中国科学技术大学

课程论文

密码学导论课程实践论文

作者姓名： 史昊霖

学科专业： 电子信息工程

导师姓名： 李卫海

完成时间： 二〇二一年六月十一日

University of Science and Technology of China

A dissertation for bachelor's degree

A experimental report of the Introduction to Cryptology course practice topic 1: Statistical analysis tools

Author: Shi Haolin

Speciality: Electronic Information Engineering

Supervisor: Weihai Li

Finished time: June 11, 2021

中文内容摘要

本文为密码学导论课程实践的结课论文。

题目一：统计分析工具。使用 Python 作为编程工具，第一个程序以字符形式读取数据，并对字母频率进行统计分析；第二个程序是简单的维吉尼亚密码加密程序；第三个程序通过查找第一个程序中统计出的字符串位置来尝试破解密钥长度。

题目二：AES 算法。使用 Python 作为编程工具，可通过多种方式设定密钥，通过缓存固定矩阵提高速度，以字节形式读取数据后进行分组加密，加解密过程使用查表方式，可以实现快速的 AES 加解密。同时对加密前后的文件数据均匀性进行了统计分析，发现 AES 加密可以有效令数据变得均匀。

题目三：BM 算法。使用 Python 作为编程工具，以字符形式读取数据，按位转换为数字 0、1 进行操作，总体实现了对选定序列线性复杂度及其对应最小阶数的线性反馈移位寄存器的求解。验证了 BM 算法的时间复杂度。

关键词：中国科学技术大学；课程论文；密码学导论；统计分析；AES 算法；BM 算法

Abstract

This paper is the conclusion of the course practice of Introduction to Cryptography.

Topic 1: Statistical analysis tools. Python is used as a programming tool, The first program reads data in the form of characters and makes statistical analysis of the frequency of letters; The second program is a simple Virginia encryption program; The third program tries to crack the key length by looking up the string position counted in the first program.

Topic 2: AES algorithm. Python is used as a programming tool, which can set key in many ways, improve speed by caching fixed matrix, read data in bytes and encrypt by grouping. In addition, table lookup is used in encryption and decryption process to realize fast AES encryption and decryption. At the same time, the data of the file before and after encryption is analyzed. It is found that AES encryption can effectively make the data even.

Topic 3: BM algorithm. Python is used as a programming tool, it read data as characters, and convert them into digits 0 and 1 by bit. The linear complexity of the selected sequence and the linear feedback shift register of the minimum order corresponding to the sequence can be solved. The time complexity of BM algorithm is verified.

Key Words: University of Science and Technology of China (USTC); Course Paper; Introduction to Cryptology; Statistical analysis; AES; Berlekamp-Massey

致　谢

感谢李卫海老师在课程方面的认真讲解和指导帮助。

感谢助教们批改作业、习题课的讲解，也感谢占用了助教课余时间的问题解答。

目 录

第一章　统计分析工具

第一节　实验程序介绍

一、课程内简介

1. 维吉尼亚密码

代换密码：维吉尼亚密码（Vigenère Cipher）是最简单的多表替换密钥，由多个凯撒替换表循环构成。

加密算法：Ci = E(K, P_i) = (P_i + $K_{imod d}$) mod 26

解密算法：Pi = D(K, C_i) = (C_i - $K_{imod d}$) mod 26

2. Kasiski 方法

Kasiski 方法是一种寻找维吉尼亚密码密钥长度的方法。

（1）假设

明文中存在重复字段。

当重复字段的间隔是 d 的整数倍时，将得到重复的密文。

不同的明文获得相同密文的巧合很少发生。

（2）操作过程

在密文中寻找重复字段。

计算重复字段的间距。

密钥长度 d 应是这些间距的公约数。

（3）缺点

查找算法运算量大，耗时长。

偶尔发生的巧合影响机器判断。

二、题目要求

编写一个软件，实现以下功能：统计一段文字中单字符、双字符、三字符的出现频率，分别用直方图、三维直方图、三维密度图表示出来。字符范围应可以指定；画图部分可以包含在你的软件之内，也可以单独使用某个工具实现。

