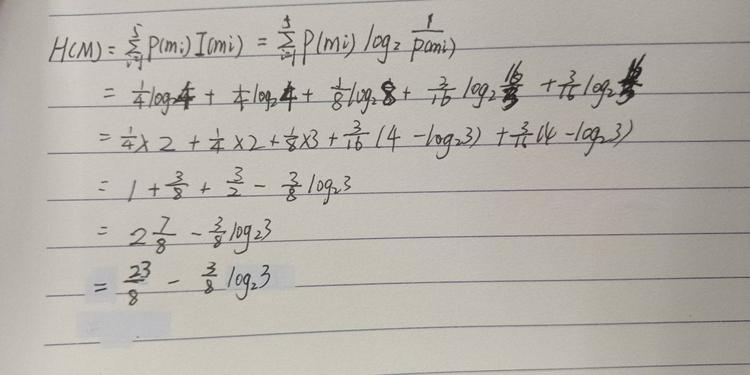
[第三章作业 1](#_Toc121608512)

[第四章作业1 11](#_Toc121608513)

[第四章作业2 12](#_Toc121608514)

# 第二章作业

(计算题)教材第二章课后习题5（5）：设密文空间有5个消息m，并且p(m1)=p(m2)=1/4，p(m3)=1/8，p(m4)=p(m5)=3/16,求H(m)

