山东大学
SHANDONG UNIVERSITY

AES 的测侧信道攻击

密码工程

2023 年 11 月 29 日

李 昕 202100460065

21 级密码 2 班

摘　要

侧信道攻击，即攻击者利用密码设备实际工作时所释放的侧信道信息，恢复敏感安全参数或者密钥信息，展开攻击的过程。依据所利用侧信道信息得类型不同，可以分为不同得分析方法：1. 能量分析（利用密码芯片在实际运行钟产生得能量消耗信息）2. 电磁分析（采集密码芯片运行期间得电磁辐射信息）3. 计时攻击（利用密码芯片执行密码算法得运行时间信息）4. 声音攻击（收集密码芯片计算时的声波信息）5. 故障分析 6. 碰撞分析

在本次实验中，分析现有 AES 的 CPA 攻击代码，编写 AES 的 DPA 攻击代码并验证正确性，并实现 CPA 攻击最后一轮轮密钥

关键词：AES　侧信道攻击

目录

1 **实验要求** 1

2 **符号标记说明** 1

3 **实验准备** 1

3.1 相关性功耗分析 (CPA) 原理 1

3.1.1 汉明重量 1

3.1.2 CPA 攻击方法 1

3.2 差分功耗分析 (DPA) 原理 2

4 **实验过程** 3

4.1 分析第一轮 CPA 攻击代码 3

4.2 实现第一轮 DPA 攻击代码 6

4.3 实现最后一轮 CPA 攻击代码 7

1 实验要求

问题一 分析提供的第一轮密钥 CPA 攻击，实现 DPA 攻击。

问题二 实现 CPA 攻击最后一轮轮密钥

2 符号标记说明

符号	解释
$\oplus$	异或
S-box	S 盒
$f(d\,k)$	密码学中间运算函数
T	功耗矩阵
H	汉明距离矩阵
R	相关性矩阵

3 实验准备

3.1 相关性功耗分析 (CPA) 原理

CPA 攻击主要采用汉明重量模型。计算汉明重量与对应能量迹之间的相关系数，若相关系数越大，则说明他们之间的相关性越强，若数据中某一猜测密钥对应的相关系数相比于其他相关系数要大的多，就可以说明这一猜测密钥是正确的

3.1.1 汉明重量

汉明重量可以表示一个二进制字符串中 1 的个数。已经有论文通过实验证明了输出结果的汉明重量与能量消耗之间有明确的关系，能量消耗随着汉明重量的增大而增大，而且有较为明确的界限，经过计算，它们的相关系数甚至能够达到 0.9919[4]。这也说明 CPA 使用汉明重量模型是合理的。

3.1.2 CPA 攻击方法

在 CPA 分析中，首先应选择所执行算法的某个中间值，在本实验中，该中间值设置为函数 f（d，k），这里的 d 一般是明文或者密文，而 k 是密钥的一部分，即 s 盒和

密钥异或的部分。

在每次设备运行时，都记录下一个能量迹，（这里的能量迹是指电压差或温度等侧信道信息），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，采集每一次运行的功耗曲线，每条曲线有 M 个功耗点（每一行为条功耗曲线，每一列对应不同时刻的功耗点）如下矩阵：

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

Figure 1: 功耗矩阵

接下来穷举猜测分段密钥 k，并计算猜测的中间结果，生成猜测矩阵 Z, 不同行对应了不同的明文, 不同列对应了分段密钥 k 的不同的猜测值，分段密钥 k 的值空间大小记为 |k|。在本实验中，|k| = 256。对于生成的 Z 矩阵，利用汉明重量（即功耗模型）将猜测的 $z_{i,i}$ 转成猜测的功耗矩阵 H 中的 $h_{i,i}$，将 H 和 T 按列两两计算相关性，形成相关性矩阵 R，矩阵 R 中绝对值最大的项提示正确的密钥。

3.2 差分功耗分析 (DPA) 原理

DPA 可以看成 CPA 的一个特例：功耗模型只输出一个比特，两个方法区别在于比较的过程，DPA 直接根据猜测功耗值（0 或者 1）对曲线分类，然后平均看差异，而 CPA 则依据模型计算并猜测功耗值和真实功耗值之间的相关性，所以当功耗模型比较准确的时候，CPA 的效果好于 DPA。

