SHANDONG UNIVERSITY

密码工程第四次实验报告

姓名: 谢子洋
学院: 网络空间安全学院 (研究院)
专业: 网络空间安全
学号:202100460116

2023 年 11 月 28 日

目录

1 实验原理

1.1 侧信道攻击

1.2 差分功耗分析:DPA

1) 寻找中间值
根据算法找到一个中间值 z=f(x,k). 其中 x 为明文,k 为分段密钥, 要求足够短可以承受穷举的代价.

2) 采集功耗
运行 AES 运算 N 次, 采集每一次运行的功耗曲线, 每条曲线有 M 个功耗点. 功耗采集结果可表示为一个 $N \times M$ 大小的矩阵 T.

$$\mathbb{T} = \begin{pmatrix} t_{1,1} & \cdots & t_{1,M} \\ \vdots & \ddots & \vdots \\ t_{\mathbb{N},1} & \cdots & t_{\mathbb{N},M} \end{pmatrix}$$

3) 计算中间值及对应猜测功耗
穷举猜测分段密钥 k，并计算不同明文对应的中间结果. 得到一个 $M \cdot |k|$ 的中间值矩阵 Z. 根据中间值矩阵中元素的最高 (最低) 比特计算对应猜测功耗矩阵 H.

$$\mathbb{Z} = \begin{pmatrix} z_{1,1} & \cdots & z_{1,|k|} \\ \vdots & \ddots & \vdots \\ z_{\mathbb{N},1} & \cdots & z_{\mathbb{N},|k|} \end{pmatrix} \xrightarrow{\text{选择 1bit}} \mathbb{H} = \begin{pmatrix} h_{1,1} & \cdots & h_{1,|k|} \\ \vdots & \ddots & \vdots \\ h_{\mathbb{N},1} & \cdots & h_{\mathbb{N},|k|} \end{pmatrix}$$

4) 曲线分类并进行比较.
对于矩阵 H 的第 i 列,$1 \le i \le |k|$. 根据该列的值分别取 0 或 1 将功耗矩阵 T 对应位置曲线进行分类, 划分为两类曲线并进行平均. 此时我们得到两个向量 $\overline{h_1}$ 和 $\overline{h_2}$, 评估两向量间差异度可使用距离的概念.

1) 绝对距离 (manhattan distance)
$L1 = \sum |x_i - y_i|$
2) 欧几里得距离 (Euclidean distance)
$L2 = [\sum (x_i - y_i)^2\ ]^{\frac{1}{2}}$
3) 闵可夫斯基距离 (Minkowski distance)
$L_m = [\sum (x_i - y_i)^m\ ]^{\frac{1}{m}}$
4) 切比雪夫距离 (Chebyshev distance)
$L_\infty = max\{|x_i - y_i|\ for\ i\}$
5) 马氏距离
$=\sqrt{(x_i - y_i)\sum^{-1}(x_i - y_i)}$

经计算可得到两向量的距离 $d_i, 1 \le i \le |k|$. 重复循环计算后得到共 $|k|$ 个向量距离, 代表了 $|k|$ 种密钥猜想, 其中的最大距离对应了最有可能的一种密钥猜想.

1.3 相关性功耗分析:CPA

1) 寻找中间值

根据算法找到一个中间值 z=f(x,k). 其中 x 为明文,k 为分段密钥, 要求足够短可以承受穷举的代价.

2) 采集功耗

运行 AES 运算 N 次, 采集每一次运行的功耗曲线, 每条曲线有 M 个功耗点. 功耗采集结果可表示为一个 $N \times M$ 大小的矩阵 T.

$$\mathbb{T} = \begin{pmatrix} t_{1,1} & \cdots & t_{1,M} \\ \vdots & \ddots & \vdots \\ t_{\mathbb{N},1} & \cdots & t_{\mathbb{N},M} \end{pmatrix}$$

3) 计算中间值及对应猜测功耗

穷举猜测分段密钥 k，并计算不同明文对应的中间结果. 得到一个 $M \cdot |k|$ 的中间值矩阵 Z.

