山东大学
SHANDONG UNIVERSITY

实验一：Enigma 的实现与破解

组员：刘舒畅，李昕，林宗茂

2023 年 9 月 24 日

成员信息及完成部分：

刘舒畅：202122460175 代码编写及报告校对

李昕：202100460065 原理分析,Task1/2/4 报告编写

林宗茂：202100460128 代码测试，Task3 报告编写

摘　要

Enigma 是德国人阿瑟. 谢尔比乌斯于 20 世纪初发明的一种能够进行加密和解密操作的机器。Enigma 这个名字在德语中是”谜”的意思。谢尔比乌斯使用能够转动的圆盘和电路，创造出人类手工无法实现的高强度密码。在刚刚发明之际，Enigma 被用于商业领域，后来到了纳粹时期，德国国防军采用了 Enigma，并将其改良后用于军事用途。

在本次实验中,我们分析了 Enigma 密码机的结构与原理,完成了 Enigma 机的加密代码，同时分析了 Enigma 机破解原理，尝试利用环路 Ciber 进行破解，并与穷举攻击进行比较。在文章最后，我们讨论了 Enigma 的唯密文攻击以及改进 Enigma 算法的一些尝试。

关键词: Enigma　密码分析

目录

1 问题重述

问题一 给定明文 wetter，按照参考文档给定参数，编程实现 Enigma 机的加密，并输出对应的密文（a 从 0 记）。

问题二 给定明密文数据流，在问题一中的两个转子和反射器的定义不变的情况下，尝试恢复密钥。(结果可能不唯一也可能无解)。

问题三 如果直接用穷举攻击猜测全部密钥，请理论分析需要猜测多少密钥？假设在个人电脑上每秒可进行 10^6 次加密运算，则仅用一台个人电脑大约多久可恢复出候选密钥的集合？对题目 1）中实现的加密算法进行 1000 次加密，统计每秒可进行的加密次数，判断在个人电脑上直接进行穷举攻击是否可行？若可行，请测试并与题目 2）中求解出的密钥集合进行比较。

问题四 是否有其他密钥恢复的方法？对改进 Enigma 的安全性有何建议？（文字描述即可，不用实现）

2 符号说明

符号	解释
$\oplus$	异或
$\rightarrow$	加密方向

3 实验准备

3.1 Enigma 机的加密过程 [1]

Enigma 密码机的构造共由四个主要部件组成。分别是：1. 包含 26 个英文字母的键盘：输入端，每次输入一个英文字母。2. 线路接线板：将 l 对字母相连接，进行加密工作。3. 扰频组合：由快速扰频器、中速扰频器、慢速扰频器组成，进行加密工作。4. 反射器：保证加密后得到的字母与输入的字母一定不相同。

Enigma 密码机加密是一种多表代换的对称加密算法，逐字母加密。设明文空间为 P，密文空间为 C，密钥 K1,K2，表示接线板 S 和两个个转子 Q1/Q2 的初始位置。记反射器为 T，则 $C = Enc_{K1,K2}(P), P = Enc_{K1,K2}(C)$

扰频器, 也称作转子。单个转子的加密是一种单表置换。转子的左右两侧各有 26 个点，分别代表 A-Z 这 26 个英文字母。一端输入一端输出。为了达到加密的目的，将左右两侧的字母交叉连接，为了提高加密的安全性，每输入一个字母后，第一个转子就会自动转动一格。当第一个转子转动一圈后，第二个转子就转动一格。这样原来的单表置换经过旋转变为多表置换，使得安全性更高。对于只有 2 个转子的 Enigma 机, 密钥空间只有 $2 * 26^3 = 35152$.

反射器的加密是一种固定的单表置换, 加密方式是将最后一个转子的其中两个触点连接起来，并将电流沿一个不同的路倒回到线路接线板。这样的反射器导致加密后的字母与输入字母一定不相同，并且使得 Enigma 密码机的加密过程是自反的，这也是加解密一致的关键部件。同时还让 Enigma 机有了另一个性质，即：一个字母不会加密成自身。这也是破解的关键。

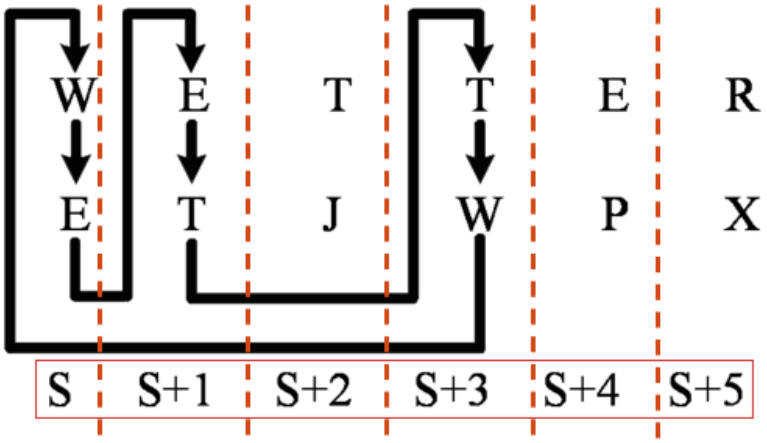
线路接线板为简单的字母置换，设使用了 l 条接线板，则置换表可能的总数为 $\frac{26!}{(26-2l)!*l!*2^l}$