DPA 分析开始时与 CPA 相似，同样需要功率曲线组成的功耗矩阵 T（如图一），也生成对应的猜测矩阵 Z, 不同行对应了不同的明文, 不同列对应了分段密钥 k 的不同的猜测值，用功耗模型将猜测的中间结果矩阵转成猜测的功耗矩阵 H，但是此时模型不再使用汉明重量，而是使用选取最高（最低）比特的方法生成矩阵 H:

$$H = \begin{pmatrix} h_{1,1} & \cdots & h_{1,|k|} \\ \vdots & \ddots & \vdots \\ h_{N,1} & \cdots & h_{N,|k|} \end{pmatrix}$$

Figure 2: H 矩阵

对于每一列的 H（每一列对应一个密钥猜测），根据 h 的值（0 或者 1）对曲线集合分类，0 分成一类，1 分成一类，对每一类的曲线集合进行平均，得到两个平均曲线，比较两个平均曲线，平均曲线区别最大的列对应正确的密钥猜测

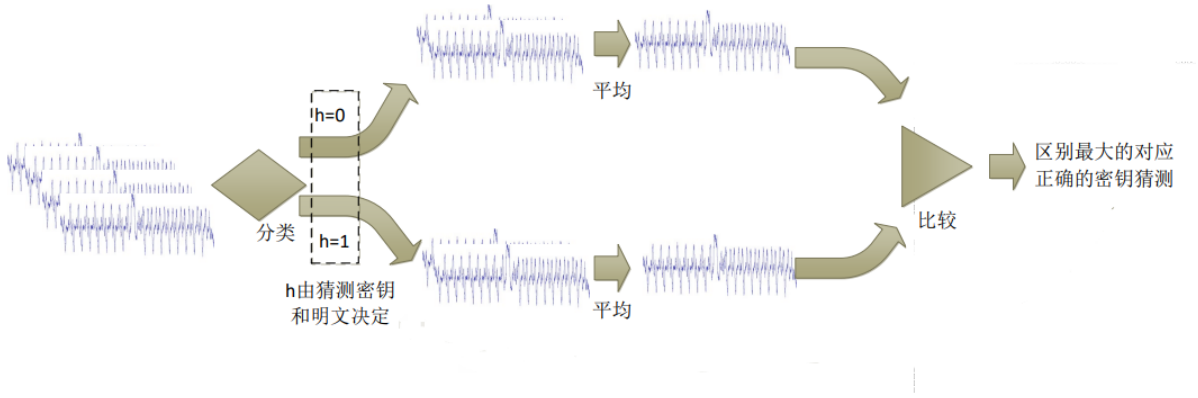

Figure 3: 密钥猜测

4 实验过程

4.1 分析第一轮 CPA 攻击代码

为了理解侧信道攻击的代码实现，首先分析老师提供的攻击 AES 第一轮加密的 CPA 攻击代码，其主要的函数是 CPA()，它接受输入曲线和输入字节数据，然后通过计算计算曲线顶点与输入字节之间的协方差（相关性）来猜测密钥。

