第二届全国密码数学挑战赛西南赛区

题目二

### 参赛单位： 电子科技大学

### 参赛队员： 詹彤、王雨飞、方淳晟

### 指导老师： 李小平

### 联系电话： xxxxxxxxxxxxx

### 联系邮箱： xxxxxxxxxxxxxxxxxxxxxx

摘要

快速相关攻击算法是解决流密码问题的一种重要方法，也是密码学中重要研究内容之一。自Siegenthaler提出快速相关攻击以来，不断有更高效的算法被提出。

本文基于Meier和Staffelbach提出的Fast correlation attacks中的两种算法，根据提供的被加密的序列密码以及其和原序列的相关概率，利用概率等相关知识，恢复线性移位寄存器的60位初始序列。

本文所用的算法，不仅可以解决快速相关攻击中，密码和输出序列的相关概率较高的流密码问题，也可以解决对于线性移位寄存器的反馈多项式而言，抽头数不高的流密码。因此有着广泛的应用前景。

目录

### 1.序列密码的相关攻击介绍................................................4

### 2.快速相关攻击研究现状...................................................5

### 3.Meier-Staffelbach型算法的基本思想....................................5

### 4.快速相关攻击算法Ⅰ..............................................................6

### 4.1算法Ⅰ实现步骤......................................................6

### 4.2算法Ⅰ计算复杂度分析..........................................7

### 5.快速相关攻击算法Ⅱ..............................................................8

### 5.1算法Ⅱ实现步骤.......................................................8

### 5.2 算法Ⅱ计算复杂度分析..........................................9

### 6.结果部分.................................................................................10

7.相关代码.................................................................................11

7.1校验方程生成代码...................................................12

7.2算法Ⅰ实现代码.......................................................14

7.3算法Ⅰ实现代码.......................................................16

8.参考文献.................................................................................20

1.序列密码的相关攻击介绍

本问题是基于线性反馈移位寄存器LFSR的相关攻击问题。令LFSR的反馈多项式为，一个60位的初始密钥，经过LFSR的线性反馈多项式的运算，输出序列,经由二元无记忆对称信道(BSC)加密，得到截取序列。实现过程如图1所示。

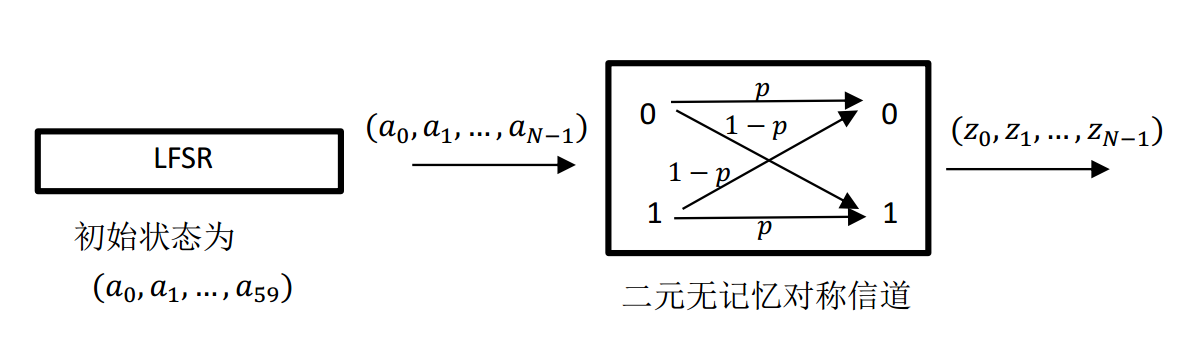


图1 加密过程演示

如果线性函数的某个输入分量un与输出zn之间存在相关性，即p=[P(un=zn)]>0.5。那么可以穷搜索LFSR的初始序列，找出使LFSR的输出与对应密钥流序列z的距离最小的初始序列作为该LFSR的初始序列（种子密钥）。假设LFSR产生最大周期序列，设级数为n，则周期为2n-1。该系统的密钥量为：

K=2n-1 (1)

如果穷搜索密钥攻击算法，n足够大的时候，计算量是无法实现的，事实上，随着n的增大，算法的复杂度呈指数增长。如果相关攻击不对种子密钥进行枚举搜索，就称为快速相关攻击。

