现代密码学第一次作业

郑凯文

2021 年 4 月 6 日

1 Enigma破译（波兰雷臼夫斯基法）

1.1 算法原理

仿照Enigma M3，我模拟的Enigma由Rotor、Reflector和Plugboard组成。其中转子的编号顺序与其相反，输入到输出的过程为Input->Plugboard->Rotor1->Rotor2->Rotor3->Reflector->Rotor3->Rotor2->Rotor1->Plugboard->Output。需要破译的内容包括3个Roter的类型、3个Rotor的初始设置、1个Reflector的类型和Plugboard交换的情况。

下面对如上Enigma机可能的设置数进行定量分析。

- 设Rotor共有m种，3个Rotor类型互不相同，可能的排列数为$m(m-1)(m-2)$
- 每个Rotor有26种初始设置，共26^3种
- Reflector有n种
- 设交换$2p$对字母，这相当于首先选择无序的$2p$个元素，从小到大排列后放到p个桶内，每个桶容纳2个元素，第一个是未选择的第一个元素，第二个分别有$2p-1, 2p-3, \cdots$种选择，于是总的情况数是

$$C_{26}^{2p} \times (2p-1) \times (2p-3) \times \cdots \times 1 = \frac{26!}{(26-2p)!p!2^p} \tag{1}$$

Enigma M3中，上述$m = 5, n = 2$，且能够得到每个类型的Rotor、Reflector的代换表（见index.js）。波兰雷臼夫斯基法的原理是，利用截获密文的前6个字母，绕过Plugboard，破译Rotor类型、Reflector类型和3个Rotor的初始设置（即日密钥）。已知密文的前6个字母是信息密钥两次加密获得，设信息密钥为xyz，密文为$abcdef$，即$xyzxyz \to abcdef$，或

$$x(A_0) = a,\ y(A_1) = b,\ z(A_2) = c,\ x(A_3) = d,\ y(A_4) = e,\ z(A_5) = f \tag{2}$$

利用Enigma的自反性，$(A_i)(A_i)$为恒等映射，可以得到

$$a(A_0)(A_3) = d,\ b(A_1)(A_4) = e,\ c(A_2)(A_5) = f \tag{3}$$

这样就完全绕过了信息密钥（即原文），只利用足够多的密文即可得到三个映射$(A_0)(A_3), (A_1)(A_4), (A_2)(A_5)$的映射表。若没有Plugboard，直接遍历并对照三个映射表即可。加入了Plugboard后，仍然具有的不变量是映射形成的环的长度和数量（容易证明，若交换前有环$A_1 \to A_2 \to A_3 \to ... \to A_1$，

则交换后一定有环$Swap(A_1) \rightarrow Swap(A_2) \rightarrow Swap(A_3) \rightarrow ... \rightarrow Swap(A_1)$）。我们仍可以遍历并对其进行比对，即使得不到唯一解，也可以大大缩小解空间。最后再针对可能的解，破译Plugboard的设置（见下节）。

1.2 代码说明

代码见`enigma.py`。首先我实例化了Enigma及其各个组件，并实现了`Setting`类型以储存一组设置。Enigma进位中的double-stepping是一个容易错误的点，群里讨论的很多，其实归纳起来也很简单：Rotor1必进位，Rotor2进位当且仅当Rotor1或Rotor2处于notch位，Rotor3进位当且仅当Rotor2处于notch位。我设置了`test_double_stepping()`函数来验证我的实现的正确性，选取了类型1、类型2（notch为E）、类型3（notch为V）的Rotor分别处于ADV和AEV的初始位置

```
def test_double_stepping():
    setting1 = Setting([2, 1, 0], [to_num("V"), to_num("D"), to_num("A")], 0, [(to_num("A"),
                                                                                 to_num("C"))])
    setting2 = Setting([2, 1, 0], [to_num("V"), to_num("E"), to_num("A")], 0, [(to_num("A"),
                                                                                 to_num("C"))])
    assert crypto(setting1, "CCCCC") == "QIBMG"
    assert crypto(setting1, "QIBMG") == "CCCCC"
    assert crypto(setting2, "CCCCC") == "GIBMG"
    assert crypto(setting2, "GIBMG") == "CCCCC"
    print("Test double-stepping ok.")
```