根据中间值矩阵中元素的汉明重量/汉明距离计算对应猜测功耗矩阵 H.

$$\mathbb{Z} = \begin{pmatrix} z_{1,1} & \cdots & z_{1,|k|} \\ \vdots & \ddots & \vdots \\ z_{\mathbb{N},1} & \cdots & z_{\mathbb{N},|k|} \end{pmatrix} \xrightarrow{\text{汉明重量/汉明距离}} \mathbb{H} = \begin{pmatrix} h_{1,1} & \cdots & h_{1,|k|} \\ \vdots & \ddots & \vdots \\ h_{\mathbb{N},1} & \cdots & h_{\mathbb{N},|k|} \end{pmatrix}$$

4) H 和 T 按列两两计算相关性.

矩阵 H 和矩阵 T 各取出一列分别为 $h_r \ for \ 1 \le r \le |k|$ 和 $t_s \ for \ 1 \le s \le M$.

计算两者相关度:

$$\rho_{h_r,r_s} = \frac{cov(h_r, r_s)}{\sigma_{h_r}\sigma_{r_s}} = \frac{E[(h_r - \mu_{h_r})(r_s - \mu_{r_s})]}{\sigma_{h_r}\sigma_{r_s}}$$

循环计算后最终得到一个 $|k| \times M$ 大小的相关性矩阵 R.

$$\mathbb{R} = \begin{pmatrix} r_{1,1} & \cdots & r_{1,M} \\ \vdots & \ddots & \vdots \\ r_{|k|,1} & \cdots & r_{|k|,M} \end{pmatrix}$$

选择其中的最大值 $r_{u,v}$, 其代表密钥为 u 且第 v 个功耗点存在 S-box 输出的泄露.

2 实验过程

2.1 DPA 攻击第一轮轮密钥

1. 寻找中间值

   选择第一轮第一个 S 盒输出作为中间值.

2. 采集功耗

题目给出了 AES 算法全过程的功率采集曲线, 以离散采集点的形式体现, 我们选择第一次 S 盒运算时间段内的采集点作为功耗数据, 即 [3000,6000] 时间段内.

3. 计算中间值及猜测功耗
   根据 AES 算法原理计算所有明文和密钥可能性下的中间值 $z_{i,j}$, 形成中间值矩阵 Z.

$$z_{i,j} = Sbox(plainByte \oplus keyByte)$$

   选择中间值的最高或最低比特作为猜测功耗矩阵 H 的元素值 $h_{i,j}$. 在比特位置的具体选择上, 本文选择同时测试两种位置并比较结果差异.

4. 曲线分类进行比较
   根据 H 矩阵一列中每一行的取值将曲线划分为两类, 并计算两曲线的" 距离". 本文实现了多种距离算法, 包括欧几里得距离、曼哈顿距离及切比雪夫距离, 并计算出使用不同" 距离" 算法得到的差异结果值. 发现不同距离算法会导致结果发生较大变化.

2.2 CPA 攻击最后一轮轮密钥

1. 寻找中间值
   选择最后一轮 S 盒的输入作为中间值. 攻击从密文倒推回中间值, 选择 Sbox 输出可以使差异很小的密钥在过 Sbox 后差异很大, 对应功耗差别大. 而若是选择 Sbox 的输出作为中间值则难以区分相近的密钥.

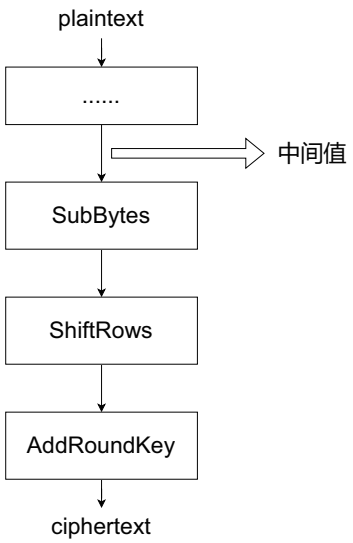

图 1: 中间值选择

2. 采集功耗
   题目给出了 AES 全过程的功耗数据:

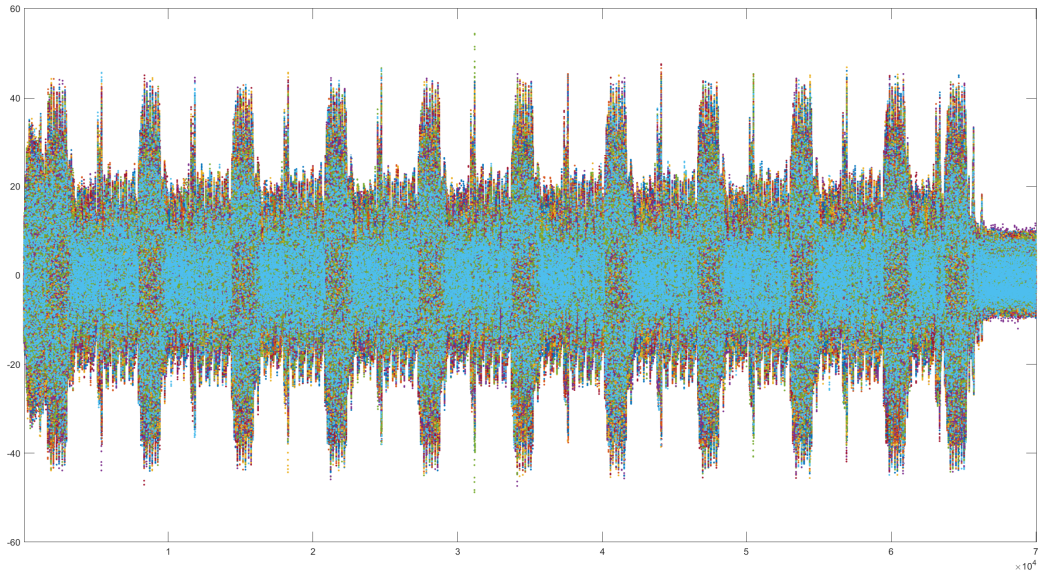

图 2: 功耗散点图

因为算法计算不同时间不同密钥猜想下功耗之间的相关性, 在理论上只要功耗泄露的时间点在选定的时间范围内即可做出正确解, 因此时间区间选择可以适当扩大, 甚至是直接用全部数据进行计算.

3. 计算中间值及猜测功耗
   根据 AES 算法原理计算所有明文和密钥可能性下的中间值 $z_{i,j}$, 可根据题目给出的密文由后向前倒推中间值. AddRoundKey 逆运算令密文分块异或密钥,ShiftRows 移位对单独字节分块无影响,SubBytes 逆运算需要过逆 S 盒.

$$z_{i,j} = invSbox(cipherByte \oplus keyByte)$$

   经多次计算后形成中间值矩阵 Z. 计算中间值 $z_{i,j}$ 的汉明重量作为猜测功耗矩阵 H 的元素值 $h_{i,j}$.

4. 计算猜测功耗和实际功耗相关性
   目前我们已有猜测功耗矩阵 H 和实际功耗矩阵 T, 按照列依次求出列与列之间的相关系数

$$\rho_{h_r,r_s} = \frac{cov(h_r, r_s)}{\sigma_{h_r}\sigma_{r_s}} = \frac{E[(h_r - \mu_{h_r})(r_s - \mu_{r_s})]}{\sigma_{h_r}\sigma_{r_s}}$$

   得到相关性矩阵 R.

   找出相关性矩阵 R 中最大值 $r_{u,v}$, 该值代表了密钥取值为 u 且在功耗点 v 处进行 S-box 运算的可能性最大. 我们可选择该值 u 作为密钥的最佳取值.

3 实验结果

3.1 DPA

DPA 攻击使用不同的参数会得到不同的结果, 本问测试了选择不同位置 bit 和使用不同区分度算法的结果及差异. 具体结果如下表所示.