3.2 Enigma 机的破解过程 [1]

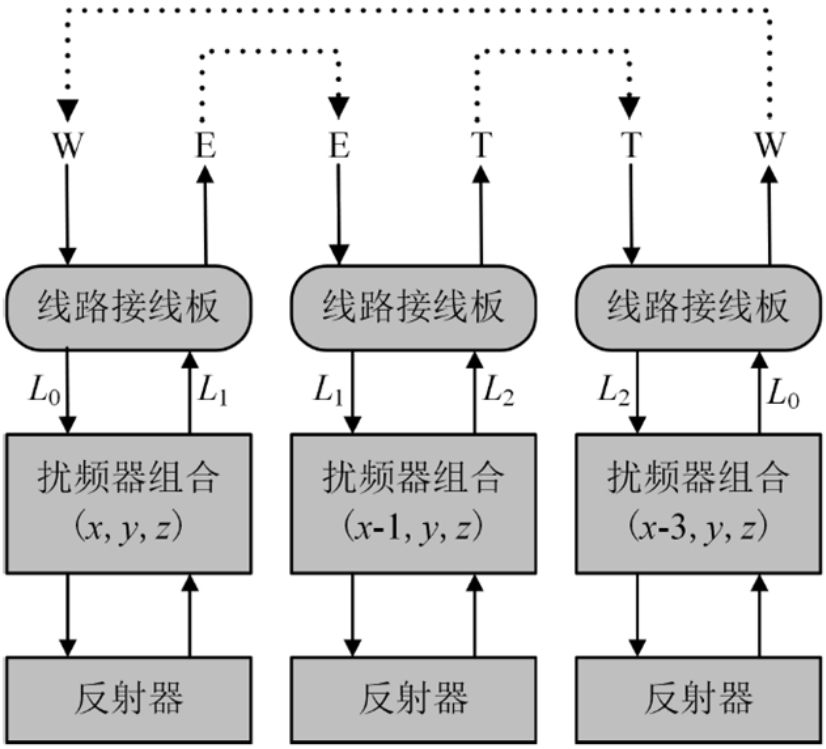
3.2.1 通过环路屏蔽接线板，实现分割

因为 Enigma 反射器和转子的设计，所以在 Enigma 机加密过程中，一个字母是不会被加密成自身的，根据 Enigma 的这个性质，在获知密文的位置之后，我们便可以将明文与密文联系起来，这种明密文组合被称作克利巴（Crib）。

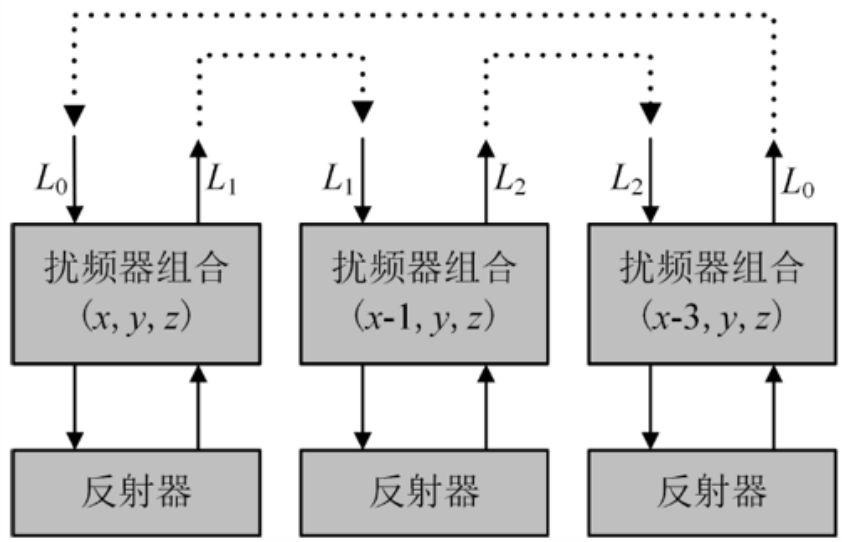
在加密时，特殊的 Crib 内部能形成一个环路，如下图例子：



以上图为例，分析其加密过程如下：



我们注意到，出现了不随机的现象（不随机现象：L0 进 L0 出），此时可以消除扰频器的影响，将 Li 直接相连，如下图：



由于本题中，转子的排列已经确定，故不需要额外遍历所有的转子组合，此时只需穷举扰频器的起点，判断 $L0$ 和 $L0'$ 是否相等，如果相等 (即环路亮灯)，说明此时找到了一个可能起点 K2，把所有的可能秘钥都记录下来, 即可形成一个较小的可能秘钥空间。遍历所有存在的 Cibe 环，就可以进一步缩小密钥 K2 空间。

3.2.2 还原接线板

对于恢复线路接线板的设置，主要依据判定条件” 每个接线只能链接两个字母，即不可能出现 A 链接 B，B 链接 C”，可与根据此判定来排除矛盾的情况。

具体来说，当进行 3.2.1 时，某条回路验证成功，此时得到了输入字母与 Li 之间的对应关系，确定了部分接线板的可能值，此时依据上述判定条件存在矛盾，则说明 K2 猜测错误，需要返回猜测扰频器的步骤，利用下一条” 亮灯” 的回路继续重复此过程。

当测试完环路明密文以后，继续挖掘该 Crib 中除环路以外的其他明密文的信息，即如果该 Crib 中其他明密文对在之前没有确定与其相关的连接情况，则进行猜测（穷举），遍历所有与 k2 所得结果不矛盾的接线情况，得到可能的接线板设置，结合多条 Crib，排除矛盾，得到最终的接线板接线情况。

4 实验过程

4.1 实现 Enigma 机的加密

Enigma 机的加密代码需要完成接线板，扰频器，反射器三个主要部件。

对于接线板，其实现较为简单，利用实验文档提供的接线板数组编写字节替换如下：

```
1
2 S = [2, 1, 0, 4, 3, 5, 7, 6, 8, 9, 24, 11, 20, 13, 14, 15, 16, 17,18, 19, 12, 21, 22, 23, 10, 25]
3 ...
4  tmp = S[tmp]    #字节替换
```

对于扰频器，首先定义两个替换数组 R0-raw,R1-raw，起点变量 k，同时定义一个转子转动函数 Rotate，用以推动起点变量 k，代码实现如下：

```
1 R0_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16, 21, 19,
      4]
2 R1_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16, 15,
      23]
3 ...
4 def Rotate(k):
5     k[0] -= 1
6     if(k[0]==-1):
7         k[1] -= 1
```

```
        k[0] = 25
        if(k[1]==-1):
            k[1] = 25
...
tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
```

对于反射器，由 3.1 部分可知，为一种固定的单表置换，只需要利用预先设置的反射器数组即可完成置换：

```
T = [10, 20, 14, 8, 25, 15, 16, 21, 3, 18, 0, 23, 13, 12, 2, 5, 6, 19, 9, 17, 1, 7, 24, 11, 22, 4]
...
tmp = T[tmp]
```