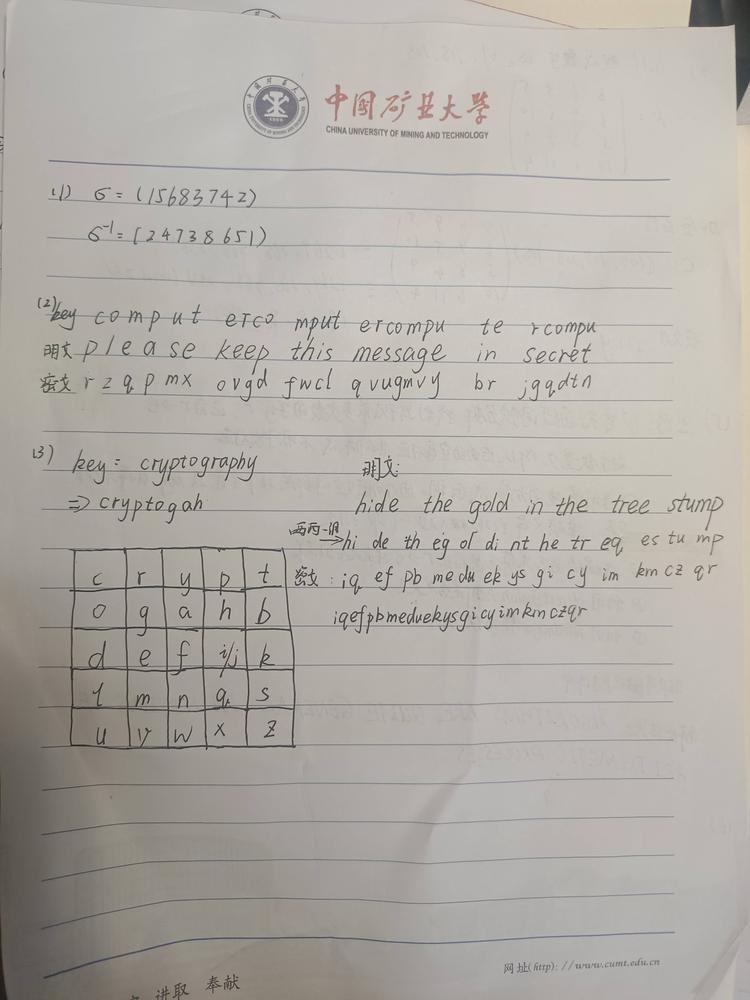


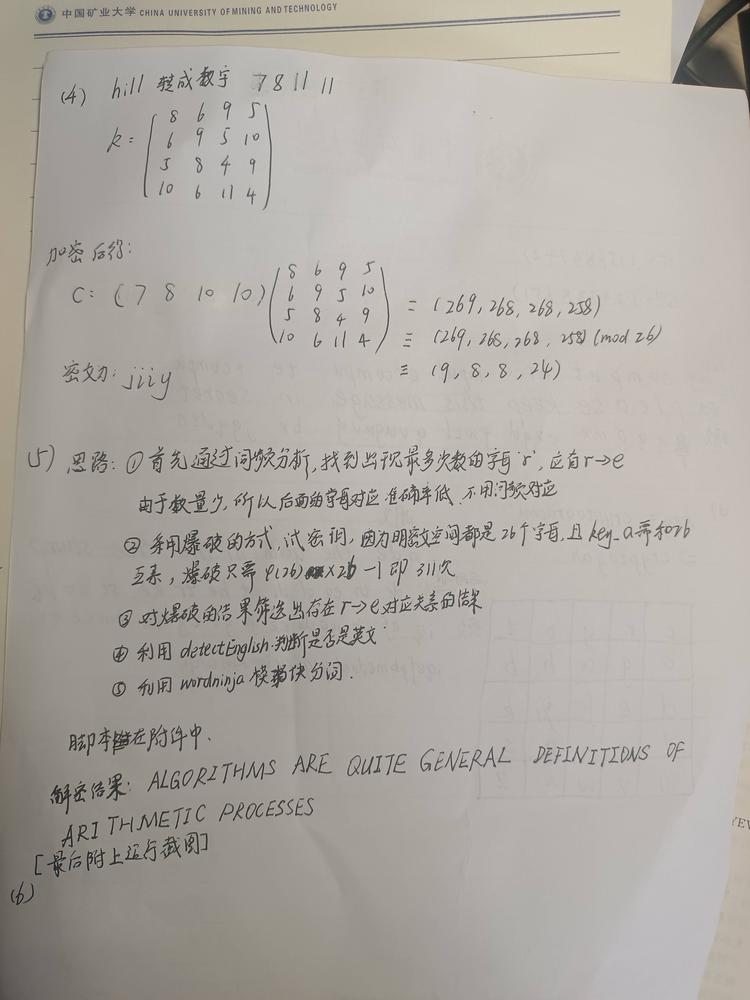
# 第三章作业

1.课本P65页简答题5的第1-5小题

2. YKHLBA JCZ SVIJ JZB LZVHI JCZ VHJ DR IZXKHLBA VSS RDHEI DR YVJV LBXSKYLBA YLALJVS IFZZXC CVI LEFHDNZY EVBTRDSY JCZ FHLEVHT HZVIDB RDH JCLI CVI WZZB JCZ VYNZBJ DR ELXHDZSZXJHDBLXI JCZ XDEFSZQLJT DR JCZ RKBXJLDBI JCVJ XVB BDP WZ FZHRDHEZY WT JCZ EVXCLBZ CVI HLIZB YHVEVJLXVSST VI V HZIKSJ DR JCLI JCVJ PZHZ DBXZ XDBILYZHZY IZXKHZ VHZ BDP WHZVMVWSZ

对上面单表替换密码进行解密，写出分析过程和最终的py3程序代码！





第5题仿射解密脚本

1. #Affine Cipher Hacker
2. **import** math,sys,detectEnglish,wordninja
3. **from** gmpy2 **import** invert
5. englishLetterFrea={'E':12.7,'T':9.06,'A':8.17,'O':7.51,'I':6.97,'N':6.75,'S':6.33,'H':6.09,'R':5.99,'D':4.25,'L':4.03,'C':2.78,'U':2.76,'M':2.41,'W':2.36,'F':2.23,'G':2.02,'Y':1.97,'P':1.93,'B':1.29,'V':0.98,'K':0.77,'J':0.15,'X':0.15,'Q':0.10,'Z':0.07}

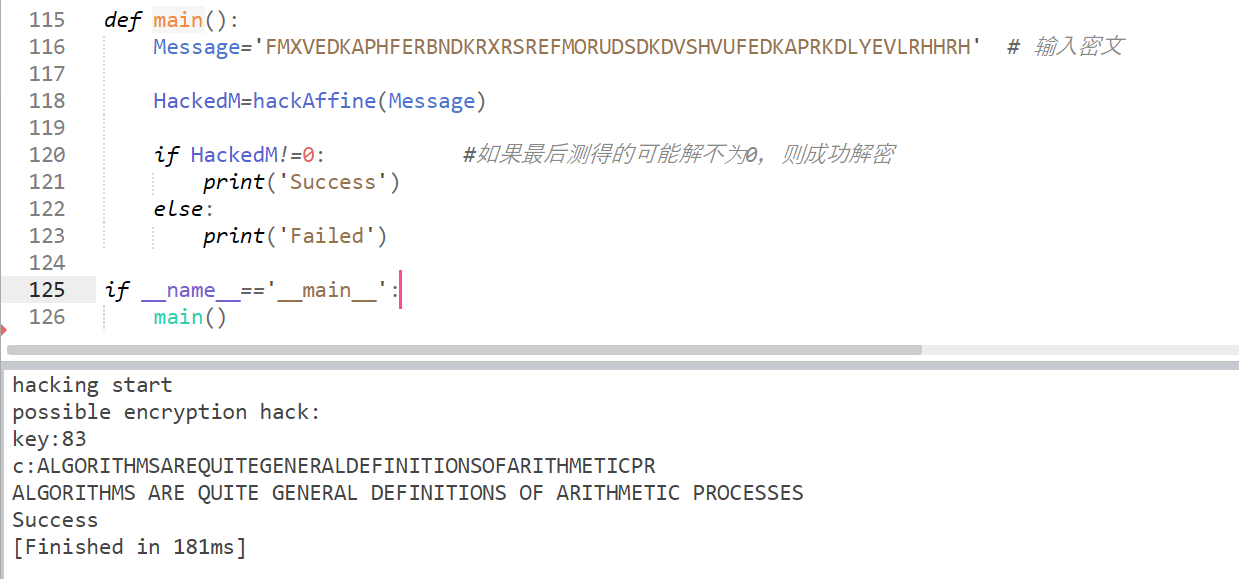
8. ETAOIN='ETAOINSHRDLCUMWFGYPBVKJXQZ'  #英文中词频排列顺序（降序）
10. Symbols='ABCDEFGHIJKLMNOPQRSTUVWXYZ'  #明文空间 密文空间
12. **def** getLetterCount(message):
14. #统计字符串中每个字母的数量
15. LetterCount={'A':0,'B':0,'C':0,'D':0,'E':0,'F':0,'G':0,'H':0,'I':0,'J':0,'K':0,'L':0,'M':0,'N':0,'O':0,'P':0,'Q':0,'R':0,'S':0,'T':0,'U':0,'V':0,'W':0,'X':0,'Y':0,'Z':0}
17. **for** i **in** message.upper():
18. **if** i **in** Symbols:
19. LetterCount[i]+=1
21. **return** LetterCount