自选一本英文书籍，用该工具分析字母统计分布（字母改为小写，删除所有非字母的符号）。

用维吉尼亚密码加密这本书，对得到的密文进行统计分析。

根据密文统计结果，用 Kasiski 方法分析密钥长度。（你发现了多少可用的串？是否遇到了偶然的干扰?）

三、算法流程图

1. 文本分析统计工具

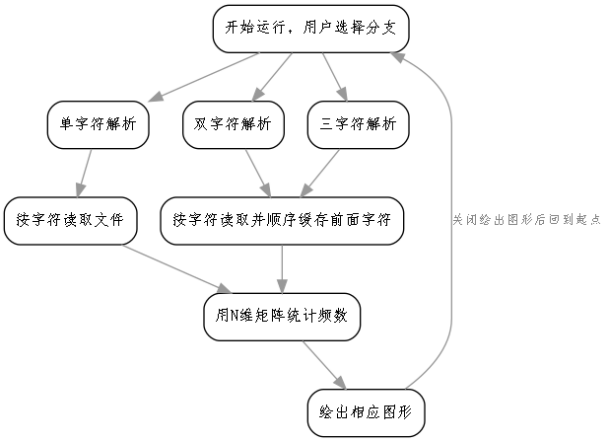

图 1.1　文本分析统计工具-程序流程图

2. 维吉尼亚算法加密

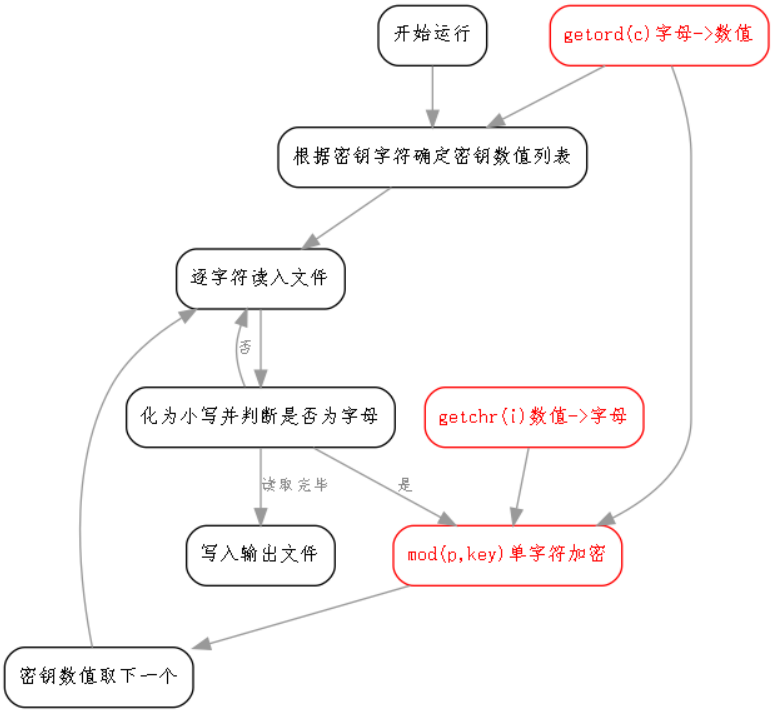

图 1.2　维吉尼亚算法加密-程序流程图

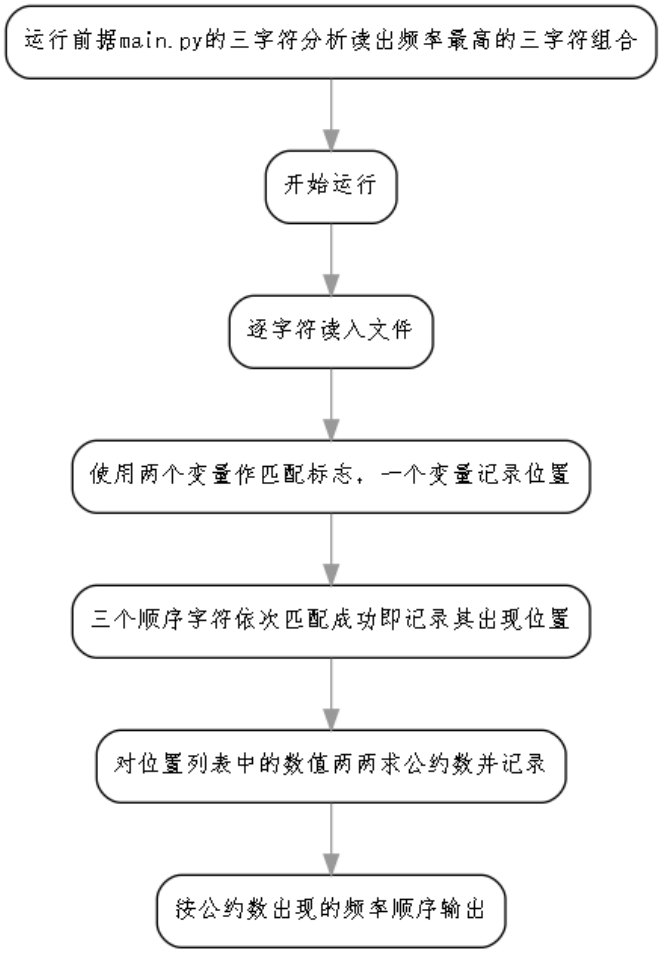

图 1.3　Kasiski 方法分析-程序流程图

3. Kasiski 方法分析

Kasiski 方法分析程序目前需要手动设定待寻找的字符串，这需要借助统计分析工具的结果实现。

四、运行效果图及实验结果分析

选用的文本文件为《老人与海》的英文版 TXT 文本文档。分析时只取英文字母，并将大写字母转换为小写。

文档大小：134KB

1. 原文/密文分析

密文使用维吉尼亚算法加密，密钥为“omymarblues”，11 个字符。

可以看出加密一定程度上令字符的分布更加均匀了，同时改变了字母与频度的对应关系。

这样可以增大字频统计攻击的难度，因此 Vigenere 加密是对凯撒密码的有效改进。

（1）单字符解析

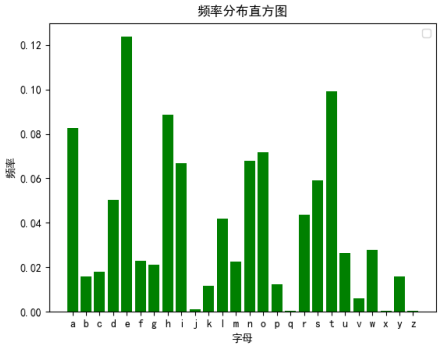

图 1.4　明文

最频繁：e 12640 次

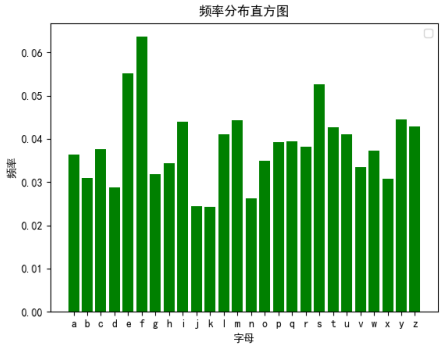

图 1.5　密文

最频繁：f 6350 次

可以看出加密令单个字符频率更为平均，且改变了原本的对应关系。

（2）双字符组合解析

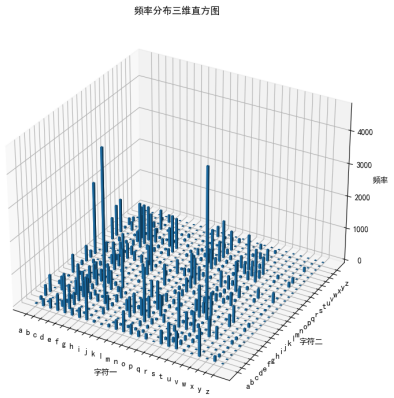

图 1.6　明文

最频繁：he 4.6174%

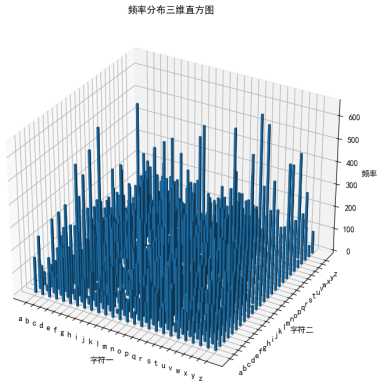

图 1.7　密文

最频繁：fh 0.6561%

与单字符解析结果类似，原本非常突出的峰值被削减了，分布相对而言更加均匀。

（3）三字符组合解析

明文中可以看到突出的深红色点，代表频率非常高的字符组合；

而密文中深红浅红色的点分布更为均匀，更不易找到规律。

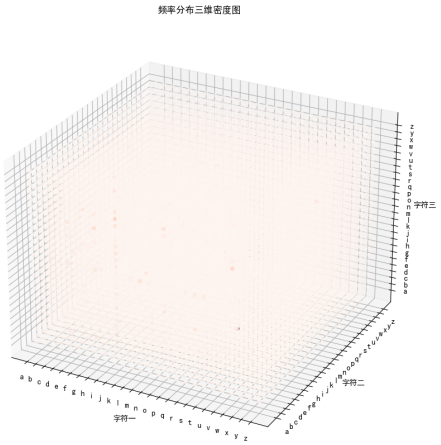

图 1.8　明文

最频繁：the 3.1318%

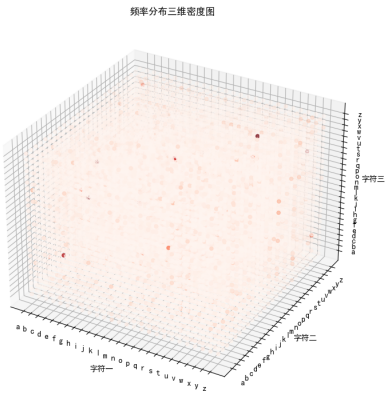

图 1.9　密文

最频繁：htc 0.2935%

五、Kasiski 方法分析结果

先用统计分析工具得出最常出现的三字符为“htc”。

利用参数“htc”对上述密文文件进行分析后得到：

11	24960
22	6208
33	3107
44	1389
66	918
55	808
77	560

表 1.1　公因数出现次数

可以看出 11 是其最常出现的公因数，次数远超其他。

并且很容易看出其他元素也含有 11 这个元素作为因数，因此可以确定 11 为密钥长度。