以上加解密结果与网站保持一致。

算法的两个辅助函数是`poland_mapping(setting: Setting)`和`poland_ring(mappings)`，前者利用某个初始Setting计算三个映射$(A_0)(A_3), (A_1)(A_4), (A_2)(A_5)$的映射表，后者将映射表处理为环的数量、长度的有序列表用于比对。我也设置了测试函数来验证Plugboard不改变环的数量和长度

```
def test_poland_ring():
    setting1 = Setting([2, 1, 0], [to_num("V"), to_num("D"), to_num("A")], 0, [])
    setting2 = Setting([2, 1, 0], [to_num("V"), to_num("D"), to_num("A")], 0, [(2, 3),
                                            (10, 6), (5, 7), (1, 4), (13, 14), (18, 25)])
    assert poland_ring(poland_mapping(setting1)) == poland_ring(poland_mapping(setting2))
    print("Test poland ring ok.")
```

破译函数为`poland_decrypto(mappings)`，其接受三个映射的映射表（假设已经通过足够多的密文得到）。首先忽略Plugboard求得Setting，方法就是简单的遍历并比对ring

```
    valid_settings = []
    ring = poland_ring(mappings)
    for rotor_types in itertools.permutations(range(rotor_type_num), 3):
        print("Trying rotors ", rotor_types)
```

```
        for reflector_type in range(reflector_type_num):
            for i in range(N):
                for j in range(N):
                    for k in range(N):
                        setting = Setting(list(rotor_types), [i, j, k], reflector_type, [])
                        if poland_ring(poland_mapping(setting)) == ring:
                            valid_settings.append(setting)
```

再对每个可能的Setting确定Plugboard。课件里说这可以通过将大量密文解密进行猜测，但我思考了一下觉得这并没有想象中的简单，比如解密后看似某两个字母被交换了，这可能是密文被交换而不是看到的两个明文字母被交换，交换的对数一多就更复杂。我原先采用的是暴力枚举所有两两交换的情况并验证正确性，这种方法在不超过4对时需要若干分钟，6对则需要接近1天。在群里讨论的启发下，我明白了搜索的过程可以大大简化。问题的本质是：已知映射f和映射$g = Swap \to f \to Swap$，求Swap。对于这种问题，其满足如下性质：若(x, y)为一个交换对，$g(x)$和$f(y)$也为一个交换对。这样，构造一个有向图，其节点为各个字母，若$z = g(x)$则有边$x \to z$。这样，对于图的每一个连通块，只需其中某一个字母对应的字母，就可以得到其中所有字母对应的字母，这时判断是否是两两交换即可。这样，进行dfs并剪枝，可以在数秒内得到插线板的解，且没有交换对数限制。

1.3 攻击样例

使用如下代码进行测试

```
    setting = Setting([2, 1, 0], [to_num("V"), to_num("D"), to_num("A")], 0, [(to_num("A"),
                                                                                to_num("B")),
                                                                               (to_num("C"),
                                                                                to_num("Z")),
                                                                               (to_num("D"),
                                                                                to_num("K")),
                                                                               (to_num("E"),
                                                                                to_num("F")),
                                                                               (to_num("M"),
                                                                                to_num("Q")),
                                                                               (to_num("X"),
                                                                                to_num("J"))])
    poland_decrypto(poland_mapping(setting))
```

破译函数的输入为：

```
Input mappings:
A0A3: A->G B->S C->D D->K E->I F->A G->Y H->R I->C J->N K->L L->O M->B
      N->Z O->U P->M Q->Q R->T S->W T->P U->F V->J W->V X->X Y->E Z->H
A1A4: A->G B->S C->I D->B E->E F->U G->C H->R I->A J->H K->K L->Y M->Q
```

```
     N->J O->V P->F Q->W R->N S->D T->T U->X V->L W->M X->P Y->O Z->Z
A2A5: A->N B->Q C->A D->V E->R F->U G->P H->T I->G J->C K->O L->Y M->Z
     N->E O->M P->H Q->J R->S S->B T->W U->D V->F W->X X->L Y->I Z->K
```