25. **def** getItemAtIndexZero(x):
26. #返回变量的下标为0的值
27. **return** x[0]
29. **def** getFrequencyOrder(message):
30. #按照字母出现的次数排序
31. letterToFreq=getLetterCount(message) #字母到次数的映射
32. freqToLetter={} #次数到字母的映射
34. **for** i **in** Symbols:
35. **if** letterToFreq[i] **not** **in** freqToLetter:   #判断字母对应次数是否是字典freqToLetter的键，若不是则以该次数为键，以对于的字母为值；若是，则在最后追加一组键值对
36. freqToLetter[letterToFreq[i]]=[i]     #因为次数可能出现相同的情况
37. **else**:
38. freqToLetter[letterToFreq[i]].append(i)
40. **for** freq **in** freqToLetter:
41. freqToLetter[freq].sort(key=ETAOIN.find,reverse=True)  #将同一次数的键所对应的字母按照ETAOIN的降序排列(做一个约定)
42. freqToLetter[freq]=''.join(freqToLetter[freq])
44. freqPairs=list(freqToLetter.items())                 #将字典变成一个元组列表
45. freqPairs.sort(key=getItemAtIndexZero,reverse=True)  #按照元组列表索引为0的值排序
47. **return** freqPairs



52. **def** getKey(key):
53. #得到两个ab密钥
54. key\_a=key//len(Symbols)
55. key\_b=key%len(Symbols)
56. **return** (key\_a,key\_b)
58. **def** checkKeys(key\_a,key\_b,mode):
59. #检查密钥是否合法
60. **if** mode=='encrypt':
61. **if** key\_a==1 **or** key\_b==1:
62. sys.exit('太简单')
63. **if** key\_a<0 **or** key\_b<0 **or** key\_b>len(Symbols)-1:
64. sys.exit('不属于区间(0,len(Symbols))')
65. **if** math.gcd(key\_a,len(Symbols))!=1:
66. sys.exit('密钥a和模数不互素')
68. **def** decryptM(key,message):
69. #已知密钥的仿射解密
70. K1,K2=getKey(key)
71. checkKeys(K1,K2,'decrypt')
72. result=''
73. modInverseOfKeya=invert(K1,len(Symbols))    #求密钥a的模逆元
75. **for** i **in** message:
76. **if** i **in** Symbols:
77. Index=Symbols.find(i)
78. result+=Symbols[((Index-K2)\*modInverseOfKeya)%len(Symbols)]
79. **else**:
80. result+=i
82. **return** result
84. **def** hackAffine(message):
85. #爆破得到明文
86. cnt=0 #最后输出的可能解密结果数
88. Max=getFrequencyOrder(message)[0][1]
90. **for** i **in** message:
91. **if** i==Max:
92. ind = message.find(i)   #找到E所在的位置
93. **break**
94. #
95. **print**('hacking start')
97. **for** key **in** range(len(Symbols)\*\*2):   #爆破密钥
98. key\_a=getKey(key)[0]
99. **if** math.gcd(key\_a,len(Symbols))!=1:  #密钥a要和模数互素
100. **continue**
102. result=decryptM(key,message)  #将密钥带入根据公式求出对应明文
104. **if** result[ind]=='E':
105. #print('tried key %s...(%s)'%(key,result[:20]))
106. text=' '.join(wordninja.split(result))  #利用wordninja工具给字符串分词，这样才能用detectEnglish判断是否是英文
107. **if** detectEnglish.isEnglish(text):    #利用detectEnglish判断是否是英文
108. cnt+=1
109. **print**('possible encryption hack:')
110. **print**('key:%s'%(key))
111. **print**('c:'+result[:50])
112. **print**(text)                      #输出最后可能结果
113. **return** cnt
115. **def** main():
116. Message='FMXVEDKAPHFERBNDKRXRSREFMORUDSDKDVSHVUFEDKAPRKDLYEVLRHHRH'  # 输入密文
118. HackedM=hackAffine(Message)
120. **if** HackedM!=0:           #如果最后测得的可能解不为0，则成功解密
121. **print**('Success')
122. **else**:
123. **print**('Failed')
125. **if** \_\_name\_\_=='\_\_main\_\_':
126. main()