综合上述分析和已经给出的代码，给出加密过程中的主要代码流程（完整代码见附录 A5），即输入的单字母依次经过 1. 线路接线板（Stecker），2. 扰频器组合（转子，Rotors），3. 反射器（Reflector，T），4. 逆向扰频器组合，5. 逆向线路接线板：

```
for i in m:
    tmp = ord(i)-ord('a')
    tmp = S[tmp]      #1
    tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26    #2
    tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
    tmp = T[tmp]      #3
    tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26    #4
    tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
    tmp = S[tmp]  #5
    c = c + chr(tmp+ord('a'))
    Rotate(k)   #转动
```

运行加密代码，得到加密后的密文为“illgrv”。

4.2 尝试恢复密钥

按照 Part3 部分的分析，我们尝试构建破解代码，首先，我们要找到“每一个字母是不会被加密成自身”的明密文对应，即 Crib：

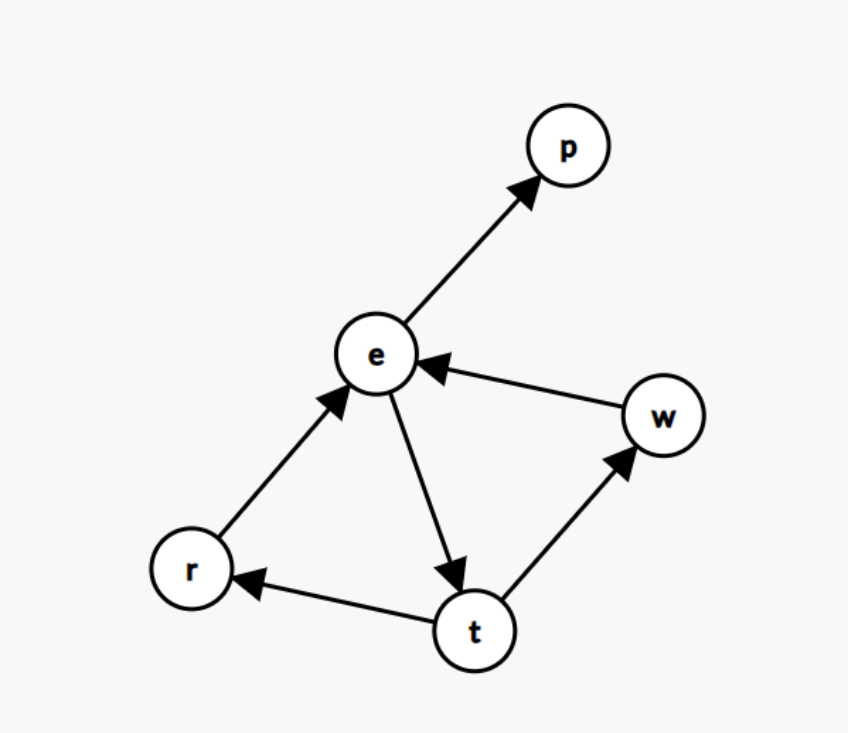
```
for i in range(len(c)-len(m)+1):
    tmp = c[i:i+len(m)]
    flag = 0
    for j in range(len(m)):
        if m[j] == tmp[j]:
            flag = 1
            break
    if flag == 1:
        continue
    print(tmp)
    res = tmp
```

接下来，在每个 Crib 中寻找环路。整体通过深度优先搜索（dfs）实现，尝试在字符串 m 和 c 中找到一条路径，使得从 m 字符串中的某个字符出发，通过与 c 字符串中的字符进行匹配，每次匹配后，将已使用过的字符标记为已使用，在递归返回后再重置标记为未使用，以便继续搜索其他可能的路径：

```
flag = [0]*len(m)
def dfs(s,u):
    # print(u)
    if m[s] == u:
        return 1
    for i in range(len(m)):
        if(flag[i] == 1):
            continue
        if m[i] == u:
            flag[i] = 1
            if dfs(s,c[i]) == 1:
                print((i,m[i],c[i]))
                flag[i] = 0
                return 1
            flag[i] = 0
```

```
    return 0
for i in range(len(m)):
    flag[i] = 1
    if dfs(i,c[i]) == 1:
        print((i,m[i],c[i]))
        print()
    flag[i] = 0
```

运行之后可以得到两条环路如下图所示：



找到了 Crib 和环路，接下来根据 3.2.1 的分析穷举 K2，利用 4.1 中编写的 Enigma 加密 Crypt(k,m) 构造解密环路 (去掉接线板 S)，从而找到同时满足两条环路 Crib 且没有接线冲突的 K2。在最外层循环，遍历所有可能的密钥组合 [i, j]，在内层循环，遍历明文 m 以模拟硬件上的 26 条接线。

针对找得到两条环路先后进行以下操作，首先计算出环路路径中五条路径的偏移量 ki(i=0,1,2,3,5)，使 m1 为第一条环路输入 L，m2 为第二条路径输入 L'，构造两条环路过程，c1 为第一条环路输出，c2 为第二条环路输出：

```
m1 = m
c1 = Crypt(k3,Crypt(k1,Crypt(k0,m1)))
```

```
m2 = Crypt(k0,m1)
c2 = Crypt(k5,Crypt(k2,Crypt(k1,m2)))
```

此时得到判断 c1 == m1 和 c2 == m2 是否同时成立，若成立，说明找到一组可能的 K2 解，接下来判断此时得到的接线板对应关系是否成立 (是否存在矛盾)：

```
if c1 == m1 and c2 == m2:
    m0 = m
    m1 = Crypt(k0,m0)
    m3 = Crypt(k1,m1)
    m2 = m3
    m5 = Crypt(k2,m2)
    c0 = Crypt(k0,m0)
    c1 = Crypt(k1,m1)
    c2 = Crypt(k2,m2)
    c3 = Crypt(k3,m3)
    c5 = Crypt(k5,m5)
    if check([m0,m1,m2,m3,m5],[c0,c1,c2,c3,c5],[0,1,2,3,5]) == 0:
        print()
        continue
    print()
    S = check([m0,m1,m2,m3,m5],[c0,c1,c2,c3,c5],[0,1,2,3,5])
    for l in range(26):
        if S[l] == -1:
        S[l] = l
    print(k)
    ans.append([k,S])
    print('ans:',ans)
    break
```