具体步骤如下：

1. 遍历每个字节的密钥猜测值。

2. 根据密钥猜测值计算每个输入字节的汉明重量，即输入字节经过 S 盒变换后二进制表示中 1 的个数。

3. 计算曲线顶点的均值和标准差。

4. 计算曲线顶点与输入字节之间的协方差。

5. 计算 CPA 相关性。

6. 找到最大相关性对应的密钥猜测值。

7. 输出最终的密钥猜测结果和相关性。

```python
import numpy as np
from tqdm import trange

# S盒，用于AES加密中的替代字节操作
sbox = [
    ...
]

def aes_internal(inputdata, key):
    return sbox[inputdata ^ key]

# 计算每个数的汉明重量（二进制表示中1的个数）
HW = [bin(n).count("1") for n in range(0, 256)]

# 计算数组的均值
def mean(X):
    return np.mean(X, axis=0)

# 计算数组的标准差
def std_dev(X, X_bar):
    return np.sqrt(np.sum((X - X_bar) ** 2, axis=0))

# 计算数组的协方差
def cov(X, X_bar, Y, Y_bar):
    return np.sum((X - X_bar) * (Y - Y_bar), axis=0)

# 将曲线顶点对应的子字节通过相关性分析还原出密钥
def CPA(trace, textin_array):
    # 计算曲线顶点的均值和标准差
    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
```

```
        # 遍历所有密钥猜测
        for kguess in range(0, 256):
            # 根据密钥猜测计算每个输入字节的汉明重量
            hws = np.array([[HW[aes_internal(textin[byte], kguess)] for textin in textin_array]]).
        transpose()
            hws_bar = mean(hws)
            o_hws = std_dev(hws, hws_bar)
            # 计算曲线顶点与输入字节之间的协方差
            correlation = cov(trace, trace_mean, hws, hws_bar)
            # 计算CPA相关性
            cpaoutput = correlation / (o_hws * trace_omega)
            maxcpa[kguess] = max(abs(cpaoutput))

        # 找到最大相关性对应的密钥猜测值
        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)


if __name__ == '__main__':
    # 加载曲线和输入字节数据
    trace = np.load("AES_tracesPart0.npy")
    trace_sbox_position = trace[:, 3000:6000]
    textin_array = np.load("AES_textinPart0.npy")
    # 使用相关性分析还原密钥
    CPA(trace_sbox_position, textin_array)
```

运行代码可得第一轮密钥及相关值：

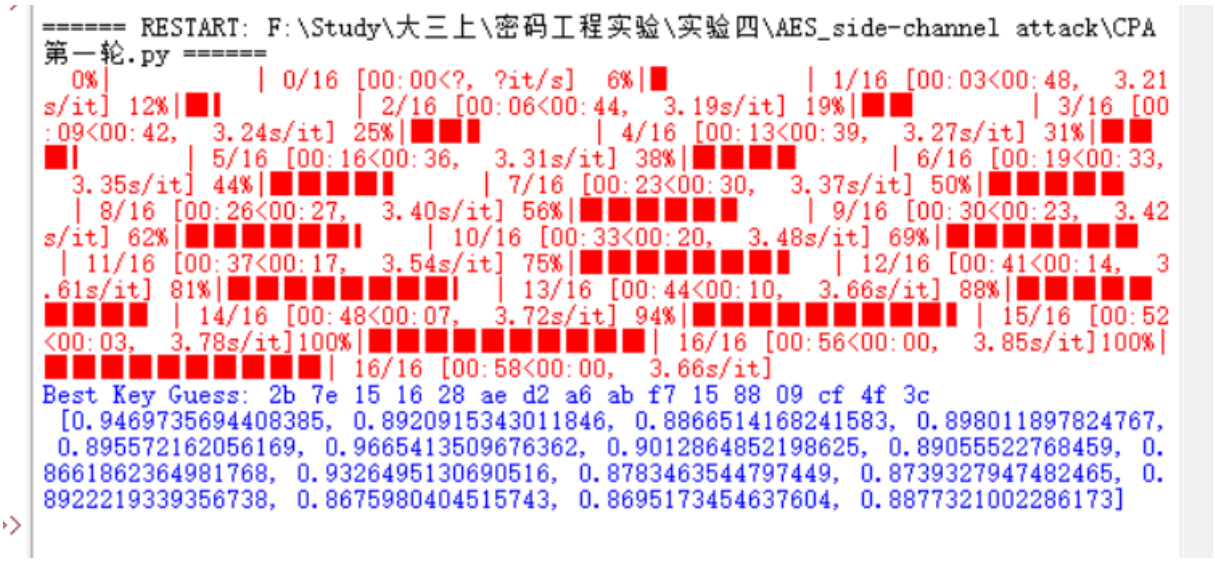

Figure 4: CPA 解密第一轮

4.2 实现第一轮 DPA 攻击代码

为了验证编写的代码结果正确性，利用第一轮轮密钥计算 2-10 轮轮密钥：

K00: 2B7E151628AED2A6ABF7158809CF4F3C

K01: A0FAFE1788542CB123A339392A6C7605

K02: F2C295F27A96B9435935807A7359F67F

K03: 3D80477D4716FE3E1E237E446D7A883B

K04: EF44A541A8525B7FB671253BDB0BAD00

K05: D4D1C6F87C839D87CAF2B8BC11F915BC

K06: 6D88A37A110B3EFDDBF98641CA0093FD

K07: 4E54F70E5F5FC9F384A64FB24EA6DC4F

K08: EAD27321B58DBAD2312BF5607F8D292F

K09: AC7766F319FADC2128D12941575C006E

K10: D014F9A8C9EE2589E13F0CC8B6630CA6

在老师提供的范例代码基础上，实现 DPA 方法，在代码中，对于每一列的 H（每一列对应一个密钥猜测），根据 h 的值（0 或者 1）对曲线集合分类，0 分成一类，1 分成一类，对每一类的曲线集合进行平均，得到两个平均曲线，比较两个平均曲线，最大的列对应正确的密钥猜测。