运行结果：



第六题单表替换脚本：

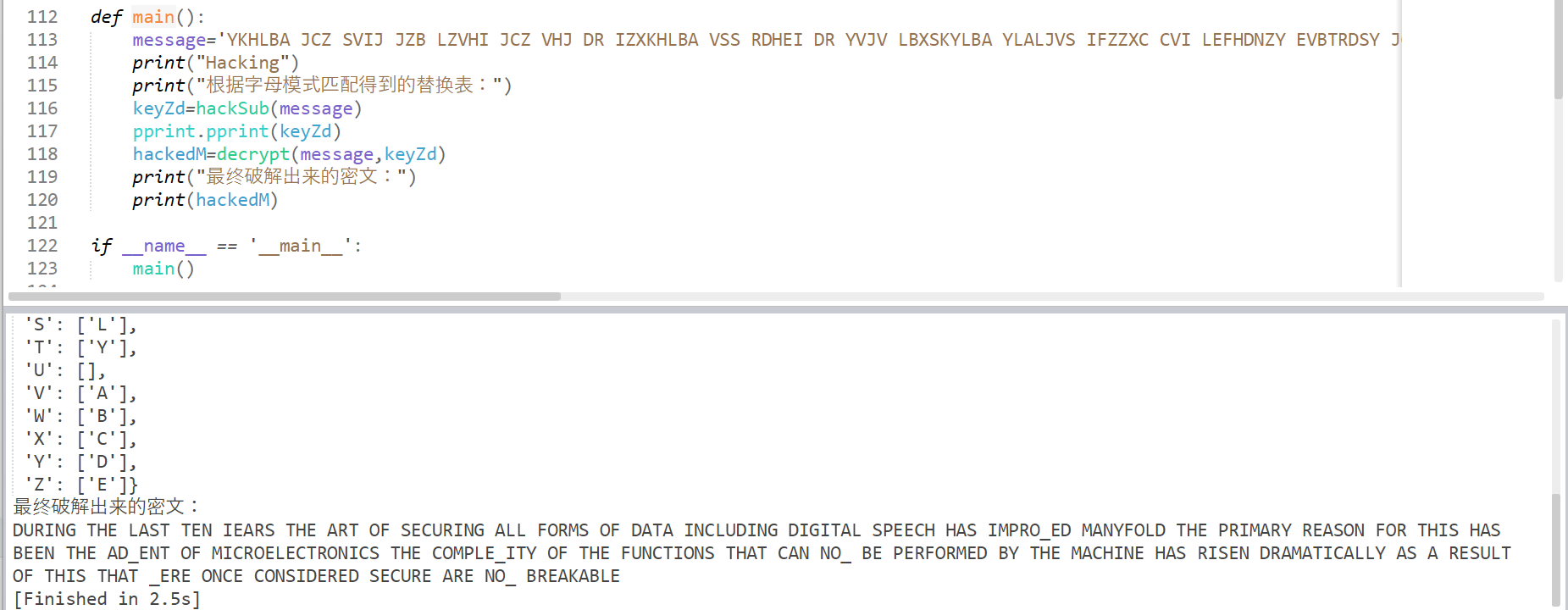
1. # 单表替换解密
3. **import** pprint,os,makeWordPatterns,re,copy
5. **if** **not** os.path.exists('wordPattern.py'): #不存在wordPattern就运行得到wordPattern
6. makeWordPatterns.main()
7. **import** wordPatterns
9. Letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ' #字母表
10. reg=re.compile('[^A-Z\s]') #固定一个匹配模式，匹配所有不是A-Z或者空格的字符
12. **def** getBlankZd():
13. #返回一个空字典
14. **return** {'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': []}
16. **def** addLettersToZd(letterZd,cipherword,candidate):
17. #根据相同模式的单词，给出密文字母可能对应明文字母的字典
18. letterZd=copy.deepcopy(letterZd) #这样才能创建一个字典副本，不然只是传的引用
19. **for** i **in** range(len(cipherword)):
20. **if** candidate[i] **not** **in** letterZd[cipherword[i]]:
21. letterZd[cipherword[i]].append(candidate[i])
22. **return** letterZd
24. **def** intersectZd(ZdA,ZdB):
25. #取两个字典的交集
26. intersectZd=getBlankZd()
27. **for** i **in** Letters:
28. **if** ZdA[i]==[]:
29. intersectZd[i]=copy.deepcopy(ZdB[i])
30. **elif** ZdB[i]==[]:
31. intersectZd[i]=copy.deepcopy(ZdA[i])
32. **else**:
33. **for** j **in** ZdA[i]:
34. **if** j **in** ZdB[i]:
35. intersectZd[i].append(j)
36. #print(intersectZd)
37. **return** intersectZd
39. **def** removeSolvedLettersFromZd(letterZd):
40. #根据
41. letterZd=copy.deepcopy(letterZd)
42. loopAgain=True
43. **while** loopAgain:
44. loopAgain=False
46. solvedList=[]
47. **for** i **in** Letters:
48. **if** len(letterZd[i])==1:
49. solvedList.append(letterZd[i][0])
51. **for** i **in** Letters:
52. **for** s **in** solvedList:
53. **if** len(letterZd[i])!=1 **and** s **in** letterZd[i]:
54. letterZd[i].remove(s)
55. **if** len(letterZd[i])==1:
56. loopAgain=True
58. #print(letterZd)
59. **return** letterZd
61. **def** hackSub(message):
62. #找出对应表
63. intersectedZd=getBlankZd()
65. cipherwordList=reg.sub('',message.upper()).split()
66. #print(cipherwordList)
68. **for** i **in** cipherwordList:
69. newZd=getBlankZd()
71. wordPattern=makeWordPatterns.getPattern(i)
72. **if** wordPattern **not** **in** wordPatterns.allPatterns:
73. **continue**
74. **for** j **in** wordPatterns.allPatterns[wordPattern]:
75. newZd=addLettersToZd(newZd,i,j)
76. #print(newZd)
77. intersectedZd=intersectZd(intersectedZd,newZd)
79. **return** removeSolvedLettersFromZd(intersectedZd)
81. **def** desiglesub(key, message):
82. #根据替换关系解密
83. result=''
84. **for** i **in** message:
85. **if** i.upper() **in** key:
86. index=key.find(i.upper())
87. **if** i.isupper():
88. result=result+Letters[index].upper()
89. **else**:
90. result=result+Letters[index].lower()
91. **else**:
92. result=result+i
94. **return** result