2.快速相关攻击研究现状

相关攻击最早是由Siegenthaler提出，其原理是利用密钥发生器的输出序列与其源LFSR的输出序列之间具有的相关性还原LFSR的初始状态。Meier和Staffelbach在此基础上提出利用概率迭代译码的快速相关攻击算法[1]，该方法接收到的密钥流序列的每一比特位代入原序列的反馈多项式组成的方程组中，利用方程组是否满足原序列的反馈多项式的概率，通过一定次数的迭代，来最终找到序列的初始状态，实现解密。在LFSR的特征多项式重量较小（）的情况下取得了较好的效果，之后有学者提出了一些改进算法。

LFSR的抽头数一直是Meier-Staffelbach型相关攻击算法的颈瓶。1999年T．Johansson和F.Jonsson把快速相关攻击应用到LFSR的一般情形，他们的攻击的关键是把LFSR序列转化成卷积码，使用卷积码的译码算法恢复LFSR的初态，仿真结果表明，使用Viterbi译码算法，在抽头数较大时，和以往算法比较，攻击成功时的相关系数减小；接着他们把卷积码和迭代概率译码融合起来，把LFSR序列转变成turbo码，继而使用turbo译码算法恢复LFSR的初态。2000年，V．Chepyzhov，T.Joansson和B．Smeets的部分思想**[2]**，提出一个较为简单而有效的快速相关攻击算法。T.Johansson和F.Jonsson提出不同于以往的BSC攻击模型即线性多项式重构模型，吸取了多项式学习的已有成果进行攻击。

近年来，快速相关攻击技术有了很大的发展，许多文献给出了新的快速相关攻击算法。大多数快速相关攻击算法利用穷举搜索猜测LFSR部分初始状态，再结合密钥流和校验方程集合逐步恢复其余的LFSR初始状态。快速相关攻击新的发展方向是：算法可由高速软件或简单硬件实现，并适于并行运算。

3.基于Meier-Staffelbach型的算法

本文提到的算法Ⅰ和算法Ⅱ，其基本思想是考虑到输出序列和原序列之间存在相关性，通常情况下，因此可以将输出序列看作是原序列的一种扰乱，故可以把序列的每一比特位代入LFSR反馈多项式组成的方程组中。

考虑在模二域上有

 (2)

所以通过对的重复乘方可以得到多个LFSR的校验多项式，对每一个街区序列上的比特位而言，如果某一比特位上，关于该比特位，这些校验多项式成立的个数越多，用该比特位代替LFSR序列的相应比特是正确的概率就越大，

在求解概率问题中主要利用形如式(3)的概率迭代公式

 (3)

以及求解条件概率公式

 (4)

贝叶斯公式主要用于求解算法Ⅱ中的概率问题

 (5)

而上面提到的汉明距离，既是指表示两个（相同长度）字对应位不同的数量。

4.快速相关攻击Ⅰ

4.1 算法Ⅰ实现步骤

设反馈多项式级数为，抽头数为，相关概率，截取序列的长度为。

第1步：计算序列每比特所需要的校验方程平均数：

# (6)

第2步：计算截取序列的比特的模二加（含有个比特）和原LFSR序列的相应比特的模二加相同的概率：



... ...

# (7)

第3步：计算在个校验方程中，至少有个方程成立的概率：

 (8)

第4步：计算在个校验方程中，至少有个方程成立，且它是正确比特位的概率，即概率为：

**** (9)

因此，在给定个方程中至少有个方程成立的条件下，的概率为：

 (10)

第5步：求出一个最大的，记为，使。计算在个比特中错误的平均数r：

 (11)

第6步：找出在个方程中至少满足个方程的比特位，用这些比特位作为相应下标位置上LFSR序列的“正确”比特，然后适当地选取包含个比特的一个集合记为Io，固定个连续比特位。

第7步：对Io中的每一比特位利用LFSR的反馈多项式组建成一个线性方程，即把它表示成固定个比特位的一个线性组合，然后将这个线性方程组合在一起组成一个线性方程组。

第8步：解线性方程，若是线性相关的，则用几个附加比特位选择一个线性无关的子方程组，将解出来的个连续比特扩展为整个LFSR序列，并与原来截取到的序列的位进行比较，若相关概率在之间(通常认为e取0.05)，则认为求解成功。

