

中国科学技术大学

学士学位论文



密码学导论课程实践

题目二 AES 算法

作者姓名： 史昊霖

学科专业： 电子信息工程

导师姓名： 李卫海 副教授

完成时间： 二〇二一年五月二十六日

University of Science and Technology of China
A dissertation for bachelor's degree



A experimental report of the Introduction to Cryptology course practice topic 2: AES algorithm

Author: Shi Haolin

Speciality: Electronic Information Engineering

Supervisor: Asso. Prof. Weihai Li

Finished time: May 26, 2021

中文内容摘要

本科生需要手动将摘要置于目录后。

本文为密码学导论课程实践的题目二：AES 算法的实验报告。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。

关键词：中国科学技术大学；课程论文；密码学导论；AES 算法；实验报告

Abstract

This paper is a experimental report of the Introduction to Cryptology course practice 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.

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

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

中文内容摘要	I
英文内容摘要	II
第一章 实验程序介绍	2
第一节 程序流程图	2
一、AES 算法加解密程序	2
第二节 运行效果	2
一、初始化 S/逆 S/T/逆 T 盒及轮密钥表	2
二、加密标准测试数据	3
三、解密标准测试数据	4
第二章 实验结果分析	5
第一节 AES 算法简介	5
第二节 题目要求	5
第三节 统计分布	5
第三章 算法详细与结论	6
第一节 用户交互说明	6
一、选定密钥	6
二、加密文件	6
三、解密文件	6
第二节 算法实现	6
一、密钥确定后生成运算所需表的过程	6
二、课程 PPT 算法流程图	7
三、加密过程	8
四、解密过程	9
第三节 结论说明	9
一、值得注意的点	9
参考文献	11