尝试换用其他密钥加密尝试，如 24 个字符“ohmygodthatisinteresting”。

统计分析工具三字符统计结果：

字符串：zvl 0.0015526861470343694

字符串：lpr 0.0015326514870726356

字符串：zvh 0.0014525128472257004

字符串：atc 0.0013523395474170316

字符串：gai 0.0013423222174361645

字符串：gns 0.0013423222174361645

字符串：lam 0.0013423222174361645

字符串：rns 0.0013122702274935638

这次最常出现的三字符为“zvl”，频率为 0.0015526861470343694。

统计后发现公因数 1 出现了 10203 次，可以看出这三个字符不是我们需要的。

频率第二的三字符为“lpr”，频率为 0.0015326514870726356。

进行尝试后发现公因数 12 出现了 5077 次，可以认为这个字符串查找的结果是有效的。分析得到：

12	5077
4	1336
36	633
1	610
60	252
132	111
84	102

表 1.2　公因数出现次数

可以看出频率高的公因数多为 12 的倍数，可以合理猜测密钥长度也是 12 的倍数，我们已知指定的密钥长度是 24，真实的长度很可能在少数次验证后被猜测出来。

第二节　算法详细与结论

一、算法实现

程序的三种需求（统计、加密、查找密钥长度）分别使用三个 Python 文件实现。

实验程序源码地址：https://github.com/lclichen/Crypt2021/tree/master/CH1-FREQ

二、结论说明

维吉尼亚密码是一种对凯撒密码的有效改进加密方法，可以将密文的字符频率较大程度地均匀化，但还是可以看出一定趋势。

Kasiski 方法在密钥长度为质数或其质因数仅出现一次时效果最好，若质因数多次出现，会产生很大的干扰。

因此使用维吉尼亚密码时密钥长度可以尽量使用较大的合数，可以有效减少被破解的可能。

通过程序进行大量数据的统计可以对古典密码学中密文的破译、密钥长度的寻找提供极大的帮助。

第二章　AES 算法

第一节　实验程序介绍

一、AES 算法简介

AES, Advanced Encryption Standard，其实是一套标准：FIPS 197，而我们所说的 AES 算法其实是 Rijndael 算法。

NIST (National INstitute of Standards and Technology) 在 1997 年 9 月 12 日公开征集更高效更安全的替代 DES 加密算法，第一轮共有 15 种算法入选，其中 5 种算法入围了决赛，分别是 MARS，RC6，Rijndael，Serpent 和 Twofish。又经过 3 年的验证、评测及公众讨论之后 Rijndael 算法最终入选。[1]

二、题目要求

AES 密码需采用课堂中介绍的快速查表方式实现。其中所查各表可在给定密钥后预先算好。

使用公开的 AES 加密数据来测试你的加密、解密工具是否正确。

加密你在题目 1 使用的书籍，统计密文的分布，结果是否均匀?

关于“以 1 个字节为 1 个字符来统计比较困难”的解决想法?