98. **def** decrypt(ciphertext,letterZd):
99. #生成密钥，解密
100. key=['x']\*len(Letters)
101. **for** i **in** Letters:
102. **if** len(letterZd[i])==1:
103. keyIndex=Letters.find(letterZd[i][0])
104. key[keyIndex]=i
105. **else**:
106. ciphertext=ciphertext.replace(i.lower(),'\_')
107. ciphertext=ciphertext.replace(i.upper(),'\_')
108. key=''.join(key)
110. **return** desiglesub(key,ciphertext)
112. **def** main():
113. message='YKHLBA JCZ SVIJ JZB LZVHI JCZ VHJ DR IZXKHLBA VSS RDHEI DR YVJV LBXSKYLBA YLALJVS IFZZXC CVI LEFHDNZY EVBTRDSY JCZ FHLEVHT HZVIDB RDH JCLI CVI WZZB JCZ VYNZBJ DR ELXHDZSZXJHDBLXI JCZ XDEFSZQLJT DR JCZ RKBXJLDBI JCVJ XVB BDP WZ FZHRDHEZY WT JCZ EVXCLBZ CVI HLIZB YHVEVJLXVSST VI V HZIKSJ DR JCLI JCVJ PZHZ DBXZ XDBILYZHZY IZXKHZ VHZ BDP WHZVMVWSZ '
114. **print**("Hacking")
115. **print**("根据字母模式匹配得到的替换表：")
116. keyZd=hackSub(message)
117. pprint.pprint(keyZd)
118. hackedM=decrypt(message,keyZd)
119. **print**("最终破解出来的密文：")
120. **print**(hackedM)
122. **if** \_\_name\_\_ == '\_\_main\_\_':
123. main()

其中makeWordPatterns模块脚本为：

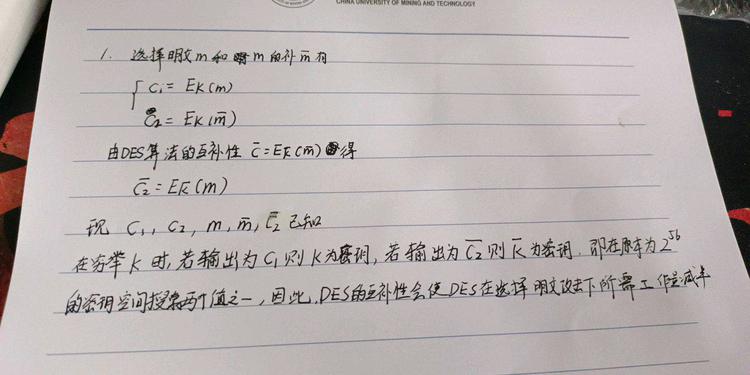
1. #将所有单词的模式(例如all是(0.1.2))统计出来，得到模式到字母的一对多的映射
3. **import** pprint  #pprint 美化输出
5. #思路：1.遍历单词，将单个单词算出模式，然后放入模式映射单词的字典中
6. #2.算出模式的函数的思路：创建当个单词字母映射数字的字典和排好模式的列表，最后用join输出成固定格式
8. **def** getPattern(word):
9. #返回单词模式
10. word=word.upper()
11. wordList=[] #排好模式的列表
12. wordZd={} #字母映射数字的字典
13. num=0
15. **for** i **in** word:
16. **if** i **not** **in** wordZd:
17. wordZd[i]=str(num)
18. num+=1
19. wordList.append(wordZd[i])
21. **return** '.'.join(wordList)
23. **def** main():
24. #找到所有单词的模式，并在字典中储存，返回模式映射单词的字典
25. allPattern={}
27. file=open('dictionary.txt','r')
28. allWords=file.read().split() #读出所有单词，并分片成列表形式
29. file.close()
31. **for** i **in** allWords:
32. pattern=getPattern(i)
33. **if** pattern **not** **in** allPattern:
34. allPattern[pattern]=[i]
35. **else**:
36. allPattern[pattern].append(i)
38. file=open('wordPatterns.py','w')
39. file.write('allPatterns=')
40. file.write(pprint.pformat(allPattern))
41. file.close()
43. **if** \_\_name\_\_ == '\_\_main\_\_':
44. main()