不考虑Plugboard进行环长度、数量匹配，得到可能的解有：

```
rotors: 1, 3, 2 reflector: 2 initial setting: T, L, R
rotors: 1, 3, 5 reflector: 2 initial setting: F, U, V
rotors: 2, 1, 3 reflector: 2 initial setting: E, W, M
rotors: 2, 1, 4 reflector: 1 initial setting: Y, B, O
rotors: 2, 3, 5 reflector: 2 initial setting: A, T, V
rotors: 2, 5, 4 reflector: 1 initial setting: A, P, W
rotors: 3, 2, 1 reflector: 1 initial setting: V, D, A
rotors: 3, 2, 1 reflector: 2 initial setting: V, B, R
rotors: 3, 4, 1 reflector: 2 initial setting: M, V, K
rotors: 3, 5, 2 reflector: 1 initial setting: F, S, M
rotors: 4, 2, 5 reflector: 2 initial setting: T, F, G
rotors: 4, 5, 2 reflector: 2 initial setting: W, D, F
rotors: 5, 1, 4 reflector: 1 initial setting: X, U, X
rotors: 5, 2, 3 reflector: 1 initial setting: U, H, D
rotors: 5, 3, 4 reflector: 1 initial setting: R, Q, A
rotors: 5, 4, 3 reflector: 1 initial setting: C, F, Y
```

对插线板进行搜索，最终存在可行的两两交换方式的结果为

```
Found 1 possible settings:
rotors: 3, 2, 1 reflector: 1 initial setting: V, D, A plugboard: A<->B J<->X M<->Q
                                                                D<->K E<->F C<->Z
```

2 Enigma破译（图灵法）

2.1 算法原理

除波兰法外，我还尝试了图灵法，思路与课件上一致。首先使用dfs找明密文中的环，我设置了环长度上限5，环数量上限30。之后，对于每种转子设置，尝试26种猜测值，验证是否满足所有环。最后，对可能的转子设置，枚举插线板。波兰法中，每个连通块一定是一个环，而图灵法则取决于具体的明文和密文，但总体思路不变。在每个连通块内，使用bfs进行flood fill，不同连通块使用dfs遍历。

2.2 攻击样例

破译函数为`Turing_decrypto(plaintext, ciphertext)`，其接受一个明密文对并输出破译过

程。

使用如下代码进行测试

```
setting = Setting([2, 1, 0], [to_num("V"), to_num("D"), to_num("A")], 0, [(to_num("A"),
                                                                            to_num("B")),
                                                                           (to_num("C"),
                                                                            to_num("Z")),
                                                                           (to_num("D"),
                                                                            to_num("K")),
                                                                           (to_num("E"),
                                                                            to_num("F")),
                                                                           (to_num("M"),
                                                                            to_num("Q")),
                                                                           (to_num("X"),
                                                                            to_num("J"))])
plaintext = "CRYPOTOGRAPYISAVERYGOODCOURSEINTSINGHUAUNIVERSITYANDILOVEITVERYMUCH"
Turing_decrypto(plaintext, crypto(setting, plaintext))
```

破译算法的输入为

```
Plaintext:  CRYPOTOGRAPYISAVERYGOODCOURSEINTSINGHUAUNIVERSITYANDILOVEITVERYMUCH
Ciphertext: XPPCFLMIUPOGAIDXJTLHTYKUSOBTQDPSXPCHZJCDRAKYBEPDKPZQTBDRFGNZUKWRHIZ
```