第一章 实验程序介绍

第一节 程序流程图

一、AES 算法加解密程序

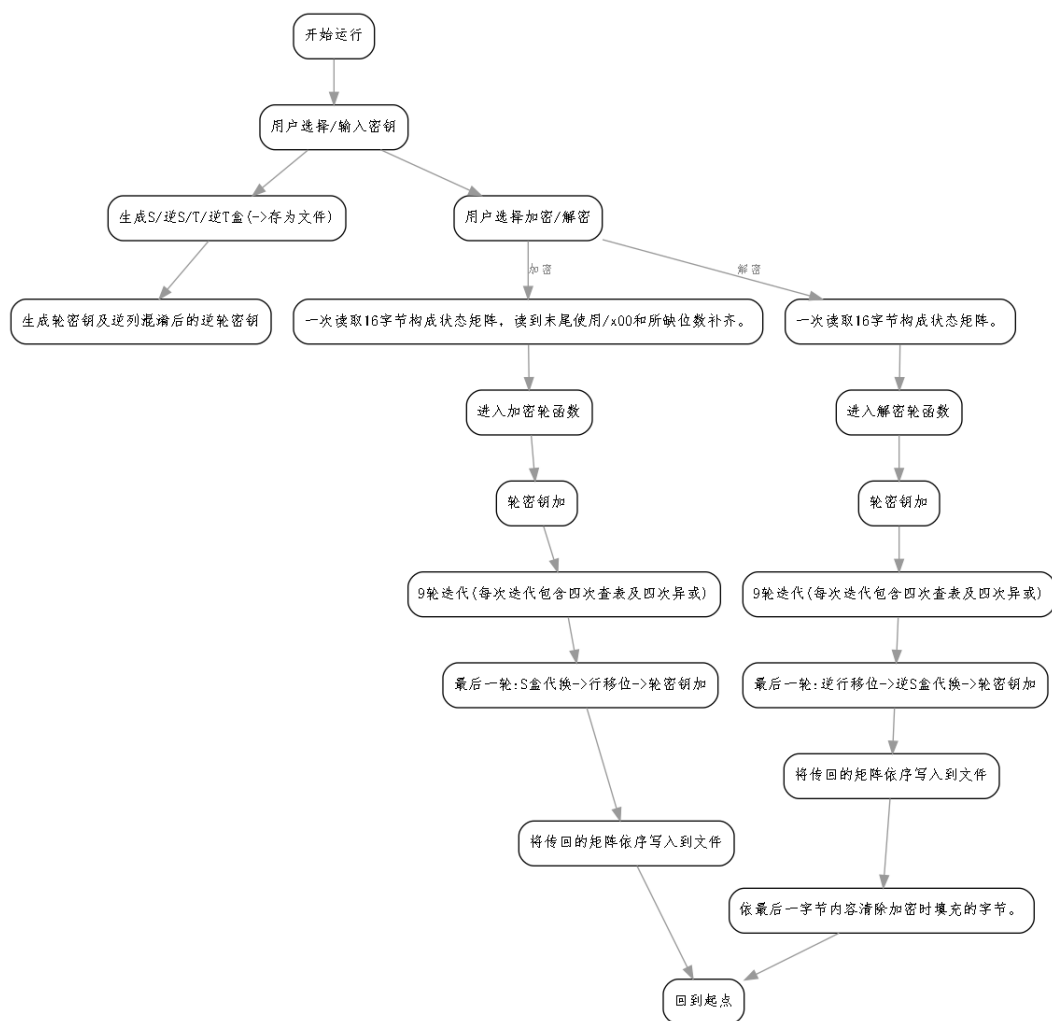


图 1.1 AES 算法-程序流程图

第二节 运行效果

一、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。

```

1 手动输入密钥
2 选择密钥文件（需与程序文件同目录）
3 使用默认测试密钥（0x00 - 0xff | 128bit）
4 生成随机密钥
其他内容 退出
输入：3
S盒生成中...
S盒生成成功
逆S盒生成中...
逆S盒生成成功
轮密钥表生成中...
轮密钥表生成成功
逆轮密钥表生成中...
逆轮密钥表生成成功
T盒生成中...
T盒生成成功
逆T盒生成中...
逆T盒生成成功
BuildKeys Cost Time: 1.2220149040222168 s
选择您要进行的操作：
1 加密文件
2 解密文件
3 重新设定密钥
其他内容 退出
输入：

```

图 1.2 无缓存文件时

生成四个矩阵盒和轮密钥表
用时：1.2220149040222168 s

```

请选择：
1 手动输入密钥
2 选择密钥文件（需与程序文件同目录）
3 使用默认测试密钥（0x00 - 0xff | 128bit）
4 生成随机密钥
其他内容 退出
输入：3
轮密钥表生成中...
轮密钥表生成成功
逆轮密钥表生成中...
逆轮密钥表生成成功
BuildKeys Cost Time: 0.08445549011230469 s
选择您要进行的操作：
1 加密文件
2 解密文件
3 重新设定密钥
其他内容 退出
输入：