运行结果：

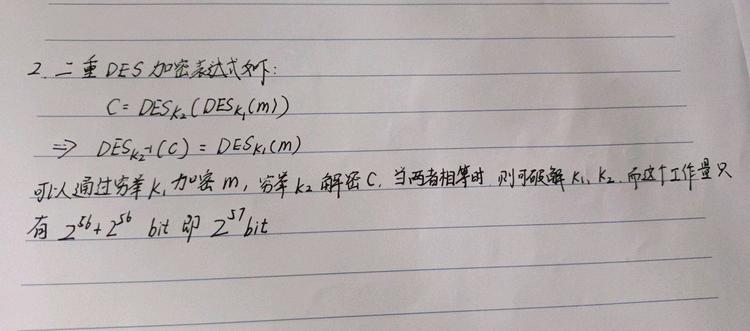


# 第四章作业1

1. (简答题) DES算法具有互补性，而这个特性会使得DES在选择明文攻击下所需的工作量减半，请说明原因。



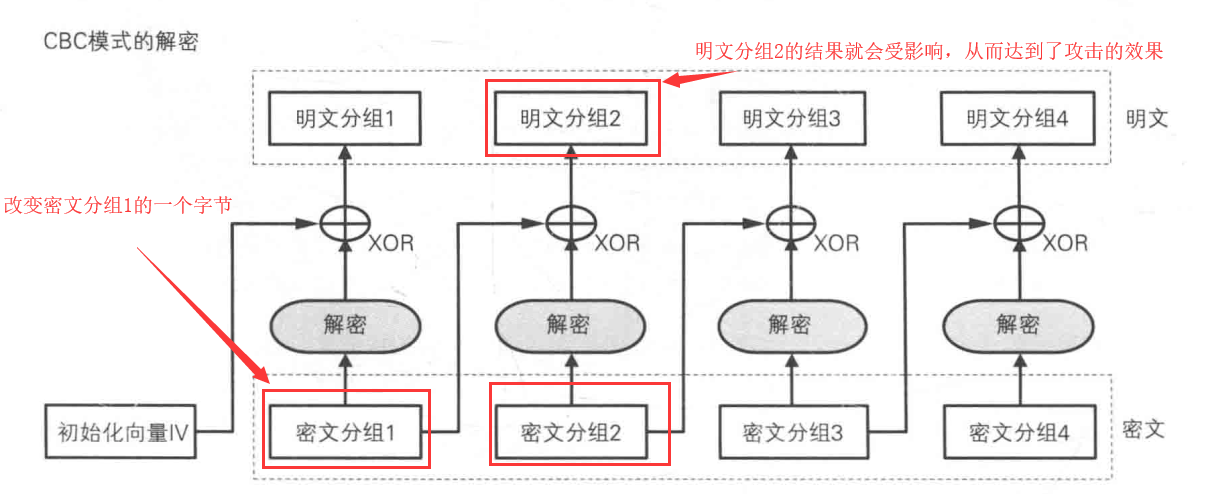
2. (简答题) 为什么二重DES并不像人们想象的那样可提高密钥长度到112比特，而相当于57比特？请说明原因。



# 第四章作业2

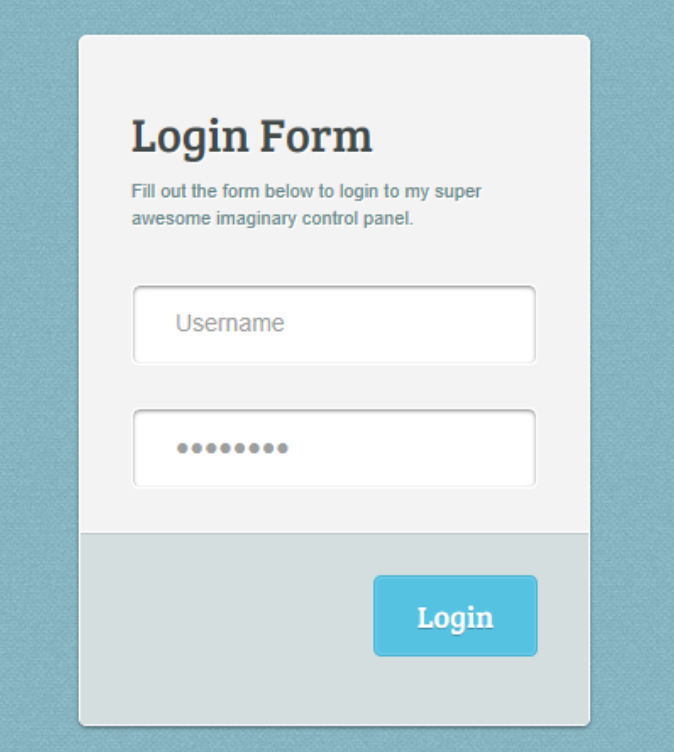
1. (简答题) 搜集一道关于CBC翻转攻击的CTF竞赛题目，提交题目、题解（具体说明原理）和解题脚本