不考虑Plugboard，得到可能的转子类型、反射器类型、转子设置有（由于共120种转子类型、反射器类型，每种类型有26*26*26种转子设置，每种转子设置需要至少枚举26个猜测值，这个过程需要若干小时）：

```
rotors: 3, 1, 5 reflector: 1 initial setting: Q, K, Q
rotors: 3, 2, 1 reflector: 1 initial setting: V, D, A
rotors: 3, 2, 1 reflector: 1 initial setting: V, E, A
```

对插线板进行搜索，最终存在可行的两两交换方式的结果为

```
Found 1 possible settings:
rotors: 3, 2, 1 reflector: 1 initial setting: V, D, A plugboard: A<->B J<->X M<->Q
                                                                D<->K E<->F C<->Z
```

3 书面作业

3.1 1.5

运行代码

```
s = "BEEAKFYDJXUQYHYJIQRYHTYJIQFBQDUYJIIKFUHCQD"
for i in range(26):
    for j in range(len(s)):
        print(chr((ord(s[j]) - ord("A") + i) % 26 + ord("A")), end="")
    print("")
```

输出：

```
BEEAKFYDJXUQYHYJIQRYHTYJIQFBQDUYJIIKFUHCQD
CFFBLGZEKYVRZIZKJRSZIUZKJRGCREVZKJJLGVIDRE
DGGCMHAFLZWSAJALKSTAJVALKSHDSFWALKKMHWJESF
EHHDNIBGMAXTBKBMLTUBKWBMLTIETGXBMLLNIXKFTG
FIIEOJCHNBYUCLCNMUVCLXCNMUJFUHYCNMMOJYLGUH
GJJFPKDIOCZVDMDONVWDMYDONVKGVIZDONNPKZMHVI
HKKGQLEJPDAWENEPOWXENZEPOWLHWJAEPOOQLANIWJ
ILLHRMFKQEBXFOFQPXYFOAFQPXMIXKBFQPPRMBOJXK
JMMISNGLRFCYGPGRQYZGPBGRQYNJYLCGRQQSNCPKYL
KNNJTOHMSGDZHQHSRZAHQCHSRZOKZMDHSRRTODQLZM
LOOKUPINTHEAIRITSABIRDITSAPLANEITSSUPERMAN
MPPLVQJOUIFBJSJUTBCJSEJUTBQMBOFJUTTVQFSNBO
NQQMWRKPVJGCKTKVUCDKTFKVUCRNCPGKVUUWRGTOCP
ORRNXSLQWKHDLULWVDELUGLWVDSODQHLWVVXSHUPDQ
PSSOYTMRXLIEMVMXWEFMVHMXWETPERIMXWWYTIVQER
QTTPZUNSYMJFNWNYXFGNWINYXFUQFSJNYXXZUJWRFS
RUUQAVOTZNKGOXOZYGHOXJOZYGVRGTKOZYYAVKXSGT
SVVRBWPUAOLHPYPAZHIPYKPAZHWSHULPAZZBWLYTHU
TWWSCXQVBPMIQZQBAIJQZLQBAIXTIVMQBAACXMZUIV
UXXTDYRWCQNJRARCBJKRAMRCBJYUJWNRCBBDYNAVJW
VYYUEZSXDROKSBSDCKLSBNSDCKZVKXOSDCCEZOBWKX
WZZVFATYESPLTCTEDLMTCOTEDLAWLYPTEDDFAPCXLY
XAAWGBUZFTQMUDUFEMNUDPUFEMBXMZQUFEEGBQDYMZ
YBBXHCVAGURNVEVGFNOVEQVGFNCYNARVGFFHCREZNA
ZCCYIDWBHVSOWFWHGOPWFRWHGODZOBSWHGGIDSFAOB
ADDZJEXCIWTPXGXIHPQXGSXIHPEAPCTXIHHJETGBPC
```

应该是“LOOKUPINTHEAIRITSABIRDITSAPLANEITSSUPERMAN”这一行，即look up in the air | it’s a bird | it’s a plane | it’s superman