若相关概率不在区间之内，则认为求解失败，从而以汉明距1,2,...检验的所有变换，每进行以此汉明变换都转入第7步。直到成功求解为止。

4.2 算法Ⅰ计算复杂度分析

算法Ⅰ的主要求解时间耗费在进行汉明变换上。我们可以假设解方程和添加附加位这一过程所用时间为，进行汉明变换的距离为，则总时间为

 (12)

则有T≤，其中H(e)表示二进制的熵函数，。整个算法复杂度为 O（），C取决于p、t、n，N，且0＜C＜1。

5.快速相关攻击Ⅱ

该算法的基本思想： 根据截取序列的每一比特位所满足的方程数。来计算截取序列每一比特的新概率 ，当这些新概率小于给定值的数目超过另一给定值，或者计算新概率的总的次数超过5，就对新概率小于给定值的截取序列的比特位进行取补，接着以新序列代替原序列，并将各比特位的概率重新置为原相关概率，重复上述过程，直到成功。

5.1算法Ⅱ实现步骤：

第1步：计算序列每比特所需要的校验方程平均数

 (13)

第2步：计算截取序列的比特的模二加（含有个比特）和原LFSR序列的相应比特的模二加相同的概率



... ...

 (14)

第3步：计算在个校验方程中，至少有个方程成立的概率

 (15)

计算在个校验方程中，对不满足个方程的比特位进行取补，取补后增加的正确的比特位概率为

 (16)

计算在个校验方程中，有个方程成立时每个比特位的后验概率为



(17)

第4步：找出使得I最大的记为，若，则表示该算法失败，若，则计算，用以表示各比特位的一个阈值，若后验概率,就对该比特位进行取补

 (18)

第5步： 用表示的比特位的数目，则可结束此次迭代，也就是直接对的比特位进行取补，开始下一次迭代：

 (19)

第6步：重置迭代次数i为0

第7步：对截取序列z的每一位比特位和原LFSR序列相应比特位的和是相同的概率S(p1,p2,...,pt,t):

**** (20)

第8步：由贝叶斯公式计算各比特位的新概率：

 (21)

pi\*: 表示第i位比特的薪概率，而pi表示元概率

：在统计校验方程成立的数目时，其表示使得方程成立的的下标。而剩下的则表示不成立的的下标。

第9步：确定的总比数目，若或,则，转到第7步。否则，对截取序列的那些满足的比特位进行取补，并重置比特位的概率位。

第10步：若取补后的截取序列中出现了有不满足反馈多项式的比特位时 , 则转到第6步。直到攻击成功。

5.2算法Ⅱ计算复杂度分析

其算法复杂度为。在、给定的情况下，复杂度随LFSR的级数增大而增大。在其他条件不变的情况下，复杂度随增大而增大，随增大而减小。缺点也很明显，在迭代过程中，可能出现分母为0的情况。

1. 结果部分

下面给出各序列的从到的解。

（1）第一个序列

0,0,1,0,1,0,1,1,1,1,1,0,0,1,1,0,0,0,1,1,1,1,1,0,0,1,0,0,0,0,1,0,1,1,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,0,0,0,1,0,0,0,1,1,1

（2）第二个序列

1,1,0,0,0,1,1,1,0,0,0,0,0,1,0,0,1,0,0,1,1,0,1,0,1,1,1,0,1,1,0,0,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,1,0,0,1,1,0,1,1,0,0,0,1,1

（3）第三个序列

1,1,0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,1,0,0,1,0,0,0,1,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0,0,1,1,0,1

（4）第四个序列

1,0,1,1,1,0,0,1,1,0,1,0,1,1,0,1,1,1,0,1,0,1,1,0,0,1,0,0,1,1,0,1,1,1,0,0,1,1,1,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,0,0

（5）第五个序列

1,0,1,0,1,0,1,1,0,1,1,0,0,1,1,0,0,0,1,1,1,1,1,0,0,1,0,0,0,0,1,0,1,1,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,0,0,0,1,0,0,0,1,1,1

（6）第六个序列

1,1,0,0,0,1,1,1,1,0,0,0,0,1,0,0,1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,1,0,0,1,1,0,1,1,0,0,0,1,1

（7）第七个序列

1,0,0,1,1,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,1,0,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,0,0,1,1,0,0,1,1,0,1

（8）第八个序列

0,1,1,1,0,0,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,0,0,1,1,0,1,1,1,0,0,1,1,1,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,0,0