实现代码如下：

```
1 def DPA(trace,textin_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):
        m_array = [0] * 256
        for kguess in range(256):
            type1 = [ ]
            type0 = [ ]
            for line,textin in zip(trace,textin_array):
                if (aes_internal(textin[byte],kguess) & 0x80) == 0x80:
                    type1.append(line)
                else:
                    type0.append(line)
            type1_bar = mean(type1)
            type0_bar = mean(type0)
            mse = np.mean(np.square(type1_bar - type0_bar))
            m_array[kguess] = mse
        bestguess[byte] = np.argmax(m_array)
    print("Best Key Guess: ")
    for b in bestguess: print("%02x " %b)
```

运行结果如下：

```
>>>
====== RESTART: F:/Study/大三上/密码工程实验/实验四/AES_side-channel attack/DPA第一轮.py ======
  0%|          | 0/16 [00:00<?, ?it/s]  6%|▌         | 1/16 [00:04<01:08,  4.55s/it] 12%|█▎        | 2/16 [00:09<01:04,  4.61s/it] 19%|█▉        | 3/16 [00:13<01:00,  4.62s/it] 25%|██▌       | 4/16 [00:18<00:55,  4.63s/it] 31%|███▏      | 5/16 [00:23<00:51,  4.68s/it] 38%|███▊      | 6/16 [00:27<00:47,  4.70s/it] 44%|████▍     | 7/16 [00:32<00:42,  4.75s/it] 50%|█████     | 8/16 [00:37<00:37,  4.73s/it] 56%|█████▋    | 9/16 [00:42<00:33,  4.74s/it] 62%|██████▎   | 10/16 [00:47<00:28,  4.79s/it] 69%|██████▉   | 11/16 [00:52<00:24,  4.82s/it] 75%|███████▌  | 12/16 [00:57<00:19,  4.87s/it] 81%|████████▏ | 13/16 [01:01<00:14,  4.88s/it] 88%|████████▊ | 14/16 [01:06<00:09,  4.92s/it] 94%|█████████▍| 15/16 [01:12<00:04,  4.99s/it]100%|██████████| 16/16 [01:17<00:00,  5.05s/it]100%|██████████| 16/16 [01:18<00:00,  4.93s/it]
Best Key Guess: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
```

Figure 5: DPA 解密第一轮

可以看到，得到的猜测密钥与 CPA 算法算出的结果相同。

4.3 实现最后一轮 CPA 攻击代码

CPA 攻击最后一轮与攻击第一轮方法基本相同，同样是利用相关性求解轮密钥，不过，需要注意的是，此时需要利用提供的密文和第十轮 S 盒曲线区间来求解，此外，还需要重构 aes_internal 函数，利用逆 S 盒，通过密文盒猜测密钥反推第十轮前的中间值