3.2 1.16 a b

运行代码

```
pi = [4, 1, 6, 2, 7, 3, 8, 5]
```

```
pi_inv = [0 for _ in range(len(pi))]
for i in range(len(pi)):
    pi_inv[pi[i] - 1] = i + 1
print(pi_inv)
s = "ETEGENLMDNTNEOORDAHATECOESAHLRMI"
for i in range(len(s) // len(pi)):
    for j in range(len(pi)):
        print(s[i * len(pi) + pi[j] - 1], end="")
```

得到逆置换为

x	1	2	3	4	5	6	7	8
$\pi^{-1}(x)$	2	4	6	1	8	3	5	7

解密结果为“GENTLEMENDONOTREADEACHOTHERSMAIL”，也就是gentlemen do not read each other's mail

3.3 1.21 a

对单字、双字词频进行统计

```
from collections import defaultdict

s = "EMGLOSUDCGDNCUSWYSFHNSFCYKDPUMLWGYICOXYSIPJCKQPKUGKMGOLICGINCGACKSNISACYKZSCKXECJCKSHYSXCGC
one_dic, two_dic = defaultdict(lambda: 0), defaultdict(lambda: 0)
one_count, two_count = 0, 0
for i in range(len(s)):
    one_dic[s[i]] += 1
    one_count += 1
for i in range(len(s) - 1):
    two_dic[s[i : i + 2]] += 1
    two_count += 1
one_list, two_list = [], []
for k, v in one_dic.items():
    one_list.append((v / one_count, k))
for k, v in two_dic.items():
    two_list.append((v / two_count, k))
one_list.sort(reverse=True)
two_list.sort(reverse=True)
print(one_list[:10])
print(two_list[:10])
```

结果为

```
[(0.145, 'C'), (0.094, 'G'), (0.078, 'S'), (0.07, 'K'), (0.059, 'Y'), (0.059, 'I'),
                            (0.055, 'U'), (0.051, 'Z'), (0.051, 'N'), (0.047, 'E')]
[(0.027, 'ZC'), (0.027, 'CG'), (0.02, 'YS'), (0.02, 'NC'), (0.02, 'GO'), (0.02, 'CN'),
                             (0.02, 'CK'), (0.02, 'AC'), (0.016, 'SF'), (0.016, 'GY')]
```

单字中C出现频率最高，且高出第二很多，猜测C->E。G第二高，猜测G->A或G->T。若G->T，双字里CG对应ET，双字频率只有0.19%，而EA有0.47%，因此大概率G->A。这样ZC->?E，频率前5里只有HE，因此猜测Z->H。利用给出的F->W，此时字符串变为

```
EMaLOSUDeaDNeUSWYSwHNSweYKDPUMLWaYIeOXYSIPJeKQPKUaKMaOLIeaINeaAeKSNISAeYKhSeKXEeJeKSHY
SXeaOIDPKheNKSHIeaIWYaKKaKaOLDSILKaOIUSIaLEDSPWhUawheeNDaYYSwUSheNXEOJNeaYEOWEUPXEhaAe
aNwaLKNSAeIaOIYeKXeJUeIUhewheeNDaYYSwEUEKUheSOewheeNeIAehEJNeSHwhEJhEaMXeYHeJUMaKUeY
```

可以看到里面出现了“wheeNDa”和“wheeNe”，大概率“wheeN”是个单词，几乎只能是“wheel”，于是N->L。由于he已经知道了，还可以寻找“the”，出现的有“Khe”、“She”、“Uhe”，再观察发现有“UhewheeN”和“UheSOewheeN”，大概率“Uhe”就是“wheel”前的定冠词，这样U->T。这样可以顺便猜一下“the SOe wheel”，三个字母且以e结尾的常用单词有：age bee die due eye fee ice lie one owe pie she see sue the tie toe use，这里只能放one，于是S->O，O->N。现在字符串变为

```
EMaLnotDeaDletoWYowHloweYKDPtMLWaYIenXYoIPJeKQPKtaKManLIeaIleaAeKolIoAeYKhoeKXEeJeKoHY
oXeanIDPKhelKoHIeaIWYaKKaKanLDoILKanItoIaLEDoPWhtawheelDaYYowtohelXEnJleaYEnWEtPXEhaAe
alwaLKloAeIanIYeKXeJteIthewheelDaYYowEtEKtheonewheeleIAehEJleoHwhEJhEaMXeYHeJtMaKteY
```