（9）第九个序列

0,0,1,0,1,0,1,1,1,1,1,0,0,1,1,0,0,0,1,1,1,1,1,0,0,1,0,0,0,0,1,0,1,1,0,1,1,1,0,0,0,1,1,1,1,0,1,0,1,1,1,0,0,1,0,0,0,1,1,1

（10）第十个序列z

1,1,0,0,0,1,1,1,0,0,0,1,0,1,0,0,1,0,0,1,1,0,1,0,1,1,1,0,1,1,0,0,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,1,0,0,1,1,0,1,1,0,0,0,1,1

（11）第十一个序列

1,1,0,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,1,0,0,1,0,0,0,1,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0,0,1,1,0,1

（12）第十二个序列

0,1,1,1,0,0,1,1,0,1,0,1,1,0,1,1,1,0,1,0,1,1,0,0,1,1,0,1,1,0,1,1,1,0,0,1,1,1,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,0,0

7.相关代码

7.1生成校验方程

**import** scipy  
**import** sys,time  
**from** scipy.special **import** comb  
**import** re  
**import** math  
  
N=4000000 *#序列数*n=60 *#级数*p=0.75 *#相关概率*fx=[3,9,16,35,37,47,56,60] *#反馈多项式的系数  
# fx=[1,2,3,5]*t=len(fx) *#计算抽头数***import** scipy  
**import** sys,time  
**import** re  
**import** math  
  
*#读取序列z***def** ReadFile(filename):  
 datafile=open(filename)  
 **try**:  
 text=datafile.read()  
 **return** eval(**'['**+text[4:-1]+**']'**)  
 **finally**:  
 datafile.close()  
  
**def** writeFile(filename,data\_write):  
 **try**:  
 file\_object = open(filename, **'w'**)  
 *#file\_object.write('0.75\n'+str(data).replace(' ','')[1:-1]+',')* file\_object.write(**'0.75\n'** + str(data\_write).replace(**' '**, **''**)[1:-1] + **','**)  
 **finally**:  
 file\_object.close()  
  
N=4000000 *#序列数*n=60 *#级数*fx=[3,9,16,35,37,47,56,60] *#反馈多项式的系数  
# fx=[1,2,3,5]*t=len(fx) *#计算抽头数*M=round(math.log(N/(2\*n),2)\*(t+1)) *#计算每位平均需要校验方程数*imax=math.floor(math.log(N/n,2))  
an = [[0 **for** i **in** range(t + 1)] **for** i **in** range(3 \* M)] *# 用于存储最终校验多项式  
# print(imax)***def** CreatePoly(w): *#w为检测第几位* imax=math.floor(math.log(N/n,2)) *#乘方的最大值* count=0 *#记录校验方程的个数* a1=[[0 **for** i **in** range(t)]**for** i **in** range(imax+1)] *#用以保存校验方程的序号* a1[0]=fx  
 *#生成倍式（第一行为原反馈多项式）* **for** i **in** range(1,imax+1):  
 **for** j **in** range(t):  
 a1[i][j]=a1[0][j] \* pow(2,i)  
 *# print(a1)  
 #生成校验方程* a2=[[0 **for** i **in** range(t+1)]**for** i **in** range(imax+1)]  
 *# print(a2)* **for** column **in** range(imax+1): *#将校验等式左边写进a2的最后一列* a2[column][t]=a1[column][t-1]  
 *# print(a2)* **for** i **in** range(imax+1):  
 **for** j **in** range(t):  
 a2[i][t-j-1]=a2[i][t]-a1[i][j]  
 *# print(a2)* **for** i **in** range(imax+1):  
 **for** j **in** range(t+1):  
 **if** w>=a2[i][j]:  
 offset=w-a2[i][j]  
 **if** (offset+a2[i][t])>N:  
 **continue  
 else**:  
 **for** k **in** range(t+1):  
 an[count][k]=offset+a2[i][k]  
 count=count+1  
  
 **return** count *#返回校验方程总数***def** findBite1():  
 rightnum=[]  
 data = ReadFile(**"data-1.txt"**)  
 **for** w **in** range(4000000):  
 count=0  
 a=CreatePoly(w)  
 **if** a>=92:  
 **for** i **in** range(a):  
 **try**:  
 **if** data[an[i][0]]^data[an[i][1]]^data[an[i][2]]^data[an[i][3]]^data[an[i][4]]^\  
 data[an[i][5]]^data[an[i][6]]^data[an[i][7]]==data[an[i][8]]:  
 *#f=open("test1.txt",'a')  
 #f.write(str(an[i])+',')* count=count+1  
 *#f.close()* **except**:  
 print(an[i][8])  
 print(count)  
 **if** count>92:  
 rightnum.append(w)  
 print(w)  
  