对于判断是否矛盾的 check() 函数，通过将所有已知关系进行整合，查看是否存在矛盾；若没有矛盾，返回 1：

```
def check(l1,l0,l2):
    l = [-1] * 26
    # print(l)
```

```
    for i in range(len(l1)):
        print(l1[i],ord(m_raw[l2[i]])-ord('a'),end=' ')
        if (l[l1[i]] != -1) and (l[l1[i]] != ord(m_raw[l2[i]])-ord('a')):
            print()
            return 0
        if (l[ord(m_raw[l2[i]])-ord('a')] != -1) and (l1[i] != l[ord(m_raw[l2[i]])-ord('a')]):
            print()
            return 0
        l[ord(m_raw[l2[i]])-ord('a')] = l1[i]
        l[l1[i]] = ord(m_raw[l2[i]])-ord('a')
        print(l)

    for i in range(len(l0)):
        print(l0[i],ord(c_raw[l2[i]])-ord('a'),end=' ')
        if (l[l0[i]] != -1) and (l[l0[i]] != ord(c_raw[l2[i]])-ord('a')):
            print()
            return 0
        if (l[ord(c_raw[l2[i]])-ord('a')] != -1) and (l0[i] != l[ord(c_raw[l2[i]])-ord('a')]):
            print()
            return 0
        l[ord(c_raw[l2[i]])-ord('a')] = l0[i]
        l[l0[i]] = ord(c_raw[l2[i]])-ord('a')
        print(l)
    return l
```

通过上述步骤，可以还原出多个可能正确的扰频器和接线板设置，记为 ans。但在实际加密后发现，最终结果的 C4 与原密文不同，这就要求我们将 P 与现 C4 的字符接线，同时保证接线不会造成原有明密文对的变动。通过这一要求，我们又可以筛掉一部分候选密钥。

```
for i in m:
    tmp = ord(i)-ord('a')
    tmp = S[tmp]
    tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
    tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
    tmp = T[tmp]
    tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26
```

```
        tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
        tmp = S[tmp]
        c = c + chr(tmp+ord('a'))
        Rotate(k)
    # print(c)
    # print(ord(c[4])-ord('a'),S[ord(c[4])-ord('a')],ord('p')-ord('a'),S[ord('p')-ord('a')])
    if ord(c[4])-ord('a') == S[ord(c[4])-ord('a')] and ord('p')-ord('a') == S[ord('p')-ord('a')]:
        print(S,k,c,ord(c[4])-ord('a'),S[ord(c[4])-ord('a')],ord('p')-ord('a'),S[ord('p')-ord('a')])
```

即利用已经求出的 K2 和 S 对明文'wetter' 进行加密，如果加密出的密文 c 第五位与密文 c 第五位对应的接线板相同，且字母 p 与 p 对应的接线板相同，如下图结果：

```
----------
[0, 1, 2, 19, 21, 5, 22, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 18, 3, 17, 4, 6, 23, 24, 25] etrwme 12 12 15 15
[0, 1, 2, 3, 6, 5, 4, 7, 17, 9, 10, 11, 12, 13, 14, 15, 16, 8, 18, 25, 20, 21, 22, 23, 24, 19] etrwce 2 2 15 15
[0, 19, 2, 3, 25, 5, 22, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 18, 1, 17, 21, 6, 23, 24, 4] etrwve 21 21 15 15
[0, 1, 17, 3, 24, 5, 6, 7, 8, 9, 10, 11, 12, 22, 14, 15, 16, 2, 18, 20, 19, 21, 13, 23, 4, 25] etrwfe 5 5 15 15
[0, 1, 2, 3, 24, 5, 6, 17, 8, 9, 10, 11, 12, 13, 14, 15, 16, 7, 18, 19, 20, 21, 25, 23, 4, 22] etrwje 9 9 15 15
[0, 1, 2, 3, 24, 22, 6, 7, 8, 17, 10, 11, 12, 13, 14, 15, 16, 9, 18, 25, 20, 21, 5, 23, 4, 19] etrwhe 7 7 15 15
[0, 1, 2, 17, 11, 5, 6, 22, 8, 19, 10, 4, 12, 13, 14, 15, 16, 3, 18, 9, 20, 21, 7, 23, 24, 25] etrwke 10 10 15 15
>
```

此时说明，可以与 p 相连导出正确结果的字母没有与第三个字母相连，且 p 连接自己本身，此时若将 p 与该字母相连，如上图第一个例子，将 12 与 15（即 m 与 p）相连，可以导出一组答案，遍历所有可能，得到下图 7 个可能的结果：

```
========== RESTART: C:\Users\waldeinsamkeit\Downloads\task2_2.py ==========
[0, 1, 2, 19, 21, 5, 22, 7, 8, 9, 10, 11, 15, 13, 14, 12, 16, 20, 18, 3, 17, 4, 6, 23, 24, 25] [2, 3] etrwpe
[0, 1, 15, 3, 6, 5, 4, 7, 17, 9, 10, 11, 12, 13, 14, 2, 16, 8, 18, 25, 20, 21, 22, 23, 24, 19] [5, 16] etrwpe
[0, 19, 2, 3, 25, 5, 22, 7, 8, 9, 10, 11, 12, 13, 14, 21, 16, 20, 18, 1, 17, 15, 6, 23, 24, 4] [9, 7] etrwpe
[0, 1, 17, 3, 24, 15, 6, 7, 8, 9, 10, 11, 12, 22, 14, 5, 16, 2, 18, 20, 19, 21, 13, 23, 4, 25] [12, 16] etrwpe
[0, 1, 2, 3, 24, 5, 6, 17, 8, 15, 10, 11, 12, 13, 14, 9, 16, 7, 18, 19, 20, 21, 25, 23, 4, 22] [19, 20] etrwpe
[0, 1, 2, 3, 24, 22, 6, 15, 8, 17, 10, 11, 12, 13, 14, 7, 16, 9, 18, 25, 20, 21, 5, 23, 4, 19] [20, 20] etrwpe
[0, 1, 2, 17, 11, 5, 6, 22, 8, 19, 15, 4, 12, 13, 14, 10, 16, 3, 18, 9, 20, 21, 7, 23, 24, 25] [25, 1] etrwpe
```