表 1: DPA 测试实例

bit 位置	区分度算法	结果	错误字节数	正确率
最低位	euclidean 距离	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 a2 cf 4f 3c	1	93.75%
最低位	manhattan 距离	2b 7e 15 16 6a ae d2 a6 59 f7 15 88 6c 3d 4f 3c	4	75%
最低位	chebyshev 距离	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c	0	100%
次低位	euclidean 距离	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c	0	100%
次高位	euclidean 距离	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c	0	100%
最高位	euclidean 距离	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c	0	100%
最高位	manhattan 距离	0f 7e 15 16 28 ae d2 a6 30 f7 15 88 da cf 4f 3c	3	81.25%
最高位	chebyshev 距离	2b 7e 15 16 28 ae d2 84 ab f7 15 88 09 cf 4f 3c	1	93.75%

可观察到, 选择最低位 bit 并使用 chebushev 距离和选择最高位并使用 eucilidean 距离所得结果最为精准.

3.2 CPA

本文测试了选取不同中间值, 不同时间曲线下的多种结果.

当中间值选取最后一个 S-box 的输入值:

表 2: CPA 测试实例

区间	结果
::	d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
50000:70000	d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
60000:65000	d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
60000:62000	d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
60000:61000	d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6
60000:60500	d0 14 f9 e9 c9 ee 34 89 e1 c6 0c c8 0e 63 0c de

当中间值选取最后一个 S-box 的输出值:

表 3: CPA 测试实例

区间	结果
::	00 eb 06 57 36 11 25 89 1e ff 0c 37 49 00 f3 59
30000:70000	00 eb 06 57 36 11 25 89 1e ff 0c 37 49 00 f3 59
40000:70000	00 eb 06 57 36 11 25 89 1e ff 0c 37 49 00 f3 59
40000:60000	4c 76 0e ac 55 59 46 14 00 ad 6f 7a 2c 02 6f 5f
60000:70000	00 eb 06 57 36 11 25 89 1e ff 0c 37 49 00 f3 59

可见中间值选择 S 盒输入效果更好. 这是因为对 AES 最后一轮轮密钥的攻击是由密文开始从后向前推导中间值, 选择 S 盒输入为中间值会使差异很小的密钥在过逆 Sbox 后差异很大, 对应功耗差别大, 更容易找出正确密钥.

在功耗曲线时间区间的选择上, 选取区间可以任意大, 只要包括与选取中间值相关的功耗点即可. 这是因为 CPA 在原理上是从所有可能密钥中选择在某个时间点上相关性最大的密钥, 理论上该时间点就在中间值时间点前后, 只要包括该时间点即可算出正确结果.