注意到前7个字母“EMaLnot”，not已经显露了出来，前面一定是主语+动词，动词最多3个字母，可以是is not/are not/do not/can not/may not/did not/had not/am not/was not，倒数第二个一定是a，那就剩can not/may not/had not/am not/was not。am not前只能是I，但前面有2个字母，不符，因此只剩can not/may not/had not/was not，这样主语是1个字母，必定是I，E->I。再注意到，动词的第一和第三个字母不能含已经破译的e a h w l t o n，直接就剩一个may，瞬间找到两个对应M->M、L->Y。现在变成了

```
imaynotDeaDletoWYowHloweYKDPtmyWaYIenXYoIPJeKQPKtaKmanyIeaIleaAeKolIoAeYKhoeKXieJeKoHY
oXeanIDPKhelKoHIeaIWYaKKaKanyDoIyKanItoIayiDoPWhtawheelDaYYowtohelXinJleaYinWitPXihaAe
alwayKloAeIanIYeKXeJteIthewheelDaYYowitiKtheonewheeleIAehiJleoHwhiJhiamXeYHeJtmaKteY
```

出现了“whiJh”，猜测J->C，出现了“alwayK”，猜测K->S，出现“loweY”，猜测Y->R。现在是

```
imaynotDeaDletoWrowHlowersDPtmyWarIenXroIPcesQPstasmanyIeaIleaAesolIoAershoesXiecesoHr
oXeanIDPshelsoHIeaIWrassasanyDoIysanItoIayiDoPWhtawheelDarrowtohelXinclearinWitPXihaAe
alwaysloAeIanIresXecteIthewheelDarrowitistheonewheeleIAehicleoHwhichiamXerHectmaster
```

看到“inclearinWit”，猜测W->G，这样就是in clearing it。看到“Aehicle”，猜测A->V。片段“thewheelDarrowitistheonewheelIAehicleoHwhichiamXerHectmaster”必然切割为the wheel Darrow. it is the one wheelI vehicle oH which i am XerHect master，这样oH是个介词，只能是on或者of，而n已经被破译了，这样H->F。Darrow特征比较明显，查了一下大概有barrow

farrow harrow marrow narrow yarrow，去掉已破译的字母还剩barrow farrow，前者是“两轮流动售货车”，后者是“一窝子猪”，显然D->B。现在形势已经比较明朗了

```
i may not be able to grow flowers bPt my garIen XroIPcesQPstas many IeaIleavesolIover shoes Xieces of roXeanIbPshelsofIeaIgrassasanyboIysanItoIayiboPght a wheel barrow to helX in clearing it PX i have always loveI anIresXecteI the wheel barrow it is the one wheeleI vehicle of which i am Xerfect master
```

从语法和语义容易看出bPt->but，garIen->garden，Xieces->pieces，helX->help，loveI->loved，也就是P->U，I->D，X->P。现在只剩下一对Q->J，这样就是

```
i may not be able to grow flowers, but my garden produces just as many dead leaves, old overshoes, pieces of rope and bushels of dead grass as anybody's. and today i bought a wheel barrow to help in clearing it up. i have always loved and respected the wheel barrow. it is the one wheeled vehicle of which i am perfect master.
```

3.4 1.21 b

使用重合指数法

```
import numpy as np

s = "KCCPKBGUFDPHQTYAVINRRTMVGRKDNBVFDETDGILTXRGUDDKOTFMBPVGEGLTGCKQRACQCWDNAWCRXIZAKFTLEWRPTYCQ
for m in range(1, 10):
    print("m=%-2d: " % m, end="")
    for i in range(m):
        f = np.zeros(26)
        for c in s[i::m]:
            f[ord(c) - ord("A")] += 1
        print(round((f * (f - 1)).sum() / (f.sum() * (f.sum() - 1)), 3), end=" ")
    print("")
```