  
  
**def** findBite2():  
 f = open(**"test2.txt"**)  
 a = f.read()  
 data = eval(a)  
 f.close()  
 b = len(data)  
 an = [[0 **for** i **in** range(60)]]  
 **for** i **in** range(60):  
 count = 0  
 **for** k **in** range(10000):  
 **for** j **in** range(9):  
 **if** data[k][j] == 12800 + i:  
 an[i] = 12800 + i  
 count = count + 1  
 print(**"找到"**, count, **"个了"**)  
 **break** print(an)

7.2算法Ⅰ实现代码

**import** scipy  
**import** sys,time  
**from** scipy.special **import** comb  
**import** re  
**import** math  
  
N=4000000 *#序列数*n=60 *#级数*p=0.75 *#相关概率*fx=[3,9,16,35,37,47,56,60] *#反馈多项式的系数  
# fx=[1,2,3,5]*t=len(fx) *#计算抽头数  
  
#step 1:*M=round(math.log(N/(2\*n),2)\*(t+1)) *#计算每位平均需要校验方程数  
  
#step 2:***def** S(p,t):  
 **if** t==1:  
 **return** p  
 **return** p\*S(p,t-1)+(1-p)\*(1-S(p,t-1))  
  
*#step 3:***def** Q(p,M,h):  
 q=0  
 **for** i **in** range(h,M+1):  
 q=comb(M,i)\*(p\*pow(S(p,t),i)\*pow((1-S(p,t)),M-i)+(1-p)\*pow(1-S(p,t),i)\*pow(S(p,t),M-i))+q  
 **return** q  
  
*# print(Q(p,M,1))  
  
#step 4:***def** V(p,M,h):  
 v=0  
 **for** i **in** range(h, M + 1):  
 v = comb(M, i) \* (p \* pow(S(p,i), i) \* pow((1 - S(p,i)), M - i)) + v  
 **return** v  
  
**def** T(p,M,h):  
 t=V(p,M,h)/Q(p,M,h)  
  
*#step 5:***def** findHmax():  
 count=0  
 **for** h **in** range(1,M+1):  
 **if** Q(p,M,h)\*N>=n:  
 count=count+1  
 **return** count  
  
**def** calR():  
 r=(1-T(p,M,h=findHmax()))\*n  
  
  
*#step6  
#从python下标为n的位的连续60个数组***def** Prospective(n,zn\_temp2):  
 point=n  
 *#print(zn\_temp)* zn\_temp=zn\_temp2.copy()  
 **while** point:  
 temp=zn\_temp[3]^zn\_temp[12]^zn\_temp[22]^zn\_temp[24]^zn\_temp[43]^zn\_temp[50]^zn\_temp[56]^zn\_temp[59]  
 zn\_temp.insert(0,temp)  
 zn\_temp.pop()  
 point=point-1  
 print(zn\_temp)  
 **return** zn\_temp  
*#step7***def** LastCheck(zn\_temp):  
 count\_right=0  
 point=60  
 **while** point<=4000000-1:  
 temp =zn\_temp[0]^ zn\_temp[4] ^ zn\_temp[13] ^ zn\_temp[23] ^ zn\_temp[25] ^ zn\_temp[44] ^ zn\_temp[51] ^ zn\_temp[57]  
 zn\_temp.append(temp)  
 zn\_temp=zn\_temp[1:]  
 **if**(zn\_temp[59]==zn[point]):  
 count\_right=count\_right+1  
 point=point+1  
 **return** count\_right/(4000000-60)  
*#step8***def** ChangeHaming(zn\_temp):  
 flag=1  
 count=0  
  
 zn\_temp\_copy=zn\_temp.copy()  
 **while** flag **and** count<=59:  
 Pros\_zn=Prospective(12800, zn\_temp\_copy)  
 result=LastCheck(Pros\_zn)  
 print(result,count)  
 print(**'开头序列'**,Pros\_zn)  
 print(**'选取序列'**,zn\_temp\_copy)  
 **if** result>0.8 **or** result<0.70:  
 zn\_temp\_copy=zn\_temp.copy()  
 zn\_temp\_copy[count]=zn\_temp\_copy[count]^1  
 count=count+1  
 **else**:  
 flag=0  
  