答：按位来统计，因为 AES 算法会将不均匀性扩散到每一位，因此直接对位进行统计更能体现这种状态，同时也能有效减小运算量和存储占用。

三、程序流程图

1. AES 算法加解密程序

四、运行效果

1. 初始化 S/逆 S/T/逆 T 盒及轮密钥表

可以看出使用缓存来保存所有加密中均会用到且不会产生变化的盒文件可以极大提高初始化的速度。

2. 加密标准测试数据

以下为 16 进制数据

明文：00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

密钥：00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

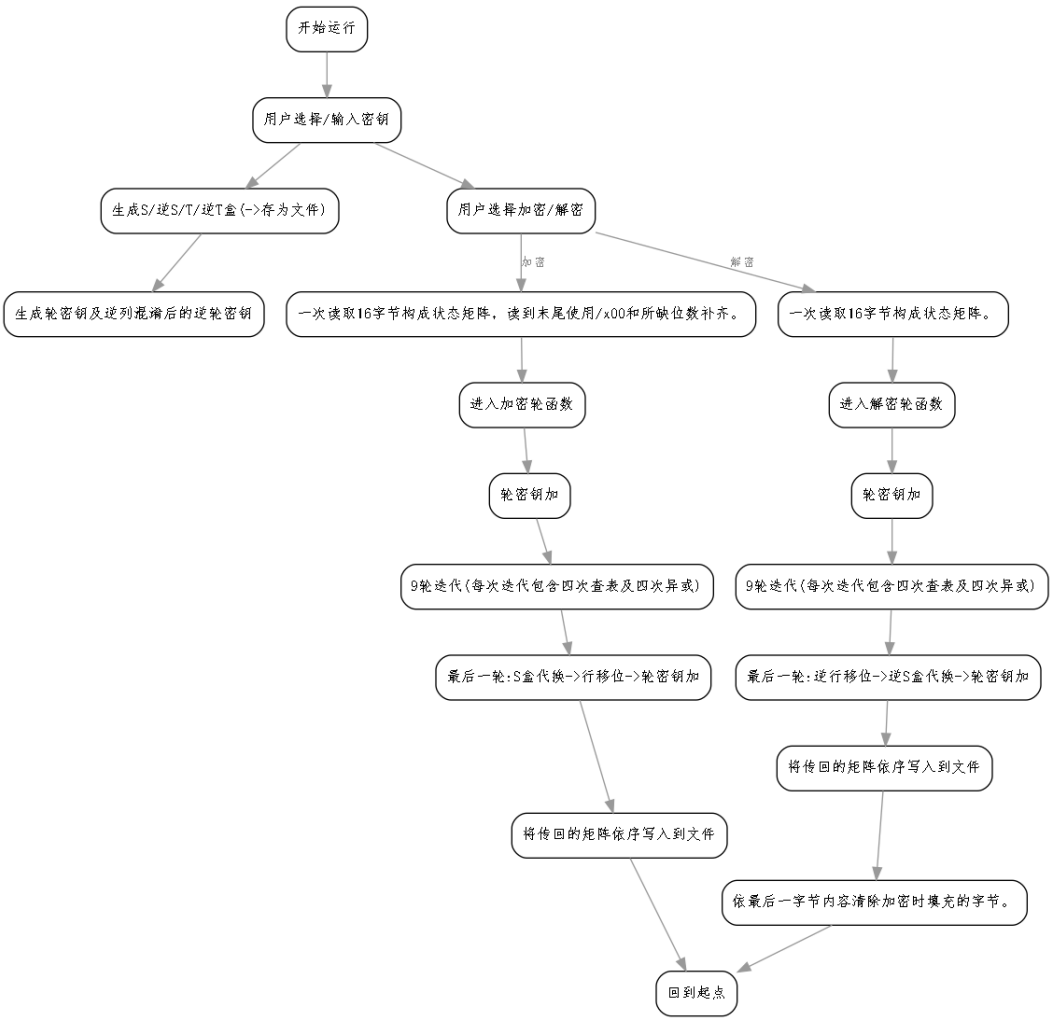

图 2.1　AES 算法-程序流程图

密文：69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a

补齐部分的加密结果因补齐机制的不同而有所不同，但对于普通 ECB 模式的 AES 加密来说，除最后一个分组外的其余部分内容是相同的。

3. 解密标准测试数据

以下为 16 进制数据

明文：69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a

密钥：00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

密文：00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

对于应用了相同补齐机制的 AES 加解密算法，解密时会清除掉加密时为补齐分组所附加的部分。

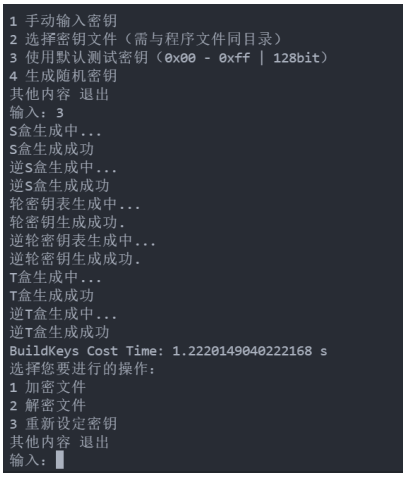

图 2.2　无缓存文件时

生成四个矩阵盒和轮密钥表

用时：1.2220149040222168 s

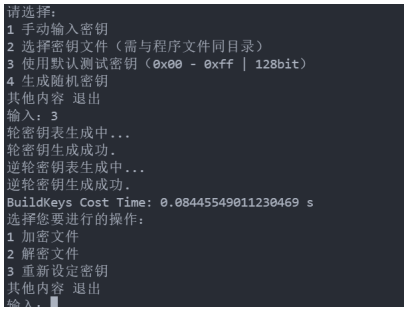

图 2.3　有缓存文件时

盒已存在，仅生成轮密钥表

用时：0.08445549011230469 s