输出如下：

```
m=1 : 0.041
m=2 : 0.038 0.047
m=3 : 0.056 0.048 0.048
m=4 : 0.037 0.043 0.038 0.049
m=5 : 0.043 0.043 0.033 0.035 0.043
m=6 : 0.063 0.084 0.049 0.065 0.043 0.073
m=7 : 0.031 0.044 0.043 0.041 0.044 0.044 0.041
m=8 : 0.033 0.041 0.034 0.041 0.039 0.045 0.041 0.055
m=9 : 0.051 0.043 0.064 0.075 0.041 0.035 0.044 0.048 0.042
```

明显看出密钥长度为6。之后对密钥进行猜测，我的方法是让每隔6位的组内出现频率最高的字母去匹配英文中出现频率的前6个字母，这样搜索空间缩小到了6^6。之后，设置一些常用的英文词汇，检查解密后它们出现的次数之和，设置一个阈值，得到满足条件的密钥并人工检查明文。代码如下：

```
words = ["THE", "FOR", "THIS", "THAT", "AND", "HOW", "WHO", "WHERE", "NOT"]
max_f = []
for i in range(6):
    max_f.append(chr(np.array([s[i::6].count(chr(ord("A") + x)) for x in range(26)]).argmax()
                                                                                + ord("A")))
candidate = ["E", "T", "A", "O", "I", "N"]
for i in range(6 ** 6):
    choices = []
    k = i
    for j in range(6):
        choices.append(k % 6)
        k //= 6
    char_list = []
    for j in range(len(s)):
        char_list.append(chr((ord(s[j]) - ord(max_f[j % 6]) + ord(candidate[choices[j % 6]])
                                                        - ord("A") + 26) % 26 + ord("A")))
    s_new = "".join(char_list)
    cnt = 0
    for word in words:
        cnt += s_new.count(word)
    if cnt > 15:
        password_lst = []
        for j in range(6):
            password_lst.append(chr((ord(s[j]) - ord(s_new[j]) + 26) % 26 + ord("A")))
        print("".join(password_lst), ":")
        print(s_new)
```

输出为：

```
CRYPTJ :
ILEARSEDHOWYOCALCZLATETMEAMOUSTOFPAUERNEEIEDFORFROOMWMENIWAXATSCHTOLYOURULTIPQYTHESVUA
REFTOTAGETFTHEWFLLSBYYHECUBNCCONTJNTSOFYHEFLOTRANDCJILINGHOMBINJDANDDTUBLEIYYOUTHJNALL
OBHALFTMETOTAQFOROPJNINGSXUCHASBINDOWXANDDOTRSTHESYOUALQOWTHETTHERHFLFFORRATCHISGTHEPF
TTERNYHENYOZDOUBLJTHEWHTLETHISGAGAISTOGIVJAMARGNNOFERWORANDYHENYOZORDERYHEPAPJR
CRYPTO :
ILEARNEDHOWTOCALCULATETHEAMOUNTOFPAPERNEEDEDFORAROOMWHENIWASATSCHOOLYOUMULTIPLYTHESQUA
REFOOTAGEOFTHEWALLSBYTHECUBICCONTENTSOFTHEFLOORANDCEILINGCOMBINEDANDDOUBLEITYOUTHENALL
OWHALFTHETOTALFOROPENINGSSUCHASWINDOWSANDDOORSTHENYOUALLOWTHEOTHERHALFFORMATCHINGTHEPA
```

TTERNTHENYOUDOUBLETHEWHOLETHINGAGAINTOGIVEAMARGINOFERRORANDTHENYOUORDERTHEPAPER
CRYPTP :
ILEARMEDHOWSOCALCTLATETGEAMOUMTOFPAOERNEECEDFORZROOMWGENIWARATSCHNOLYOULULTIPKYTHESPUA
REFNOTAGENFTHEWZLLSBYSHECUBHCCONTDNTSOFSHEFLONRANDCDILINGBOMBINDDANDDNUBLEISYOUTHDNALL
OVHALFTGETOTAKFOROPDNINGSRUCHASVINDOWRANDDONRSTHEMYOUALKOWTHENTHERHZLFFORLATCHIMGTHEPZ
TTERNSHENYOTDOUBLDTHEWHNLETHIMGAGAIMTOGIVDAMARGHNOFERQORANDSHENYOTORDERSHEPAPDR