以上即为破解得到的所有可能密钥 (K2,S)。

4.3 穷举攻击

直接用穷举攻击猜测全部密钥，对于接线数量未知的接线板，需要考虑接线数量 l 的所有可能性 a_l 最后求和; 对于已经确定了顺序的两个固定转子，只有 26^2 个可能。从连线数量 l=0 开始考虑，26 个字母中不选取字母，可能性为

$$a_0 = 1$$

连线数量 l=1 时，考虑在 l=0 的基础上从 26 个字母中选取两个连线，可能性为

$$a_1 = C_{26}^2 * a_0$$

连线数量 l=2 时，考虑在 l=1 时每种连线方式的基础上，从剩下 24 个字母中选取两个连线，共 $C_{26}^2 * C_{24}^2 * a_0$ 种连线方式；由于 l=2 的每种连线方式都可以由 2 种 l=1 时的连线方式再新连一根线得到，故需要除以重复数 2，可能性为

$$a_2 = \frac{C_{26}^2 * C_{24}^2 * a_0}{2}$$

依次递推，当连线数量 l=i 时，都可以由 l=i-1 时每种连线方式的基础上，再从剩下 26-2i+2 个字母中选取两个；同理，l=i 的每种连线方式可由 i 种 l=i-1 时的连线方式再连一根线得到，得可能性为

$$a_i = \frac{C_{26-2i+2}^2 * a_{i-1}}{i} = \frac{C_{26}^2 * C_{24}^2 * \cdots * C_{26-2i+2}^2 * a_0}{i!}$$

最后将 14 种情况可能性相加，得到接线板密钥数量 S。最终密钥数量为

$$S * 26^2$$

python 代码实现为：

```
from scipy.special import comb, perm
def C(n,i):
    return int(comb(n,i))
import math
tmp = 1
res = 1
for i in range(1,14):
    tmp *= C(26-2*(i-1),2)
    tmp = tmp//i
    res += tmp
res = res*26*26
print('密钥数量：',res,'改写为底为2的指数：',len(bin(res))-2,'改写为底为10的指数：',math.log
    (res/(10**6),10),'破解所需年数：',res//(31536000*10**6))
```

最终结果为: 密钥数量：360298000743589376，改写为底为 2 的指数：59，改写为底为 10 的指数：11.556661852274008，破解所需年数：11424. 由此可知，当一个人的电脑每秒能处理 10^6 次加密运算时，依靠穷举攻击破解有两个转子的恩格玛机需要 11424 年。

接下来测试我们编写的 python 加密程序，测试程序见附录 A.4，加密 1000 次所需时间约为 0.0077s，即每秒加密 $1.3*10^5$ 次，很显然通过穷举攻击恢复 Enigma 机密钥是不可行的。

4.4 其他密钥恢复的方法

4.4.1 对于 Enigma 转子设置的唯密文攻击 [2]

对于 Enigma 知道明密文对的情况下，可以寻找 Crib 来进行已知明文攻击，当只有密文时，且 S 和 K2 均未知时，尝试进行唯密文攻击。

Enigma 密码机还存在一个弱点，即插线板不是所有的字母都是存在接线的，即若存在 5 条接线，则大半字母没有经过置换，如果我们用正确的转子设置、反射器设置和转子消息密钥设置解密一条消息，但在插拔板上没有任何插头，那么输出中的某些字母将具有正确的明文值，这些没有被置换的字母的数量会根据使用的插头数量和消息的长度而有所不同。尽管不知道输出中哪些字母是正确的，但如果消息足够长，可以在消息的整体统计中看到它们对重合指数 IC 的影响。以此类推，当使用部分正确的转子密钥 K2 时，也会出现类似的效果。

$$IC = \sum_{i=A}^{Z} \frac{f_i \cdot (f_i - 1)}{n \cdot (n - 1)}.$$

攻击方法可以抽象为：假设没有接线板影响，穷举转子起始点，反解出得到的明文，并计算其重合指数 IC，并有理由怀疑 IC 值更高的转子起点更接近正确密钥 K2。

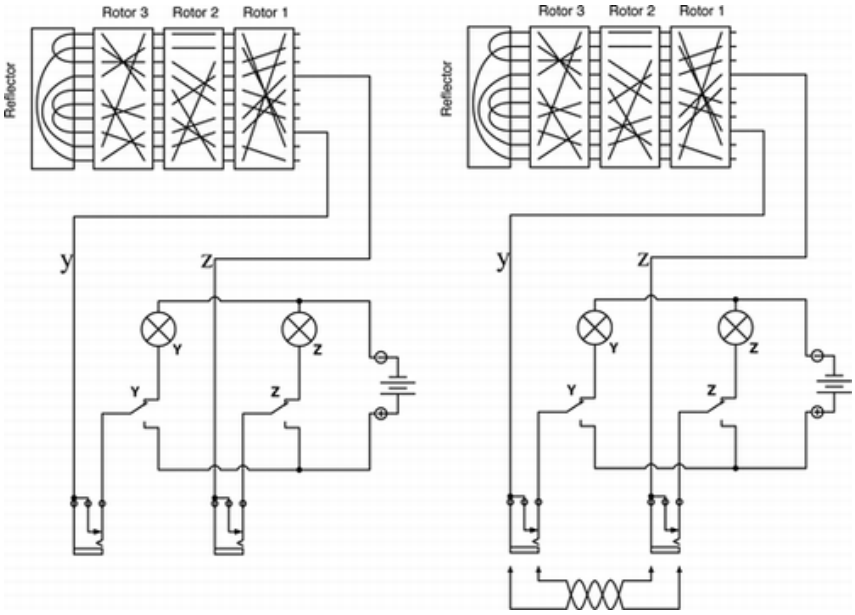
特别的，Enigma Machine 作为一个置换加密系统，其频率分析相较于普通的多表置换可谓非常困难，但若使用机器学习的方法（如遗传算法），则可大大加快这一过程[3]

4.4.2 对于 Enigma 插线板设置的唯密文攻击 [4]