```

图 1.3 有缓存文件时

盒已存在，仅生成轮密钥表
用时：0.08445549011230469 s

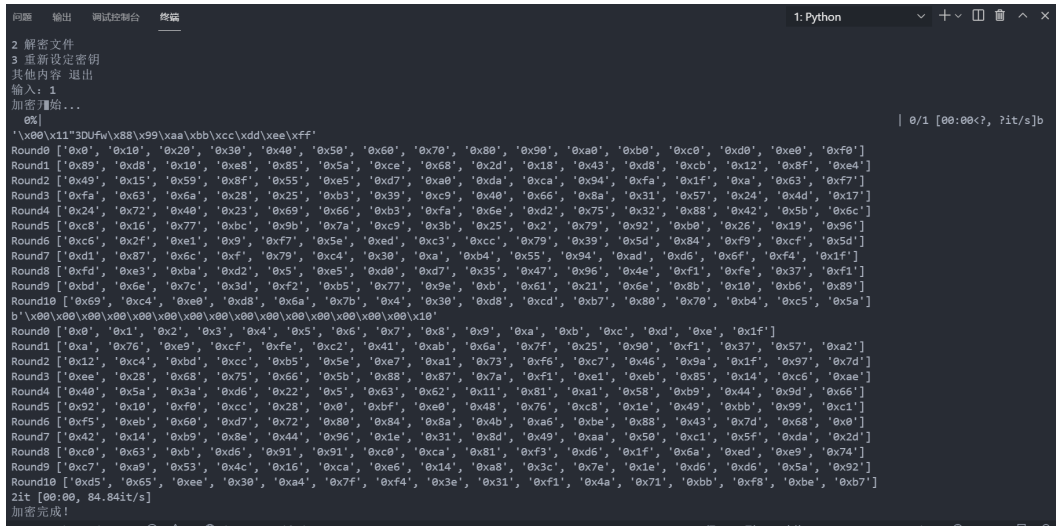
二、加密标准测试数据

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

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



```

1: Python
2 解密文件
3 重新设定密钥
其他内容 退出
输入：1
加密开始...
0x0
'\x00\x11'3DUfu\x88\x99\xaa\xbb\xcc\xdd\xee\xff'
Round0 ['0x0', '0x10', '0x20', '0x30', '0x40', '0x50', '0x60', '0x70', '0x80', '0x90', '0xa0', '0xb0', '0xc0', '0xd0', '0xe0', '0xf0']
Round1 ['0x8', '0xd8', '0x18', '0xe8', '0x5', '0x5a', '0xc', '0x68', '0x2d', '0x18', '0x43', '0xd8', '0xcb', '0x12', '0x8f', '0xe4']
Round2 ['0x49', '0x15', '0x59', '0x8f', '0x5', '0x5', '0xd7', '0xa0', '0xda', '0xca', '0x94', '0xfa', '0x1f', '0xa', '0x63', '0xf7']
Round3 ['0xfa', '0x63', '0x6a', '0x28', '0x25', '0xb3', '0x39', '0xc9', '0x40', '0x66', '0x8a', '0x31', '0x57', '0x24', '0x4d', '0x17']
Round4 ['0x24', '0x72', '0x40', '0x23', '0x69', '0x66', '0xb3', '0xfa', '0x6e', '0xd2', '0x75', '0x32', '0x88', '0x42', '0x5b', '0x6c']
Round5 ['0xc8', '0x16', '0x77', '0xbc', '0x9b', '0x7a', '0xc9', '0x3b', '0x25', '0x2', '0x79', '0x92', '0xb0', '0x26', '0x19', '0x9e']
Round6 ['0xc6', '0x2f', '0xe1', '0x9', '0xf7', '0x5e', '0xed', '0xc3', '0xcc', '0x79', '0x39', '0x5d', '0x84', '0xf9', '0xcf', '0x5d']
Round7 ['0xd1', '0x87', '0xc6', '0xf', '0x79', '0xc4', '0x30', '0xa', '0xb4', '0x55', '0x94', '0xad', '0xd6', '0x6f', '0xf4', '0x1f']
Round8 ['0xfd', '0xe3', '0xba', '0xd2', '0x5', '0xe5', '0xd9', '0xd7', '0x35', '0x47', '0x96', '0x4e', '0xf1', '0xfe', '0x37', '0xf1']
Round9 ['0xbd', '0xe', '0x7c', '0x3d', '0xf2', '0xb5', '0x77', '0x9e', '0xb', '0x61', '0x21', '0x6e', '0xb', '0x10', '0xb6', '0x89']
Round10 ['0x69', '0xc4', '0xe0', '0xd8', '0x6a', '0x7b', '0x04', '0x30', '0xd8', '0xcd', '0xb7', '0x80', '0x70', '0xb4', '0xc5', '0x5a']
Round11 ['0x0', '0x1', '0x2', '0x3', '0x4', '0x5', '0x6', '0x7', '0x8', '0x9', '0xa', '0xb', '0xc', '0xd', '0xe', '0xf']
Round12 ['0x12', '0xc4', '0xbd', '0xcc', '0xb5', '0x5e', '0xe7', '0xa1', '0x73', '0xf6', '0xc7', '0x46', '0x9a', '0x1f', '0x97', '0x7d']
Round13 ['0xae', '0x28', '0x68', '0x75', '0x66', '0x5b', '0x88', '0x87', '0x7a', '0xf1', '0xe1', '0xeb', '0x85', '0x14', '0xc6', '0xae']
Round14 ['0x40', '0x5a', '0x3a', '0xd6', '0x22', '0x5', '0x63', '0x62', '0x11', '0x81', '0xa1', '0x58', '0xb9', '0x44', '0x9d', '0x66']
Round15 ['0x92', '0x10', '0xf0', '0xc', '0x28', '0x5', '0xbf', '0xe0', '0x48', '0x75', '0xc8', '0x1e', '0x49', '0xbb', '0x9d', '0xc1']
Round16 ['0xf5', '0xeb', '0x60', '0xd7', '0x72', '0x80', '0x84', '0x8a', '0x4b', '0xa6', '0xbe', '0x88', '0x43', '0x7d', '0x68', '0xb']
Round17 ['0x42', '0x14', '0xb9', '0x8e', '0x44', '0x96', '0x1e', '0x31', '0x8d', '0x49', '0xaa', '0x50', '0xc1', '0x5f', '0xda', '0x2d']
Round18 ['0xc0', '0x63', '0xb', '0xd6', '0x91', '0x91', '0xc0', '0xca', '0x81', '0xf3', '0xd6', '0x1f', '0x6a', '0xed', '0xe9', '0x74']
Round19 ['0xc7', '0xa9', '0x53', '0x4c', '0x16', '0xca', '0xe6', '0x14', '0xa8', '0x3c', '0x7e', '0x1e', '0xd6', '0x5a', '0x92']
Round10 ['0xd5', '0x65', '0xee', '0x30', '0xa4', '0x7f', '0xf4', '0x3e', '0x31', '0xf1', '0x4a', '0x71', '0xbb', '0xf8', '0xbe', '0xb7']
2it [00:00, 84.84it/s]
加密完成！