图 2.4　加密效果图

第二节　实验结果分析

一、统计分布

将统计分析程序修改为可自适应位数分组的函数。

从统计结果可以看出，AES 加密可以很好地消除明文的不均匀性，有效将加密扩散到了每一位，由于 0、1 的频数均匀性对比非常直观，因此没有画图。

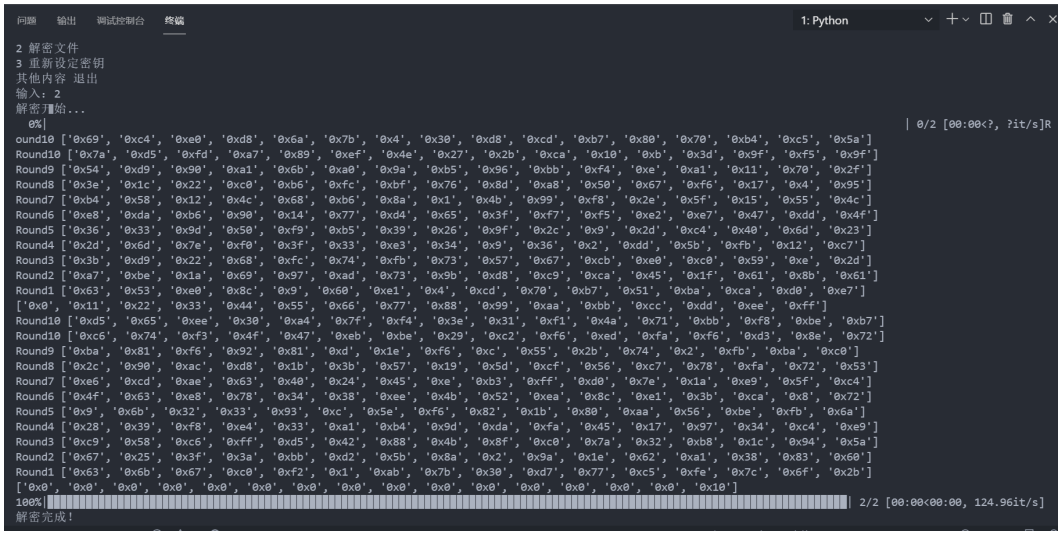

图 2.5　解密效果图

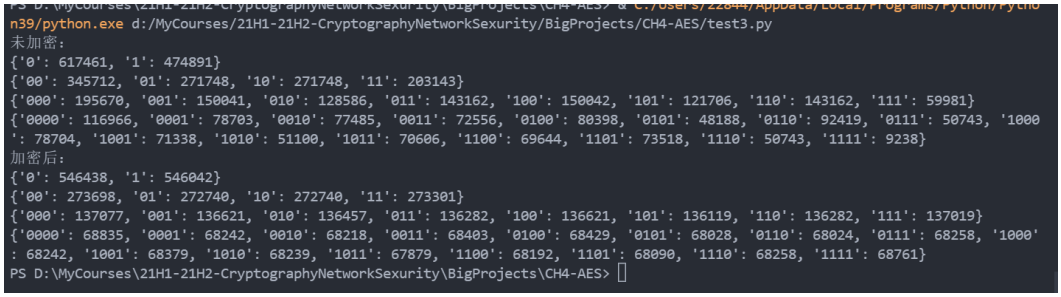

图 2.6　加密前后 0/1 位频度统计对比

第三节　算法详细与结论

一、用户交互说明

1. 选定密钥

（1）手动输入 16 位 ASCII 码

（2）手动输入 32 位 HEX 文本

（3）选定一个文件作为密钥来源，读取其开头 16 个字节

（4）使用默认测试密钥：00-0f 的 16 个字节[2]

（5）生成随机密钥

可选保存到文件、复制 hex 值到剪贴板、存为用 hex 值表示的文本文档。

2. 加密文件

先输入要被加密的文件名，再输入要输出的文件名。完成输入就会开始加密。

3. 解密文件

先输入要被解密的文件名，再输入要输出的文件名。完成输入就会开始解密。

二、算法实现

程序使用单个 Python 文件实现，运行一次后会生成可重复利用的矩阵缓存文件，可以加快以后的加密速度。

实验程序源码地址：https://github.com/lclichen/Crypt2021/tree/master/CH4-AES

1. 密钥确定后生成运算所需表的过程

主要描述密钥初次生成过程。

（1）生成 S 盒

初始化 S 盒。x 行 y 列的元素值为 $(xy)_{16}$。利用扩展欧几里得算法求 S 盒中每个元素在 GF(2^8) 域中的逆。进行字节代换，求得可用的 S 盒，缓存以便以后使用。

（2）生成逆 S 盒

由于逆 S 盒 (S 盒 (a)) = a。可以利用 S 盒简单求得逆 S 盒。

（3）生成 T 盒

在 32 位处理器上，可预先算出 4 张 256 字的 T 盒代替每轮加密运算中的 S 盒字节代换、行移位、列混淆操作，这样在实际加密时可以省去大量运算，只需查表，既加快了运算速度也提高了安全性。