当已知转子设置，求解唯密文攻击下的插线板设置时，同样借助自然语言的规律性来分析，使用 Sinkov 统计来分析，也使用 f_i 来表示 A-Z 在自然语言中的出现概率，n_i 表示在特定文本中的概率，计算 $s = \sum_{i=A}^{Z} n_i * f_i$(s 越高，越可能为自然语言)，在知道转子起点的情况下，穷举所有单插线板设置，统计其 s 取值大小，将最大值记录为当前状态，在此基础上，重复该操作，直到得到的 s 中没有突出值。

4.5 改进 Enigma 的方法

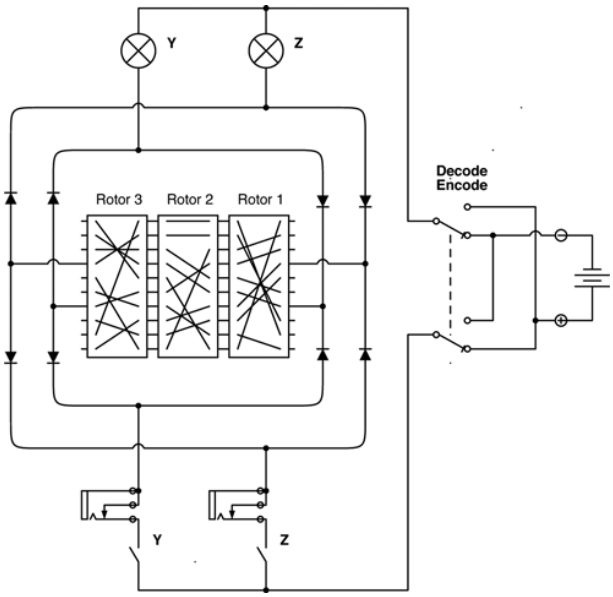
4.5.1 破坏 Enigma 可逆结构和自我编码特点 [5]



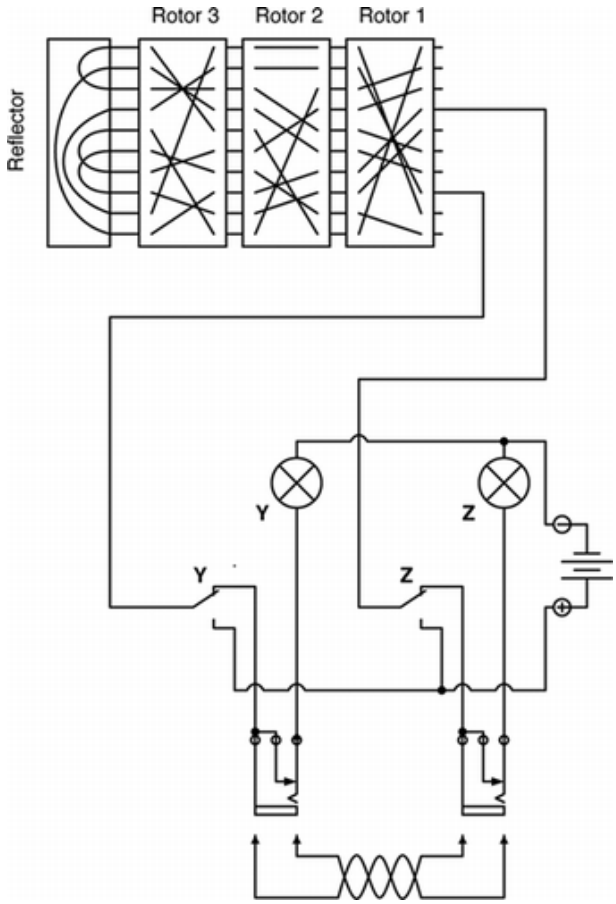
由上图分析，Enigma 提供了一个加密解密同构的可逆结构，以便可以使用与明文编码相同的密钥解码密文，如果 A 被编码为 B，那么（在相同的 Enigma 配置中）B 将被相互（解）编码为 A。如果破坏这个结构，则可以很好地破坏自动炸弹解密 (即 3.2 论述的利用 Crib 通过环路屏蔽接线板，实现分割的方法)。

同时，由图可知，Enigma 还有一个自我编码的弱点，即没有字母可以编码为它自己，这个漏洞也被用于炸弹自动解密中。

对于可逆结构，可以认识到这是为了方便使用同一台机器进行加解密，此处问题可以通过设计开关来进行规避，即加密解密利用开关与二极管 (或其他更复杂单向电元器件) 来设计两条电路，破坏加密解密可逆结构，如下图所示（只显示 z,y 两键）:



对于自编码弱点，可以改变插线板的接线位置，即只交换输出（显示灯板），而不是输入输出同时交换 (将接线板与显示灯板和键盘连接)，如下图（只显示 z,y 两键）:



值得注意的是，我们小组认为上述改进对 4.4 提到的唯密文攻击理论上安全性没有很大影响，如果想避免频率攻击，需要在 Enigma 中引入混淆和扩散。

参考文献

[1] 王美琴等. 密码分析学 [M]. 2023.

[2] James J. Gillogly (1995) CIPHERTEXT-ONLY CRYPTANALYSIS OF ENIGMA,CRYPTOLOGIA, 19:4, 405-413, DOI: 10.1080/0161-119591884060

[3] Åvald Åslaugson Sommervoll & Leif Nilsen (2021) Genetic algorithm attack on Enigma' s plugboard, Cryptologia, 45:3, 194-226, DOI: 10.1080/01611194.2020.1721617

[4] Smart N P. Cryptography Made Simple[M]. Springer International Publishing Switzerland, 2016.

[5] Harold Thimbleby (2016) Human factors and missed solutions to Enigma design weaknesses, Cryptologia, 40:2, 177-202, DOI: 10.1080/01611194.2015.1028680.