第二个结果是通顺的，也就是

I learned how to calculate the amount of paper needed for a room when I was at school. You multiply the square footage of the walls by the cubic contents of the floor and ceiling combined and double it. You then allow half the total for openings such as windows and doors. Then you allow the other half for matching the pattern. Then you double the whole thing again to give a margin of error and then you order the paper.

3.5 1.21 c

统计单字频率，发现C最多，B其次，猜测C->E，B->T，解得映射为$f(c) = 11c + 8 \pmod{26}$，
于是得到明文

ocanadaterredenosaieuxtonfrontestceintdefleuronsglorieuxcartonbrassaitporterlepeeilsaitport
erlacroixtonhistoireestuneepopeedesplusbrillantsexploitsettavaleurdefoitrempeeprotegeranosfo
yersetnosdroits

3.6 1.21 d

尝试后发现可能是维吉尼亚密码，使用上上节的程序跑一下：

m=1 : 0.042
m=2 : 0.044 0.046
m=3 : 0.045 0.048 0.048
m=4 : 0.043 0.058 0.047 0.047
m=5 : 0.045 0.043 0.041 0.044 0.037
m=6 : 0.051 0.063 0.055 0.07 0.056 0.07
m=7 : 0.041 0.045 0.044 0.038 0.046 0.032 0.046
m=8 : 0.058 0.059 0.051 0.048 0.038 0.065 0.046 0.05
m=9 : 0.049 0.045 0.042 0.036 0.04 0.04 0.043 0.045 0.062

同样猜测密钥长度为6。将阈值从15减小到10，得到了唯一的输出

THEORY :
IGREWUPAMONGSLOWTALKERSMENINPARTICULARWHODROPPEDWORDSAFEWATATIMELIKEBEANSINAHILLANDWHE
NIGOTTOMINNEAPOLISWHEREPEOPLETOOKALAKEWOBEGONCOMMATOMEANTHEENDOFASTORYICOULDNTSPEAKAWH

OLESENTENCEINCOMPANYANDWASCONSIDEREDNOTTOOBRIAHTSOIENROLLEDINASPEECHCOUQSETAUGHTBYORVILLESANDTHEFOUNDEROFREFLEXIVERELAXOLOGYASELFHYPNOTICTECHNIQUETHATENABLEDAPERSONTOSPEAKUPTOTHREEHUNDREDWORDSPERMINUTE

也就是

I grew up among slow talkers men in particular who dropped words a few at a time like beans in a hill and when I got to Minneapolis where people took a lake wobegon comma to mean the end of a story. I couldn't speak a whole sentence in company and was considered not to obriaht so I enrolled in a speech couqse taught by orvilles and thefounder of reflexive relaxology a self hypnotic technique that enabled a person tospeak up to three hundred words per minute.

3.7 1.26

代码如下

```
s = "CTAROPYGHPRY"
m = 4
n = 3
result_list = []
for i in range(len(s)):
    row, col = i // m, i % m
    result_list.append(s[col * n + row])
print("".join(result_list))
```

尝试了42的各种因子分解后发现没有想要的结果，于是意识到可能是分组密码，最终尝试得出$m = 3, n = 2$

```
s = "MYAMRARUYIQTENCTORAHROYWDSOYEOUARRGDERNOGW"
m = 3
n = 2
result_list = []
for k in range(len(s) // (m * n)):
    for i in range(m * n):
        result_list.append(s[(i % m) * n + i // m + k * m * n])
print("".join(result_list))
```

结果为MARYMARYQUITECONTRARYHOWDOESYOURGARDENGROW，即Mary, Mary, quite contrary. How does your garden grow?。