初始化 256*4 的全零矩阵，使用有限域内多项式乘法，计算 [S 盒]*[行移位 & 列混淆用矩阵]，得出 T 盒。

（4）生成逆 T 盒

与 T 盒类似，计算 [逆 S 盒]*[逆行移位 & 逆列混淆用矩阵]，得出逆 T 盒。

（5）生成轮密钥

（6）生成逆轮密钥

将轮密钥表复制一份，对其除第一个和最后一个 4*4 矩阵块以外的部分，按照 4*4 矩阵为单元，进行逆列混淆，即可得到逆轮密钥矩阵。

2. 课程 PPT 算法流程图

这张截图清晰展示了 AES 加密的整体流程，其中蓝色底色的部分便是可以使用 T 盒查表操作代换的部分，使用 T 盒查表再与轮密钥异或，可将每个轮函

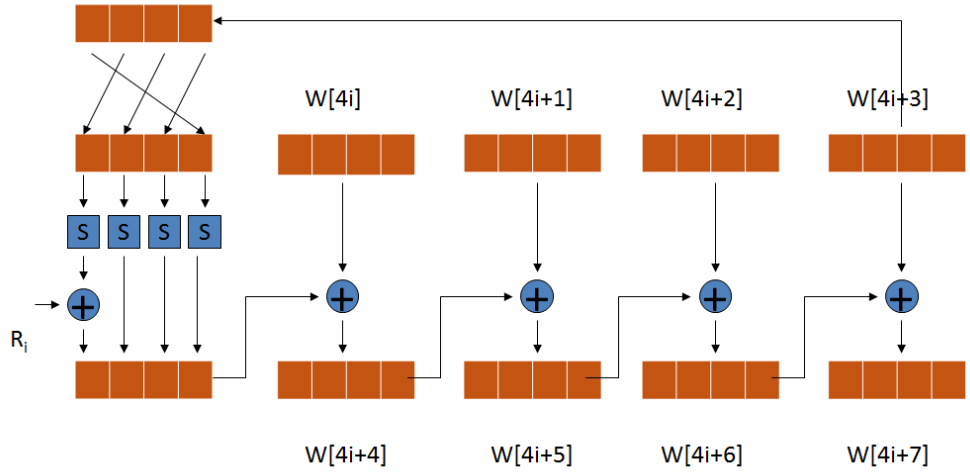

图 2.7　轮密钥扩展

输入 4*4 的密钥矩阵，扩展至 4*44。

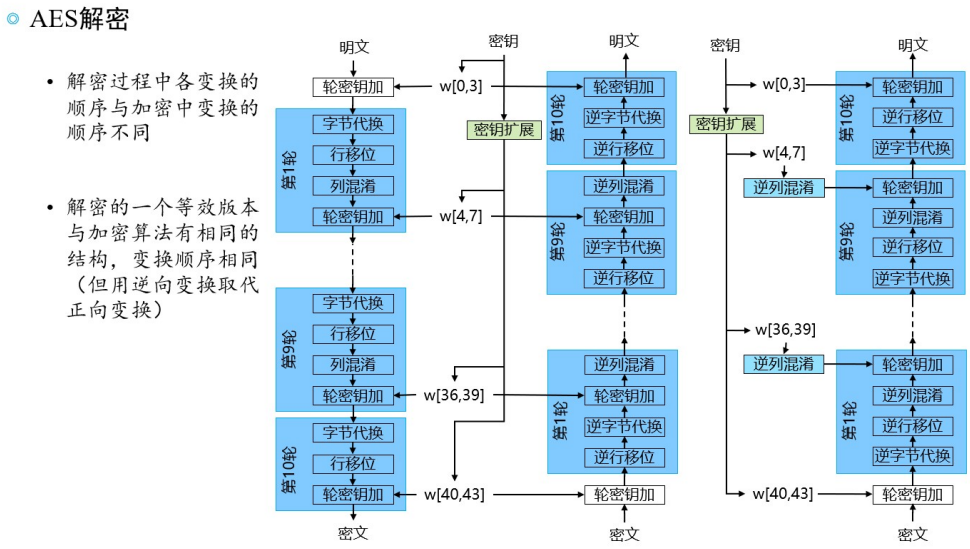

图 2.8　课件截图

数简化为四次查表和四次异或操作。

3. 加密过程

（1）初始化

先确定密钥，然后导入要加密的文件。

以 16 字节（128bit）为一组，置入状态矩阵进行加密操作。

最后一组不足 16 字节时，在末尾补 0，最后一个补上的字节为缺少的字节数；刚好 16 字节时，补 15 字节的 0，末尾为 16。

以下以单组的加密过程为例，其他分组与此一致。

（2）轮密钥加 w[0,3]

（3）9 轮 T 盒代换并与该轮对应轮密钥异或

（4）S 盒代换

（5）行移位

（6）轮密钥加 w[40,43]

将当前状态矩阵中的数据按顺序导出并写入输出的文件中，然后读取下一个分组。

全部加密完成后，即可得到加密的文件。

4. 解密过程

先确定密钥，然后导入要解密的文件。

以 16 字节（128bit）为一组，置入状态矩阵进行解密操作。

由于加密时进行了补齐操作，故导入的数据一定是 16 字节的整数倍，可以刚好导入完毕。

以下以单组的解密过程为例，其他分组与此一致。

（1）轮密钥加 w_{inv}[40,43]

（2）9 轮逆 T 盒代换并与该轮对应逆轮密钥异或

（3）逆行移位

（4）逆 S 盒代换

（5）轮密钥加 w_{inv}[0,3]

将当前状态矩阵中的数据按顺序导出并写入输出的文件中，然后读取下一个分组。

当所有要解密的数据读取完毕后，根据之前的补零方式，读取末尾的字节的数字为要删去的长度，移动文件指针至该位置并切除后面的内容。

即可得到解密的文件。

三、结论说明

1. 值得注意的点

在 32 位或以上的计算机中，无符号整数长度是 32bit，每个占 4 字节。最开始以为 ubyte/uint 是 8bit 长度，后来发现不是。

numpy 中对矩阵使用等号赋值是指针形式的，如果想不改变原矩阵则需要使用 copy() 创建一个浅拷贝。

在以矩阵形式计算完成 S 盒后，可将 16*16 的矩阵化为 1*256 的形式，更便

于查表操作。

对矩阵的处理要谨慎，在第一次测试加解密完成后整理代码并做交互接口时可能是动了哪里的函数导致加密正常解密不正常，最后发现在生成逆轮密钥的逆列混淆函数中的矩阵需要转置后计算再转置回来才是正确的。说明编程过程中程序设计思路不够明确，对算法和矩阵运算过程理解不深入。后面需要养成更好的编程习惯，规范代码风格，避免出现不该出现的 bug。

编程时应充分考虑程序的泛用性和易读性，尽量使用函数或类封装接口，而不是使用主函数混杂子函数的形式。这样既有助于后期修改也有助于函数复用。

第三章　BM 算法

第一节　实验程序介绍

一、课程内简介

BM 算法全称 Berlekamp-Massey 算法。

用于求解最小阶数的线性反馈移位寄存器，使之输出的前 n 个比特与目标序列相同。

定义 d=s_k+c_1 s_{k-1}+c_2 s_{k-2}+$\cdots$+c_L s_{k-L} 为迭代到第 k 轮时的下一步离差。即第 k-1 轮迭代结果对 s_k 的预测与实际 s_k 的差。