A 附录

A.1 Enigma 加密源代码

```
S = [2, 1, 0, 4, 3, 5, 7, 6, 8, 9, 24, 11, 20, 13, 14, 15, 16, 17,18, 19, 12, 21, 22, 23, 10, 25]
R0_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16, 21, 19, 4]
R1_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16, 15, 23]
T = [10, 20, 14, 8, 25, 15, 16, 21, 3, 18, 0, 23, 13, 12, 2, 5, 6, 19, 9, 17, 1, 7, 24, 11, 22, 4]
R0 = [0]*26
R1 = [0]*26
for i in range(26):
    R0[R0_raw[i]] = i
    R1[R1_raw[i]] = i
k = [4,4]
m = 'wetter'
#m = 'illgrv'
c = ''
def Rotate(k):
    k[0] -= 1
    if(k[0]==-1):
        k[1] -= 1
        k[0] = 25
        if(k[1]==-1):
            k[1] = 25
for i in m:
    tmp = ord(i)-ord('a')
    tmp = S[tmp]
    tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
    tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
    tmp = T[tmp]
    tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26
    tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
    tmp = S[tmp]
    c = c + chr(tmp+ord('a'))
    Rotate(k)
print(c)
```

A.2 破解 Enigma 源代码-1

```
    m = 'wetter'
c = 'betrwpeer'
res = ''
print(m)
for i in range(len(c)-len(m)+1):
    tmp = c[i:i+len(m)]
    flag = 0
    for j in range(len(m)):
        if m[j] == tmp[j]:
            flag = 1
            break
    if flag == 1:
        continue
    print(tmp)
    res = tmp

m_raw = m
c = res
c_raw = c

flag = [0]*len(m)
def dfs(s,u):
    # print(u)
    if m[s] == u:
        return 1
    for i in range(len(m)):
        if(flag[i] == 1):
            continue
        if m[i] == u:
            flag[i] = 1
            if dfs(s,c[i]) == 1:
                print((i,m[i],c[i]))
```

```
                flag[i] = 0
                return 1
            flag[i] = 0

    return 0
for i in range(len(m)):
    flag[i] = 1
    if dfs(i,c[i]) == 1:
        print((i,m[i],c[i]))
        print()
    flag[i] = 0

#(0,1,3),(1,2,5)

R0_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16, 21, 19,
    4]
R1_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16, 15,
    23]
T = [10, 20, 14, 8, 25, 15, 16, 21, 3, 18, 0, 23, 13, 12, 2, 5, 6, 19, 9, 17, 1, 7, 24, 11, 22, 4]

R0 = [0]*26
R1 = [0]*26

for i in range(26):
    R0[R0_raw[i]] = i
    R1[R1_raw[i]] = i

def Rotate(k):
    k[0] = (k[0]-1+26)%26
    return k

def Crypt(k,m):
    tmp = m
    # print(tmp)
    tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
    tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
    tmp = T[tmp]
    tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26
```

```
69      tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
70      c = tmp
71      # k = Rotate(k)
72      return c
73
74  def R(k,r):
75      tmp = k
76      for _ in range(r):
77          tmp = Rotate(tmp)
78      return tmp
79
80  def check(l1,l0,l2):
81      l = [-1] * 26
82      # print(l)
83      for i in range(len(l1)):
84          print(l1[i],ord(m_raw[l2[i]])-ord('a'),end=' ')
85          if (l[l1[i]] != -1) and (l[l1[i]] != ord(m_raw[l2[i]])-ord('a')):
86              print()
87              return 0
88          if (l[ord(m_raw[l2[i]])-ord('a')] != -1) and (l1[i] != l[ord(m_raw[l2[i]])-ord('a')]):
89              print()
90              return 0
91          l[ord(m_raw[l2[i]])-ord('a')] = l1[i]
92          l[l1[i]] = ord(m_raw[l2[i]])-ord('a')
93          print(l)
94
95      for i in range(len(l0)):
96          print(l0[i],ord(c_raw[l2[i]])-ord('a'),end=' ')
97          if (l[l0[i]] != -1) and (l[l0[i]] != ord(c_raw[l2[i]])-ord('a')):
98              print()
99              return 0
100         if (l[ord(c_raw[l2[i]])-ord('a')] != -1) and (l0[i] != l[ord(c_raw[l2[i]])-ord('a')]):
101             print()
102             return 0
103         l[ord(c_raw[l2[i]])-ord('a')] = l0[i]
104         l[l0[i]] = ord(c_raw[l2[i]])-ord('a')
105         print(l)
106     return l
```

```
107
108 ans = []
109 for i in range(26):
110     for j in range(26):
111         print([i,j])
112         for m in range(26):
113             k = [i,j]
114             k0 = R([i,j],0)
115             k1 = R([i,j],1)
116             k2 = R([i,j],2)
117             k3 = R([i,j],3)
118             k5 = R([i,j],5)
119             # print(k1,k2,k3)
120             m1 = m
121             c1 = Crypt(k3,Crypt(k1,Crypt(k0,m1)))
122             m2 = Crypt(k0,m1)
123             c2 = Crypt(k5,Crypt(k2,Crypt(k1,m2)))
124             if c1 == m1 and c2 == m2:
125                 m0 = m
126                 m1 = Crypt(k0,m0)
127                 m3 = Crypt(k1,m1)
128                 m2 = m3
129                 m5 = Crypt(k2,m2)
130                 c0 = Crypt(k0,m0)
131                 c1 = Crypt(k1,m1)
132                 c2 = Crypt(k2,m2)
133                 c3 = Crypt(k3,m3)
134                 c5 = Crypt(k5,m5)
135                 if check([m0,m1,m2,m3,m5],[c0,c1,c2,c3,c5],[0,1,2,3,5]) == 0:
136                     print()
137                     continue
138                 print()
139                 S = check([m0,m1,m2,m3,m5],[c0,c1,c2,c3,c5],[0,1,2,3,5])
140                 for l in range(26):
141                     if S[l] == -1:
142                         S[l] = l
143                 print(k)
144                 ans.append([k,S])
```

```
145                 print(ans)
146                 break
147             # print(k)
148 
149 print(ans)
```