```

图 1.4 加密效果图

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。

第二章 实验结果分析

第一节 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]

第二节 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。

第三节 统计分布

```
PS D:\MyCourses\21H1-21H2-CryptographyNetworkSecurity\BigProjects\CH4-AES> .\C:\Users\22844\AppData\Local\Programs\Python\Python39\python.exe d:/MyCourses/21H1-21H2-CryptographyNetworkSecurity/BigProjects/CH4-AES/test3.py
未加密:
{'0': 617461, '1': 474891}
{'00': 345712, '01': 271748, '10': 271748, '11': 283143}
{'000': 195670, '001': 150041, '010': 128586, '011': 143162, '100': 150042, '101': 121706, '110': 143162, '111': 59981}
{'0000': 116966, '0001': 78703, '0010': 77485, '0011': 72556, '0100': 80398, '0101': 48188, '0110': 92419, '0111': 50743, '1000': 78704, '1001': 71338, '1010': 51100, '1011': 70606, '1100': 69644, '1101': 73518, '1110': 50743, '1111': 9238}
加密后:
{'0': 546438, '1': 546042}
{'00': 273698, '01': 272740, '10': 272740, '11': 273301}
{'000': 137077, '001': 136621, '010': 136457, '011': 136282, '100': 136621, '101': 136119, '110': 136282, '111': 137019}
{'0000': 68835, '0001': 68242, '0010': 68218, '0011': 68403, '0100': 68429, '0101': 68028, '0110': 68024, '0111': 68258, '1000': 68242, '1001': 68379, '1010': 68239, '1011': 67879, '1100': 68192, '1101': 68090, '1110': 68258, '1111': 68761}
PS D:\MyCourses\21H1-21H2-CryptographyNetworkSecurity\BigProjects\CH4-AES>
```

图 2.1 加密前后 0/1 位频度统计对比

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。

第三章 算法详细与结论

第一节 用户交互说明

一、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。

二、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。

三、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。

第二节 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>

一、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 ($S^{-1}(a) = a$)。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 $[S \text{ 盒}] \times [\text{行移位} \& \text{列混淆用矩阵}]$ ，得出 T 盒。