(1)cbc字节翻转攻击的原理



(2)bugku-login4

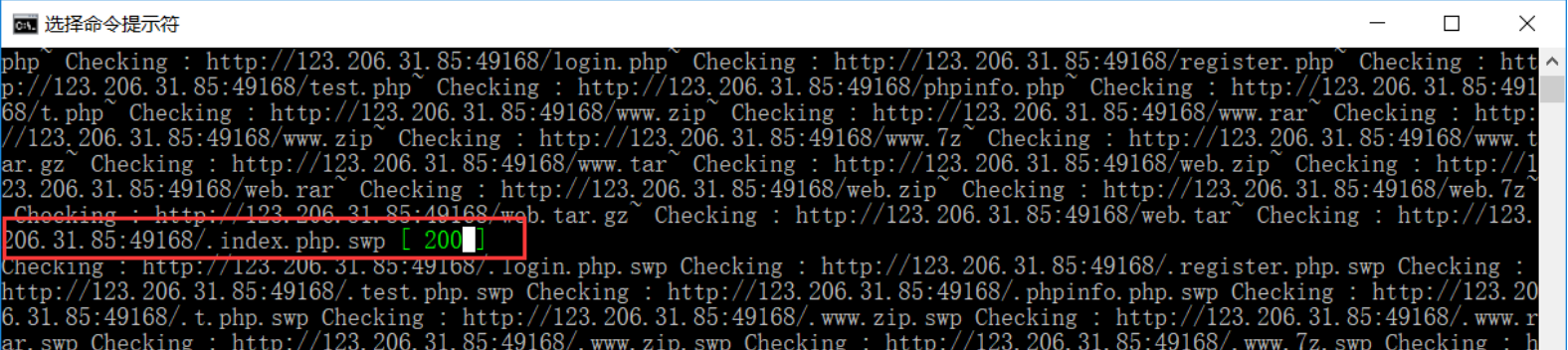
打开是一个登陆界面，随便输入一个账号密码就能登陆，但是admin不能登陆



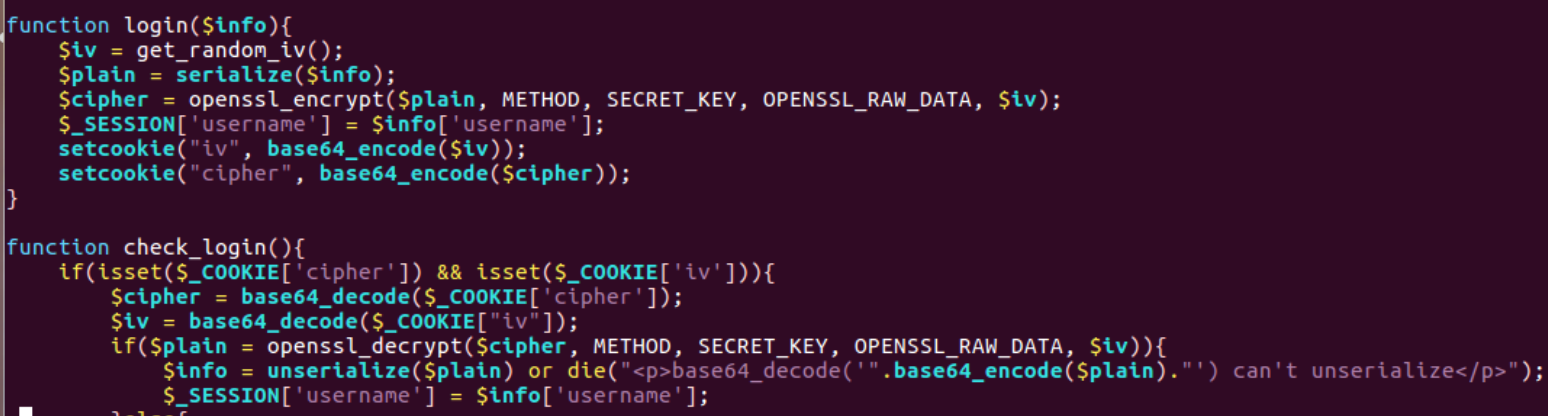
抓包发现，有个iv和cipher，猜测是CBC模式的加密

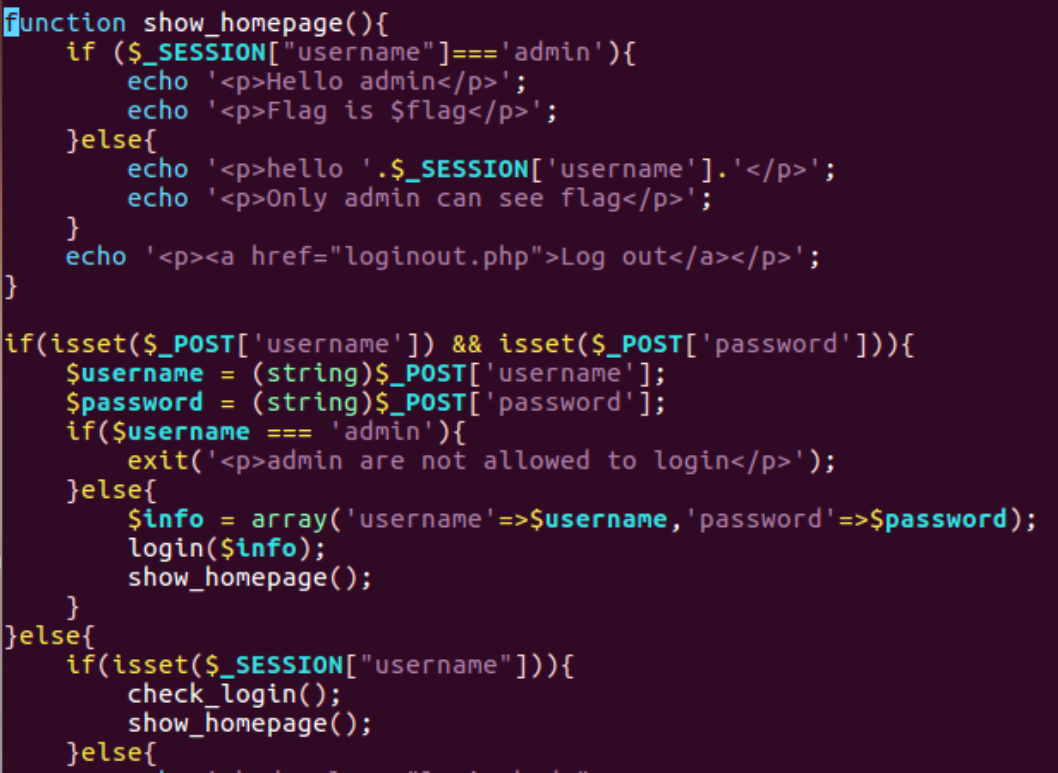


扫描一波发现有swp源码泄露



vim -r index.php.swp恢复后得到源代码，审计关键代码





进行代码审计，可以看到要想获得 flag 需要以 admin 的身份登陆，但是又禁止了直接用 admin 登录，但是是通过获取 cookie 中的值来判断是否为 admin，所以就需要利用 cookie 伪造登录。这里把登录的用户名及其密码存入数组，序列化后进行 AES-CBC 模式的加密，其中 iv 和 cipher 以 cookie 储存，可以控制，导致存在攻击的可能，即利用 CBC 字节翻转攻击。然后以账号为admia，密码为12345登陆得到的明文是：a:2:{s:8:"username";s:5:"admia";s:8:"password";s:5:"12345"}然后16个字节分组得到