A.3 破解 Enigma 源代码-2

```
1  import itertools
2  import tqdm
3  c_raw = 'etrwpe'
4  def Crypt(S,k,t):
5      # S = [2, 1, 0, 4, 3, 5, 7, 6, 8, 9, 24, 11, 20, 13, 14, 15, 16, 17,18, 19, 12, 21, 22, 23, 10, 25]
6      if t == 0:
7          R0_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16,
       21, 19, 4]
8          R1_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16,
       15, 23]
9      else :
10         R1_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16,
       21, 19, 4]
11         R0_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16,
       15, 23]
12     T = [10, 20, 14, 8, 25, 15, 16, 21, 3, 18, 0, 23, 13, 12, 2, 5, 6, 19, 9, 17, 1, 7, 24, 11, 22, 4]
13 
14     R0 = [0]*26
15     R1 = [0]*26
16 
17     for i in range(26):
18         R0[R0_raw[i]] = i
19         R1[R1_raw[i]] = i
20 
21     # k = [4,4]
22 
23     m = 'wetter'
24     c = ''
```

```
25
26      def Rotate(k):
27          k[0] -= 1
28          if(k[0]==-1):
29              k[1] -= 1
30              k[0] = 25
31              if(k[1]==-1):
32                  k[1] = 25
33
34      for i in m:
35          tmp = ord(i)-ord('a')
36          tmp = S[tmp]
37          tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
38          tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
39          tmp = T[tmp]
40          tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26
41          tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
42          tmp = S[tmp]
43          c = c + chr(tmp+ord('a'))
44          Rotate(k)
45      # print(c)
46      # print(ord(c[4])-ord('a'),S[ord(c[4])-ord('a')],ord('p')-ord('a'),S[ord('p')-ord('a')])
47      if ord(c[4])-ord('a') == S[ord(c[4])-ord('a')] and ord('p')-ord('a') == S[ord('p')-ord('a')]:
48          print(S,c,ord(c[4])-ord('a'),S[ord(c[4])-ord('a')],ord('p')-ord('a'),S[ord('p')-ord('a')])
49
50
51  ans = [[[2, 3], [0, 1, 2, 19, 21, 5, 22, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 18, 3, 17, 4, 6, 23, 24,
          25]], [[5, 16], [0, 1, 2, 3, 6, 5, 4, 7, 17, 9, 10, 11, 12, 13, 14, 15, 16, 8, 18, 25, 20, 21, 22,
          23, 24, 19]], [[8, 6], [17, 1, 2, 3, 20, 5, 6, 19, 8, 9, 10, 11, 12, 13, 14, 15, 16, 0, 18, 7, 4, 21,
          23, 22, 24, 25]], [[9, 7], [0, 19, 2, 3, 25, 5, 22, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 18, 1,
          17, 21, 6, 23, 24, 4]], [[12, 16], [0, 1, 17, 3, 24, 5, 6, 7, 8, 9, 10, 11, 12, 22, 14, 15, 16, 2, 18,
          20, 19, 21, 13, 23, 4, 25]], [[18, 4], [0, 19, 2, 22, 24, 5, 17, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          16, 6, 18, 1, 20, 21, 3, 23, 4, 25]], [[19, 6], [0, 1, 2, 4, 3, 5, 6, 7, 17, 19, 10, 22, 12, 13, 14,
          15, 16, 8, 18, 9, 20, 21, 11, 23, 24, 25]], [[19, 20], [0, 1, 2, 3, 24, 5, 6, 17, 8, 9, 10, 11, 12,
          13, 14, 15, 16, 7, 18, 19, 20, 21, 25, 23, 4, 22]], [[20, 20], [0, 1, 2, 3, 24, 22, 6, 7, 8, 17, 10,
          11, 12, 13, 14, 15, 16, 9, 18, 25, 20, 21, 5, 23, 4, 19]], [[25, 1], [0, 1, 2, 17, 11, 5, 6, 22, 8,
          19, 10, 4, 12, 13, 14, 15, 16, 3, 18, 9, 20, 21, 7, 23, 24, 25]], [[25, 8], [0, 1, 2, 3, 15, 5, 6, 7,
          8, 19, 10, 11, 12, 13, 14, 4, 22, 24, 18, 9, 20, 21, 16, 23, 17, 25]], [[25, 16], [0, 22, 4, 3, 2, 5,
```

```
        6, 7, 8, 19, 10, 11, 12, 13, 14, 15, 16, 23, 18, 9, 20, 21, 1, 17, 24, 25]]]

for k,S in ans:
    Crypt(S,k,0)
```

A.4 时间测试代码

```
    S = [2, 1, 0, 4, 3, 5, 7, 6, 8, 9, 24, 11, 20, 13, 14, 15, 16, 17,18, 19, 12, 21, 22, 23, 10, 25]
R0_raw = [0, 18, 24, 10, 12, 20, 8, 6, 14, 2, 11, 15, 22, 3, 25, 7, 17, 13, 1, 5, 23, 9, 16, 21, 19,
    4]
R1_raw = [0, 10, 4, 2, 8, 1, 18, 20, 22, 19, 13, 6, 17, 5, 9, 3, 24, 14, 12, 25, 21, 11, 7, 16, 15,
    23]
T = [10, 20, 14, 8, 25, 15, 16, 21, 3, 18, 0, 23, 13, 12, 2, 5, 6, 19, 9, 17, 1, 7, 24, 11, 22, 4]
import time
def test(n):
    for j in range(n):
        R0 = [0]*26
        R1 = [0]*26
        for i in range(26):
            R0[R0_raw[i]] = i
            R1[R1_raw[i]] = i
        k = [4,4]
        m = 'wetter'
        #m = 'illgrv'
        c = ''
        def Rotate(k):
            k[0] -= 1
            if(k[0]==-1):
                k[1] -= 1
                k[0] = 25
                if(k[1]==-1):
                    k[1] = 25
        for i in m:
            tmp = ord(i)-ord('a')
            tmp = S[tmp]
            tmp = (R0[(tmp + k[0])%26]-k[0]+26)%26
```

```
28              tmp = (R1[(tmp + k[1])%26]-k[1]+26)%26
29              tmp = T[tmp]
30              tmp = (R1_raw[(tmp + k[1])%26]-k[1]+26)%26
31              tmp = (R0_raw[(tmp + k[0])%26]-k[0]+26)%26
32              tmp = S[tmp]
33              c = c + chr(tmp+ord('a'))
34              Rotate(k)
35  st=time.time()
36  test(1000)
37  et=time.time()
38  print(et-st)
```