参考文献

A Code

A.1 DPA 攻击第一轮轮密钥.

```python
import numpy as np
from tqdm import trange
sbox = [
    # 0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f
    0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76, ... # 0
    0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0, ... # 1
    0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15, ... # 2
    0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75, ... # 3
    0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84, ... # 4
    0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf, ... # 5
    0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8, ... # 6
    0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2, ... # 7
    0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73, ... # 8
    0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb, ... # 9
    0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79, ... # a
    0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08, ... # b
    0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a, ... # c
    0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e, ... # d
    0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf, ... # e
    0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16  ... # f
]

def aes_internal(inputdata, key):
    return sbox[inputdata ^ key]
def selectBit(num,isLow):
    #return num%2 if isLow else num>>7 #正常
    #return (num>>6)&1
    return (num>>1)%2
def euclidean(x, y):
    return np.sqrt(np.sum((x - y)**2))
def manhattan(x, y):
    return np.sum(np.abs(x - y))
def chebyshev(x, y):
    return np.max(np.abs(x - y))

def DPA(trace,textin_array):
```

```
    cpa_correlation=[0]*16
    bestguess=[0]*16
    for byte in trange(0, 16):
        maxcpa = [0] * 256
        for kguess in range(0, 256): #猜测轮密钥的 1Byte
            #每次循环计算一列(一种猜测密钥)
            isLow=0
            Hcol=[[selectBit(aes_internal(textin[byte],kguess),isLow) for textin in ...
                textin_array]]
            Hcol = np.array(Hcol).transpose()#1000*1                              ...
                #3000*1
            #根据H进行分类
            pos0=np.where(Hcol==0)
            pos1=np.where(Hcol==1)
            #计算平均
            aver0=np.mean(trace[pos0[0],:],axis=0)
            aver1=np.mean(trace[pos1[0],:],axis=0)
            #计算区别度
            maxcpa[kguess]=euclidean(aver0,aver1)
        bestguess[byte] = np.argmax(maxcpa)  #轮密钥的一个Byte
        cpa_correlation[byte] = max(maxcpa)  #区分度

    print("Best Key Guess: ", end="")
    for b in bestguess: print(",0x%02x " % b, end="")
    print("\n", cpa_correlation)

def trans_txt(filename):
    #npy文件转化为txt文件
    edge = np.load(filename+'.npy',allow_pickle = True)  #读取npy文件
    f = open(filename+'.txt','w')   #打开要被写入的txt文件
    print(f'{edge.shape[0]} {edge.shape[1]}')
    for i in range(0,edge.shape[0]):
        for j in range(edge.shape[1]):
            if j == edge.shape[1]-1:
                f.write(str(edge[i,j]))
            else:
                f.write(str(edge[i,j])+'\t')
        f.write('\r\n')
    f.close()

def testDPA():
    trace=np.load("AES_tracesPart0.npy")          #1000*70000 原始采集数据
    trace_sbox_position=trace[:,3000:6000]        #1000*3000  sbox3000个点
    textin_array=np.load("AES_textinPart0.npy")  #1000*16    16Byte明文
    DPA(trace_sbox_position,textin_array)
testDPA()
```