(inv_sbox[inputdata ⊕ key]). 通过观察测信道区间曲线，选择 60000 到 64500 部分，代码如下：

```
import numpy as np
from tqdm import trange
# 逆S盒，用于AES加密中的替代字节操作
inv_sbox = [
    # 行号0
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    # 行号1
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    # 行号2
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    # 行号3
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    # 行号4
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    # 行号5
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    # 行号6
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    # 行号7
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
    # 行号8
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    # 行号9
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    # 行号10
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    # 行号11
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    # 行号12
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    # 行号13
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    # 行号14
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    # 行号15
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
```

```
]

def aes_internal(inputdata, key):
    #return sbox[inputdata ^ key]
    return inv_sbox[inputdata ^ key]


# 计算每个数的汉明重量（二进制表示中1的个数）
HW = [bin(n).count("1") for n in range(0, 256)]

# 计算数组的均值
def mean(X):
    return np.mean(X, axis=0)

# 计算数组的标准差
def std_dev(X, X_bar):
    return np.sqrt(np.sum((X - X_bar) ** 2, axis=0))

# 计算数组的协方差
def cov(X, X_bar, Y, Y_bar):
    return np.sum((X - X_bar) * (Y - Y_bar), axis=0)

# 将曲线顶点对应的子字节通过相关性分析还原出密钥
def CPA(trace, textin_array):
    # 计算曲线顶点的均值和标准差
    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
        # 遍历所有密钥猜测
        for kguess in range(0, 256):
            # 根据密钥猜测计算每个输入字节的汉明重量
            hws = np.array([[HW[aes_internal(textin[byte], kguess)] for textin in textin_array]]).
        transpose()
            hws_bar = mean(hws)
            o_hws = std_dev(hws, hws_bar)
```

```
74            # 计算曲线顶点与输入字节之间的协方差
75            correlation = cov(trace, trace_mean, hws, hws_bar)
76            # 计算CPA相关性
77            cpaoutput = correlation / (o_hws * trace_omega)
78            maxcpa[kguess] = max(abs(cpaoutput))
79
80        # 找到最大相关性对应的密钥猜测值
81        bestguess[byte] = np.argmax(maxcpa)
82        cpa_correlation[byte] = max(maxcpa)
83
84    print("Best Key Guess: ", end="")
85    for b in bestguess:
86        print("%02x " % b, end="")
87    print("\n", cpa_correlation)
88
89
90 if __name__ == '__main__':
91    # 加载曲线和输入字节数据
92    trace = np.load("AES_tracesPart0.npy")
93    trace_sbox_position = trace[:, 60000:64500]
94    textin_array = np.load("AES_textoutPart0.npy")#随机密文
95    # 使用相关性分析还原密钥
96    CPA(trace_sbox_position, textin_array)
97    #print(trace_sbox_position)
```

运行得到了正确的第十轮密钥，说明攻击成功：

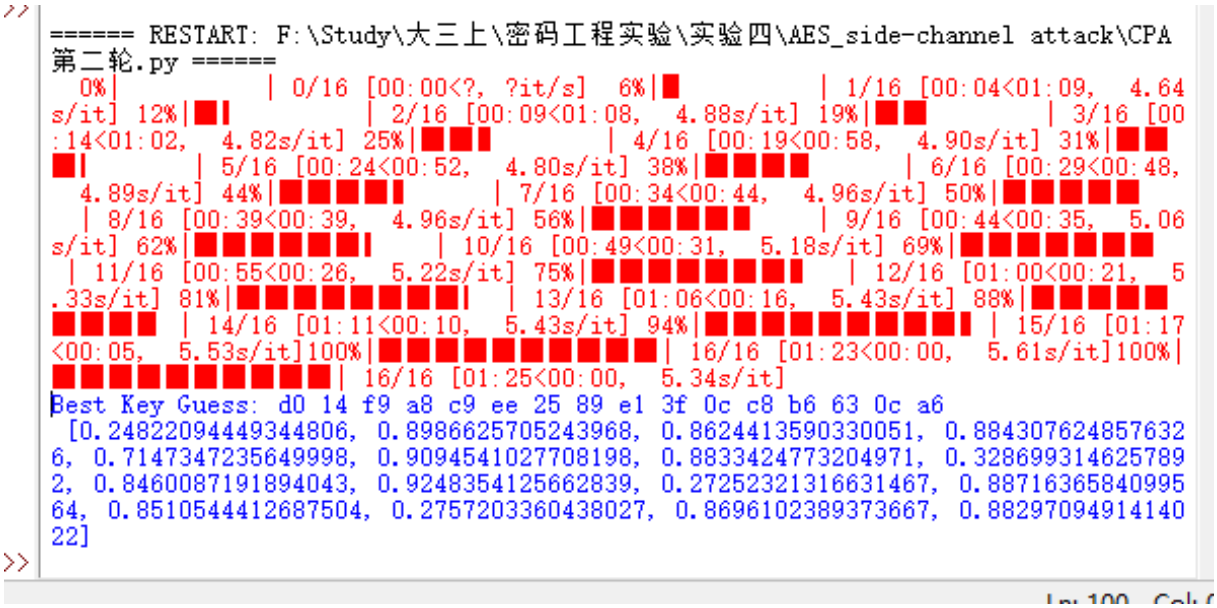

Figure 6: CPA 解密最后一轮