4. 生成逆 T 盒

与 T 盒类似，计算 $[\text{逆 S 盒}] \times [\text{逆行移位} \& \text{逆列混淆用矩阵}]$ ，得出逆 T 盒。

5. 生成轮密钥

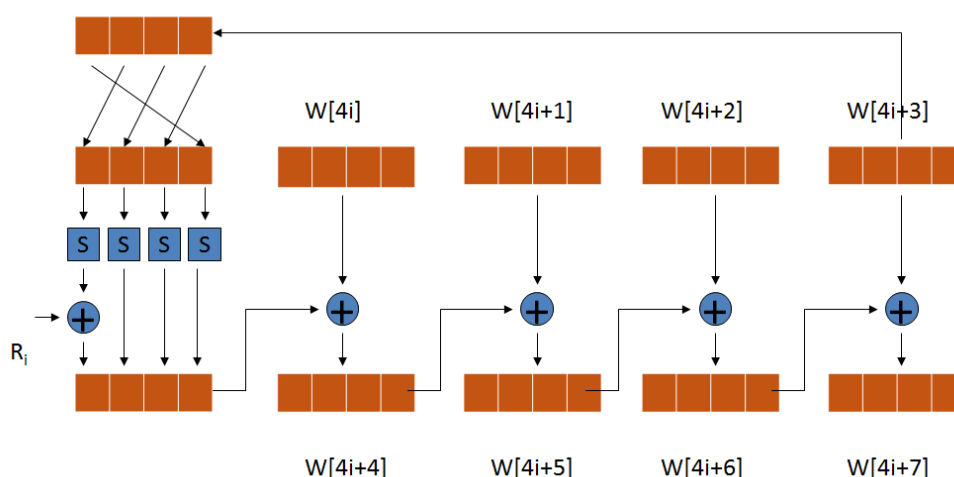


图 3.1 轮密钥扩展

输入 4×4 的密钥矩阵，扩展至 4×44 。

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。

二、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

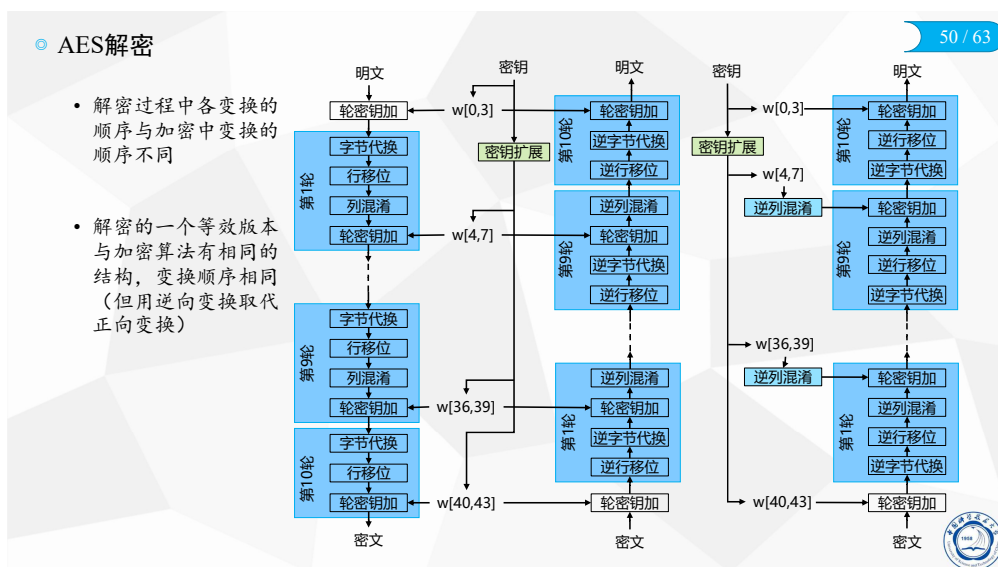


图 3.2 课件截图

数简化为四次查表和四次异或操作。

三、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。

四、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 $w_0[40,43]$
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 $w_0[0,3]$

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。

第三节 结论说明

一、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。

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>.