  
  
  
zn=ReadFile(**'data-1.txt'**)  
ChangeHaming(zn[12800:12860])

7.3算法Ⅱ实现代码

**import** scipy  
**from** scipy.special **import** comb  
**import** sys, time  
**import** random  
  
an=[[0 **for** i **in** range(9)]**for** i **in** range(180)]  
data=[0 **for** i **in** range(4000000)]  
list\_p=[0 **for** i **in** range(160)]  
list\_S=[0 **for** i **in** range(160)]  
pn=[0.75 **for** i **in** range(4000000)] *#zn的概率pn  
  
#存储校验正确与错误的校验方程系数*correctpos=[0 **for** i **in** range(160)]  
errorpos=[0 **for** i **in** range(160)]  
  
**def** ReadFile(filename):  
 datafile=open(filename)  
 **try**:  
 text=datafile.read()  
 **return** eval(**'['**+text[4:-1]+**']'**)  
 **finally**:  
 datafile.close()  
**def** WriteFile(filename):  
 **try**:  
 file\_object = open(filename, **'w'**)  
 file\_object.write(**'0.75\n'**+str(data).replace(**' '**,**''**)[1:-1]+**','**)  
 **finally**:  
 file\_object.close()  
  
  
**def** CreatePoly(n):  
 imax=0  
 count=0  
 a0 = [0, 4, 13, 23, 25, 44, 51, 57, 60]  
 ai=[[0, 4, 13, 23, 25, 44, 51, 57, 60] **for** i **in** range(20)]  
 **while** pow(2,imax)\*60<4000000:  
 **for** j **in** range(9):  
 ai[imax][j] = pow(2, imax) \* a0[j]  
 imax=imax+1  
 **for** i **in** range(imax):  
 **for** j **in** range(9):  
 offset=n-ai[i][j]  
 **if** offset>=0:  
 **if** ai[i][8]+offset>=4000000:  
 **continue  
 for** z **in** range(9):  
 an[count][z]=ai[i][z]+offset  
 count=count+1  
 *#print("成功生成",count,"个校验方程")  
 #for i in range(count):  
 # print (an[i])* **return** count  
  
*#s***def** Calusp\_t(p,t):  
 **if** t==0:  
 **return** pow(1-p,t)  
 **return** (pow(2\*p-1,t-1)\*(p-0.5)+0.5)  
  
*#将pi的值打表***def** MakeListOfValue\_p(p):  
 **for** i **in** range(160):  
 list\_p[i]=Calusp\_t(p,i)  
  
  
  
*#第i个校验式的t位相等的概率***def** CalcuS(i,t):  
 **if** t==1:  
 **return** pn[an[i-1][0]]  
 **else**:  
 **return** (1-pn[an[i-1][t-1]])\*(1-CalcuS(i-1,t-1))+pn[an[i-1][t-1]]\*CalcuS(i-1,t-1)  
  
  
  
**def** MakeListOfValue\_S():  
 **for** i **in** range(160):  
 list\_S[i]=CalcuS(i,8)  
  
*#计算第i个位置的条件概率P^\****def** CalcuP(h,i,M):  
 Y=1  
 **for** i **in** range(1,h+1):  
 Y=CalcuS(correctpos[i],8)\*Y  
 **for** i **in** range(h+1,M+1):  
 Y=Y\*(1-CalcuS(errorpos[i],8))  
 Y=Y\*pn[i]  
 T=1  
 **for** i **in** range(1,h+1):  
 T=(1-CalcuS(correctpos[i],8))\*T  
 **for** i **in** range(h+1,M+1):  
 T=T\*CalcuS(errorpos[i],8)  
 T=(1-pn[i])\*T  
 **return** Y/(Y+T)  
  
  
*#Q***def** CalcuU(p,M,h):  
 Q=0  
 s = CalcuS(0, 8)  
 **for** i **in** range(h,M+1):  
 Q=comb(M,i)\*(p\*pow(s,i)\*pow((1-s),M-i)+(1-p)\*pow(1-s,i)\*pow(s,M-i))+Q  
 **return** Q  
  
*#R***def** CalcuV(p,M,h):  
 T=0  
 s=CalcuS(0, 8)  
 **for** i **in** range(1,h):  
 T=comb(M,i)\*(p\*pow(s,i)\*pow((1-s),M-i))+T  
 **return** T  
  
**def** CalcuT(p,M,h):  
 **return** CalcuV(p,M,h)/CalcuU(p,M,h)  
  
  
**def** CalcuW(p,M,h):  
 T=0  
 s=CalcuS(0,8)  
 **for** i **in** range(1,h+1):  
 T=comb(M,i)\*((1-p)\*pow(1-s,i)\*pow(s,M-i))+T  
 **return** T  
  
**def** findHmax(p,M):  
 Hmax=0  
 Imax=0  
 **for** i **in** range(M):  
 *#print(CalcuW(p,M,i),CalcuV(p,M,i))* **if** CalcuW(p,M,i)-CalcuV(p,M,i)>Imax:  
 Hmax=i  
 *#print(Hmax)* **return** 0.5\*(CalcuP(Hmax,p,M)+CalcuP(Hmax+1,p,M)),Hmax  
  
**def** GetEvenVerify(n):  
  
 M=CreatePoly(n)  
 Pthr,Hmax=findHmax(0.75,M)  
 Nthr=CalcuU(0.75, M,Hmax )\*4000000  
 *#S = CalusS(0.75, 8)  
 #Hmax = findHmax(0.75, M, S)* currentnum=0  
 errornum=0  
 **for** j **in** range(M):  
 **if** data[an[j][0]]^data[an[j][1]]^data[an[j][2]]^data[an[j][3]]^data[an[j][4]]^\  
 data[an[j][5]]^data[an[j][6]]^data[an[j][7]]^data[an[j][8]]==0:  
 correctpos[currentnum] = j  
 currentnum=currentnum+1  
 **else**:  
 errorpos[errornum]=j  
 errornum=errornum+1  
 pn[n]=CalcuP(currentnum, n, M)  
 **return** pn[n]<Pthr,Nthr  
  
**def** main():  
 *#M=CreatePoly(1000000)  
 #S=CalusS(0.75,8)  
 #Q=CalcuQ(0.75,M,80,S)  
 #T=CalcuT(0.75,M,80,S,Q)  
 #print(M,Q,T)  
  
 #Hmax=findHmax(0.75,M,S)* **while** 1:  
 data = ReadFile(**"data-1 原本数据.txt"**)  
 **for** i **in** range(3999999):  
 GetEvenVerify(i+1)  
 WriteFile(**"data-1.txt"**)  
  
**def** StartMain():  
 data = ReadFile(**"data-1 原本数据.txt"**)  
 count=0  
 Nw=0  
 **while** 1:  
 **for** i **in** range(3999999):  
 flag,Nthr=GetEvenVerify(i+1)  
 **if** flag:  
 Nw=Nw+1  
 print(**"Nw++"**)  
 **if** Nw<Nthr | count<100:  
 count=count+1  
 **else**:  
 **for** i **in** range(4000000):  
 **if** pn[i]<1:  
 **pass** *# WriteFile("data-1.txt")*StartMain()

8.参考文献

[1] Meier W. and Staffelbach O.“Fast correlation attacks on certain stream ciphers,Journal of Cryptology”, 1(3), pp. 159-176, 1989.

[2]Chepyzhov V.V., Johansson T., Smeets B.“A simple algorithm for fast  
correlation attacks on stream ciphers”.In Goos G., Hartmanis J., van LeeuwenJ., Schneier B. (eds) Fast Software Encryption, FSE 2000, Lecture Notes in Computer Science, vol. 1978, pp. 181-195. Springer, Berlin, Heidelberg.

[3] Canteaut A., Trabbia M.:“Improved fast correlation attacks using parity-check equations of weight 4 and 5”. In Preneel B. (eds) Advances in Cryptology —EUROCRYPT 2000, Lecture Notes in Computer Science, vol. 1807, pp. 573-588. Springer, Berlin, Heidelberg.

[4] 刘琼.流密码的快速相关攻击研究.西安:西安电子科技大学，2010.

[5] 周亮，李胜强.流密码与纠错码联合设计新方向-快速相关攻击译码算法研究进展.成都：电子科技大学，2009.

[6] 张文政.快速相关攻击的新进展.成都:西南通信研究所，1994.