1. s:2:{s:8:"userna
2. me";s:5:"admia";
3. s:8:"password";s

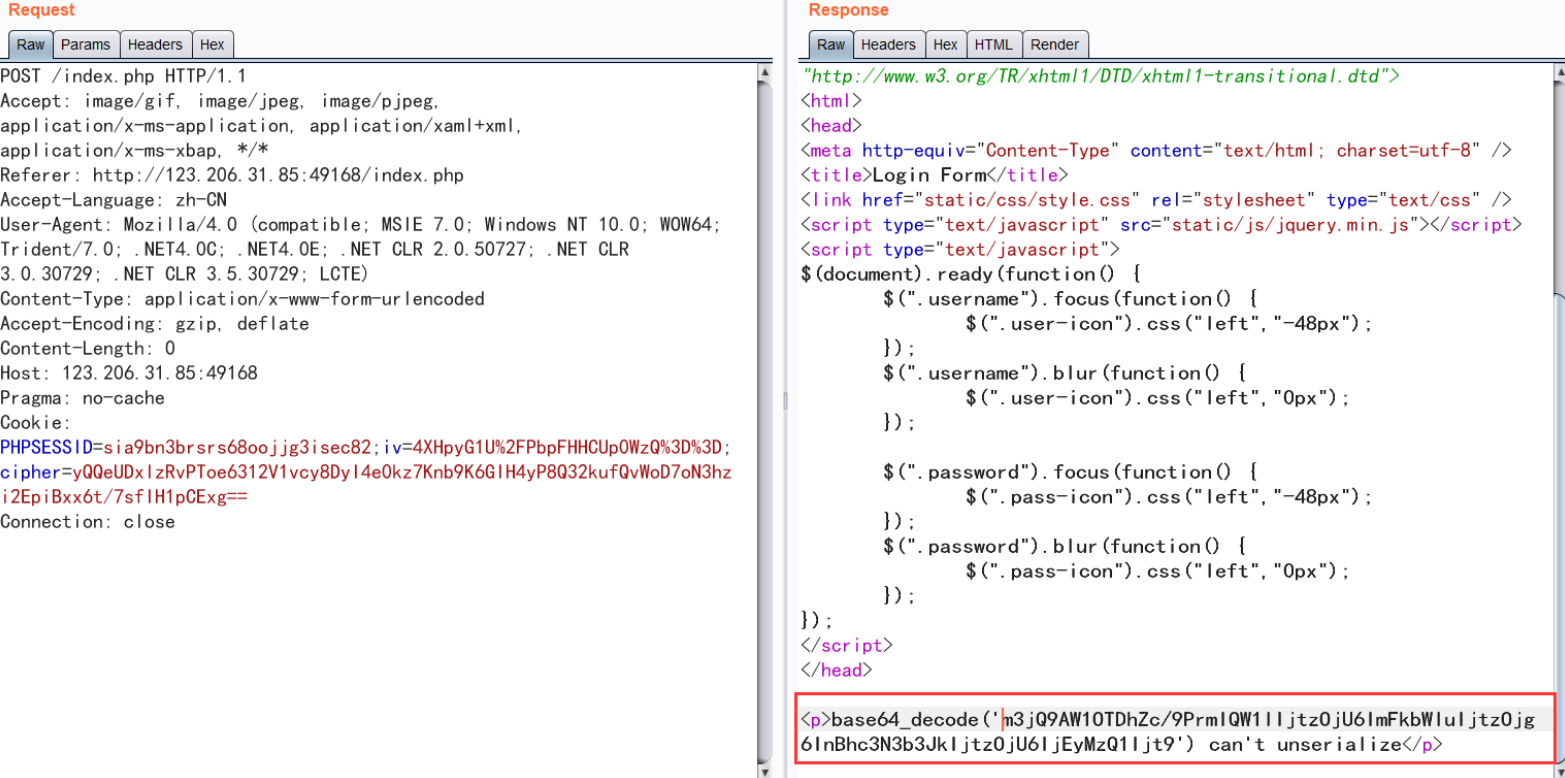
④:3:"12345";}

所以我们要翻转的是第二组的“a”翻转为“n”，所以要改变第一组的密文从而达到攻击的效果.

利用cbc字节翻转代码如下：

1. **import** base64
2. cipher="yQQeUDxlzRvPToe631KV1vcy8DyI4e0kz7Knb9K6GIH4yP8Q32kufQvWoD7oN3hzi2EpiBxx6t/7sfIH1pCExg=="
3. plain=base64.b64decode(cipher).decode('unicode\_escape')
4. result=plain[0:13]+chr(ord(plain[13])^ord("n")^ord("a"))+plain[14:]
5. **print**(base64.b64encode(result.encode('latin-1')).decode())
6. #print yQQeUDxlzRvPToe6312V1vcy8DyI4e0kz7Knb9K6GIH4yP8Q32kufQvWoD7oN3hzi2EpiBxx6t/7sfIH1pCExg==

POST过去提示不能正常反序列化，因为修改了第一组的密文，导致第一组的密文与iv xor会出错，从而导致了第一组明文不能正常解密，所以我们还要对iv进行修改



代码如下

1. **import** base64
2. cipher="wRPT3VONV2zFV6D2PbHjIm1lIjtzOjU6ImFkbWluIjtzOjg6InBhc3N3b3JkIjtzOjU6IjEyMzQ1Ijt9"
3. plain=base64.b64decode(cipher).decode('unicode\_escape')
4. oldiv=base64.b64decode("uxrq4TtskqrNJh7JUZV9rg==").decode('unicode\_escape')
5. one='a:2:{s:8:"userna'
6. iv=""
7. **for** i **in** range(0,16):
8. iv=iv+chr(ord(one[i])^ord(plain[i])^ord(oldiv[i]))
9. **print**(base64.b64encode(iv.encode('latin-1')).decode())#GzMLBhOS//4yU8tMCVbw7Q==

将得到的值修改为 iv，刷新页面得到 flag