运行时间为 O(2^n) 次比特操作。

设 s 的线性复杂度为 L, 则以 s 的一个长度至少为 2L 的子序列为输入的 B-M 算法可以唯一确定生成 s 且长度为 L 的 LFSR。

二、题目要求

读入一个 0/1 串，串的长度在 10 比特到 1M 比特之间。

输出线性复杂度及相应的联结多项式。

以题目 2 的 AES 工具加密题目 1 中使用的书籍，在密文中任意截取若干长度为 1kb, 5kb, 10kb, 20kb 的子串，测试它们的线性复杂度。

三、算法流程图

1. BM 算法求解程序

四、运行效果图

1. 求解题干中示例序列 1001101001101
2. 求解 20Kbit 数据（数据来自预先生成的随机字节文件）

   由于结果过长，将输出结果存到文本文件中。

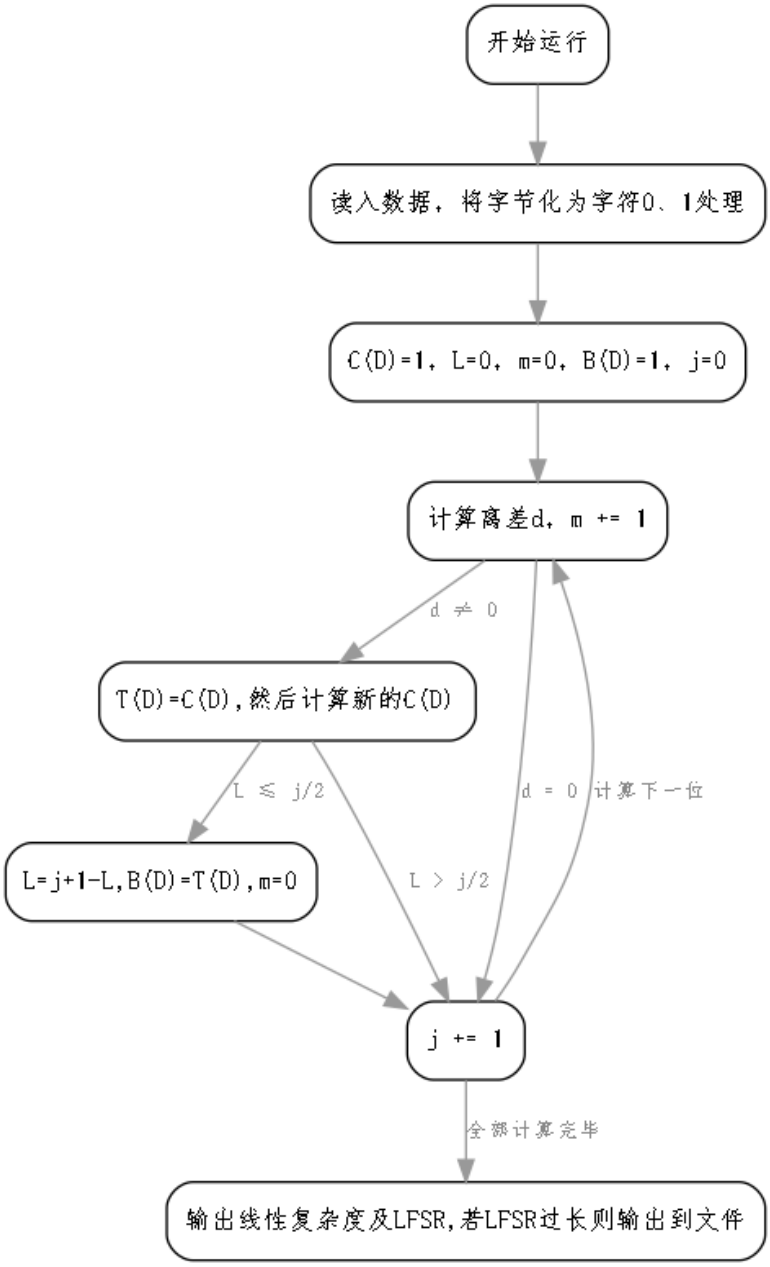

图 3.1　BM 算法-程序流程图

第二节　实验结果分析

一、密文测试

为便于对比，对于本次测试，采取同一个随机偏移量，故先使用 random 函数取一个范围 [0,134000] 的字节偏移量，为 96995。

二、有效性测试

采用 PPT 中示例序列、网络上搜索的测试序列进行测试，可以确认程序有效。

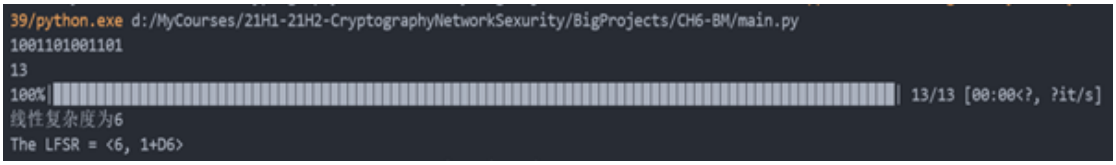

图 3.2　示例序列测试运行效果

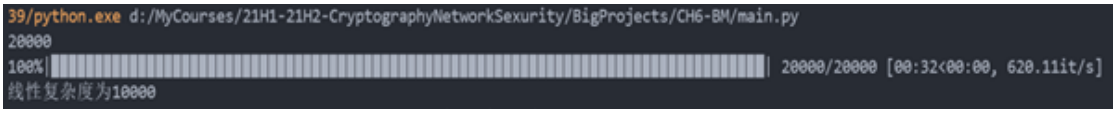

图 3.3　20Kbit 数据序列测试运行效果

图 3.4　输出的文本文件

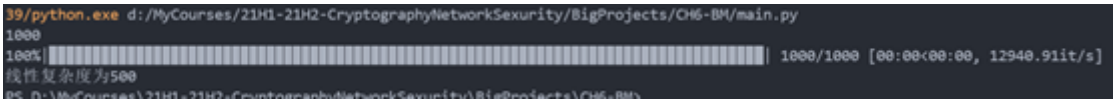

图 3.5　1Kbit 测试

线性复杂度：500 time 函数计时：130.28264045715332 ms

（1）序列：001101110（课堂 PPT 中示例）

线性复杂度为 5

The LFSR = <5, 1+D3+D5>

经调试、打印、对照，程序运算过程及运算中各项的值与教案中完全一致。

（2）序列：1001101001101（题干中示例）

线性复杂度为 6

The LFSR = <6, 1+D6>

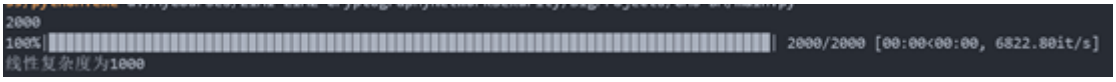

图 3.6　2Kbit 测试

线性复杂度：1000 time 函数计时：517.6577568054199 ms

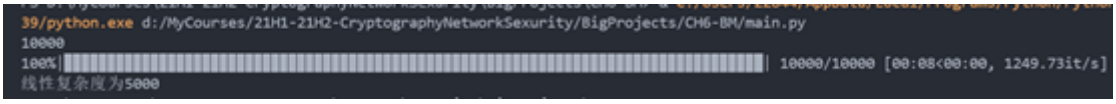

图 3.7　10Kbit 测试

线性复杂度：5000 time 函数计时：11528.244018554688 ms

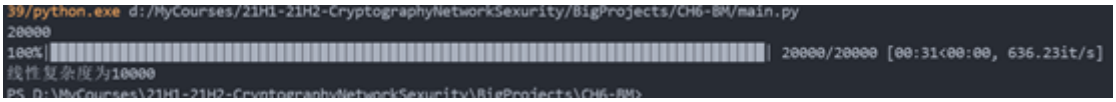

图 3.8　20Kbit 测试

线性复杂度：10000 time 函数计时：36218.80531311035 ms

（3）序列：00101101（来源网络 https://blog.csdn.net/qq_33877253/article/details/99622328）

线性复杂度为 4

The LFSR = <4, 1+D1+D2>

（4）序列：10101111（来源网络）

线性复杂度为 4

The LFSR = <4, 1+D3+D4>

第三节　算法详细与结论

一、算法实现

程序使用单个 Python 文件实现。

实验程序源码地址：https://github.com/lclichen/Crypt2021/tree/master/CH6-BM

算法3.1 主要描述计算过程。

二、结论说明