A.2 CPA 攻击最后一轮轮密钥.

```
import numpy as np
from tqdm import trange
inv_sbox=[
  0x52 ,0x09 ,0x6a ,0xd5 ,0x30 ,0x36 ,0xa5 ,0x38 ,0xbf ,0x40 ,0xa3 ,0x9e ,0x81 ,0xf3 ...
      ,0xd7 ,0xfb,
  0x7c ,0xe3 ,0x39 ,0x82 ,0x9b ,0x2f ,0xff ,0x87 ,0x34 ,0x8e ,0x43 ,0x44 ,0xc4 ,0xde ...
      ,0xe9 ,0xcb,
```

```
    0x54 ,0x7b ,0x94 ,0x32 ,0xa6 ,0xc2 ,0x23 ,0x3d ,0xee ,0x4c ,0x95 ,0x0b ,0x42 ,0xfa ...
        ,0xc3 ,0x4e,
    0x08 ,0x2e ,0xa1 ,0x66 ,0x28 ,0xd9 ,0x24 ,0xb2 ,0x76 ,0x5b ,0xa2 ,0x49 ,0x6d ,0x8b ...
        ,0xd1 ,0x25,
    0x72 ,0xf8 ,0xf6 ,0x64 ,0x86 ,0x68 ,0x98 ,0x16 ,0xd4 ,0xa4 ,0x5c ,0xcc ,0x5d ,0x65 ...
        ,0xb6 ,0x92,
    0x6c ,0x70 ,0x48 ,0x50 ,0xfd ,0xed ,0xb9 ,0xda ,0x5e ,0x15 ,0x46 ,0x57 ,0xa7 ,0x8d ...
        ,0x9d ,0x84,
    0x90 ,0xd8 ,0xab ,0x00 ,0x8c ,0xbc ,0xd3 ,0x0a ,0xf7 ,0xe4 ,0x58 ,0x05 ,0xb8 ,0xb3 ...
        ,0x45 ,0x06,
    0xd0 ,0x2c ,0x1e ,0x8f ,0xca ,0x3f ,0x0f ,0x02 ,0xc1 ,0xaf ,0xbd ,0x03 ,0x01 ,0x13 ...
        ,0x8a ,0x6b,
    0x3a ,0x91 ,0x11 ,0x41 ,0x4f ,0x67 ,0xdc ,0xea ,0x97 ,0xf2 ,0xcf ,0xce ,0xf0 ,0xb4 ...
        ,0xe6 ,0x73,
    0x96 ,0xac ,0x74 ,0x22 ,0xe7 ,0xad ,0x35 ,0x85 ,0xe2 ,0xf9 ,0x37 ,0xe8 ,0x1c ,0x75 ...
        ,0xdf ,0x6e,
    0x47 ,0xf1 ,0x1a ,0x71 ,0x1d ,0x29 ,0xc5 ,0x89 ,0x6f ,0xb7 ,0x62 ,0x0e ,0xaa ,0x18 ...
        ,0xbe ,0x1b,
    0xfc ,0x56 ,0x3e ,0x4b ,0xc6 ,0xd2 ,0x79 ,0x20 ,0x9a ,0xdb ,0xc0 ,0xfe ,0x78 ,0xcd ...
        ,0x5a ,0xf4,
    0x1f ,0xdd ,0xa8 ,0x33 ,0x88 ,0x07 ,0xc7 ,0x31 ,0xb1 ,0x12 ,0x10 ,0x59 ,0x27 ,0x80 ...
        ,0xec ,0x5f,
    0x60 ,0x51 ,0x7f ,0xa9 ,0x19 ,0xb5 ,0x4a ,0x0d ,0x2d ,0xe5 ,0x7a ,0x9f ,0x93 ,0xc9 ...
        ,0x9c ,0xef,
    0xa0 ,0xe0 ,0x3b ,0x4d ,0xae ,0x2a ,0xf5 ,0xb0 ,0xc8 ,0xeb ,0xbb ,0x3c ,0x83 ,0x53 ...
        ,0x99 ,0x61,
    0x17 ,0x2b ,0x04 ,0x7e ,0xba ,0x77 ,0xd6 ,0x26 ,0xe1 ,0x69 ,0x14 ,0x63 ,0x55 ,0x21 ...
        ,0x0c ,0x7d,
]
def mean(X):
    return np.mean(X,axis=0)
def std_dev(X, X_mean):
    return np.sqrt(np.sum((X-X_mean)**2,axis=0))
def cov(X, X_mean, Y, Y_mean):
    return np.sum((X-X_mean)*(Y-Y_mean),axis=0)

def aes_internal(data, key):
    return inv_sbox[data^key]
    #return data^key
HW = [bin(n).count("1") for n in range(0, 256)]


def CPA(trace,textout_array):
    trace_mean=np.average(trace,axis=0)
    trace_omega= np.sqrt(np.sum((trace - trace_mean)**2, axis=0))
    cpa_correlation=[0]*16
    bestguess=[0]*16
    for byte in trange(0, 16):

        maxcpa = [0] * 256
        for kguess in range(0, 256): #猜测轮密钥的 1Byte
            #每次循环计算一列(一种猜测密钥)
            hws = np.array([[HW[aes_internal(textout[byte],kguess)] for textout in ...
                textout_array]]).transpose()#1000*1
            hws_mean=mean(hws)
            o_hws=std_dev(hws,hws_mean)
            correlation=cov(trace,trace_mean,hws,hws_mean)
            cpaoutput = correlation/(o_hws*trace_omega) ...
                                                #3000*1
```

```
            maxcpa[kguess] = max(abs(cpaoutput))

        bestguess[byte] = np.argmax(maxcpa)
        cpa_correlation[byte] = max(maxcpa)

    print("Best Key Guess: ", end="")
    for b in bestguess: print("%02x " % b, end="")
    print("\n", cpa_correlation)

trace=np.load("AES_tracesPart0.npy")          #1000*70000 原始采集数据
trace_sbox_position=trace[:,60000:60500]
textout_array=np.load("AES_textoutPart0.npy")  #1000*16    16Byte明文
CPA(trace_sbox_position,textout_array)
```