随着数据量的增大，每一位的计算量呈上升趋势，这既是由于 L 的增大使得步骤 2 中异或的次数增大，也由于 C(D)、B(D) 的长度增加使得其单次乘法运算

算法 3.1 BM 算法

```
Data: 序列文本
Result: 线性复杂度及 LFSR
初始化类 BM;
初始化：导入数据，为各计算项赋初值;
while j < N do
    m += 1;
    d = data[j];
    if L ≠ 0 then
        i = 0;
        while 1 ⩽ i ⩽ L+1 do
            d = d XOR (data[j - i] * C(D)[i]);
            i += 1;
        end
    else
    end
    if d ≠ 0 then
        T(D) = C(D);
        C(D) = C(D) XOR B(D)*D^m;
        if L ⩽ j/2 then
            L = j+1-L;
            B(D) = T(D);
            m = 0;
        else
        end
    else
    end
    j += 1;
end
```

中需要的移位次数越来越多。

根据测试数据中的平均速度对比可以看出，位数增大 N 倍，单步计算的平均时间便增大 N 倍，同时因为有多少位便要进行多少次循环，步数也增大到原先的 N 倍。可以看出时间复杂度为 $O(n^2)$。

该算法主要内容为先确定一个多项式，然后逐位检验数据来修正多项式。

读取文件时首个字节在读取时会被去掉首位的 0，因此引入了简单的长度判断用于补 0。

在看 PPT 上算法内容时，由于没有进行放映，其拼写检查挡住了 L≤j/2 中的等号，检查了很多次程序都看不出问题，之后尝试自行增加了等号后发现可以得出正确结果，然后还跑去问老师这个是不是 PPT 上写错了，实在是非常尴尬。

第四章　总结

第一节　创新性说明

由于实在找不到密度云的绘制方法，因此只能采用 Python 中 Matlibpot 库的 3D 描点绘图，发现可以将数据存为一整个矩阵后进行绘制，而非按点描绘，可以显著提高速度。

AES 算法中生成 S 盒后将 S 盒扁平化，更便于调用。固定不变的盒文件生成后可以重复利用，加快加密前准备的速度。

BM 算法中使用可变长度的列表储存数据，可以对任意长度的序列进行运算。

测试数据较为全面，在 BM 算法中采取了各种渠道的数据进行验证，同时程序可以输出每一步的结果便于理解算法本身。

第二节　课程实践收获

一、提升了编程能力

全部代码均采用 Python 实现，对 Python 中类的定义和接口的封装等有了更深刻的了解，可以更熟练运用子函数等加快编程速度和完善代码可读性。

二、对论文的写作有了更多的了解

第一次使用 LaTeX 撰写论文，提高了熟练度和对格式的了解，为日后的学习提高了基础能力。

三、不足之处

对 LaTeX 插入图片的方法还是不太了解，导致论文中很多图片位置不如预期。

代码风格不够简洁明朗，还需要更多的学习改进。

参 考 文 献

[1] MATT WU. AES 简介[EB/OL]. 2017. https://github.com/matt-wu/AES/blob/master/README.md.

[2] Federal Information Processing Standards Publication 197[EB/OL]. 2001. https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf.