# 密码学课程实验报告

# Lab1

学院：网络空间安全学院

专业：信息安全、法学

学号：2113203

姓名：付政烨

班级：信安法班

## 摘要

本片报告主要探讨了 SPN（Substitution-Permutation Network）密码体制中的线性密码分析。线性密码分析通过发现 S 盒的线性特性来寻找明文、密文和密钥比特之间可能存在的概率线性关系。这种关系利用了某些比特子集的异或操作表现出的非随机分布特性。通过线性逼近，试图发现 S 盒的输入和输出之间的线性关系，这种关系在某些情况下比完全随机期望的更为准确。本文详细介绍了如何使用代换规则和置换规则对数据进行处理，并给出了 S 盒的线性逼近表示，以及如何利用堆积引理估计多轮密码的线性逼近的效果。实验的核心部分涉及了 SPN 的线性逼近，通过对多轮迭代进行逐步分析，得到了一个多轮的线性逼近公式。这种方法特别是在处理多轮的加密算法时非常有效，能够提供一个估计多个线性逼近组合时总体偏差的方法。本次实验中，我们结合了四轮的偏差值，利用堆积引理计算出总体偏差，从而得到了一个更为精确的线性逼近。此外，本研究还提供了一个实验过程，旨在复现课本中的 SPN 线性密码分析。通过利用前期实验的 SPN 对 16 比特数据进行加密，生成了大量的明文-密文对，从而进一步对 SPN 网络进行线性密码攻击。本研究为了对线性密码分析有一个更加深入的理解和实践，提供了理论和实验两方面的支持。

**关键字：SPN 网络；　线性密码分析；　概率线性关系**

# 目录

## 一、　实验目的

本实验旨在探索线性密码分析在 SPN 网络中的实际应用，通过对线性密码分析的基础理论、SPN 的代换与置换规则、S 盒的线性逼近方法以及多轮 SPN 的线性逼近分析的学习，试图模拟真实的攻击场景。利用已知的明文——密文对对 SPN 网络进行了线性密码攻击，从而有效地验证了线性密码分析的理论的可行性。

## 二、　实验原理

线性密码分析是通过分析 S 盒的线性特性，发现明文比特、密文比特和密钥比特之间可能存在的概率线性关系，即存在一个比特子集使得其中元素的异或表现出非随机的分布来进行分析的密码分析方法。同时线性密码分析一种已知明文攻击方法，已知 X 和 Y，确定 k 或 k 的部分比特。

### （一）　代换规则 $\pi_s(z)$ 与置换规则 $\pi_p(z)$

定义如下表1的代换/置换规则：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_s(z)$ | E | 4 | D | 1 | 2 | F | B | 8 | 3 | A | 6 | C | 5 | 9 | 0 | 7 |
| $\pi_p(z)$ | 1 | 5 | 9 | 13 | 2 | 6 | 10 | 14 | 3 | 7 | 11 | 15 | 4 | 8 | 12 | 16 |

表 1

代换规则的二进制表示（表2）：

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

表 2

密钥编排如下:

$$K = (k_1, ..., k_{32}) \text{ 对于} 1 \le r \le 5, k^r = k_{4r-3}, k_{4r-2}, ..., k_{4r+12}$$

## （二） 偏差与堆积引理

**定义 1：偏差**

对于随机变量 $X_i$，设其取值为 0 的概率为 $P(X_i = 0)$，则我们定义该随机变量的偏差 $\epsilon_i$ 为:

$$\epsilon_i = P(X_i = 0) - 0.5 \tag{1}$$

由此定义，我们可以得出:

$$P(X_i = 0) = \epsilon_i + 0.5 \tag{2}$$

**定义 2：堆积引理 (Piling-up Lemma)**

堆积引理是线性密码分析中的关键概念，特别在处理多轮的加密算法时显得尤为重要。此引理为我们提供了一种方法，用以估算多个线性逼近组合的总偏差。

设我们有一系列独立的随机变量 $X_1, X_2, \ldots, X_n$，其中每一个随机变量的取值为 0 或 1，且我们知道它们等于 0 的概率。堆积引理为我们提供了一种计算这些随机变量的异或（即模 2 加法）为 0 的概率的方法，其中 $\bigoplus$ 表示异或操作。

$$P\left(\bigoplus_{i=1}^{n} X_i = 0\right) = 2^{n-1}\prod_{i=1}^{n} P(X_i = 0) - 2^{n-2} \tag{3}$$

根据堆积引理，当我们有 $n$ 个独立的随机变量 $X_1, X_2, \ldots, X_n$ 的线性逼近时，其组合的总偏差 $\epsilon$ 由以下公式给出:

$$\epsilon = 2^{n-1}\prod_{i=1}^{n} \epsilon_i \tag{4}$$

这个公式提供了一种方法，即通过已知的单个线性逼近的偏差，计算这些逼近组合起来时的总偏差。这对于评估多轮密码的线性逼近攻击的效果特别有用。

## （三） S 盒线性逼近

S 盒的线性逼近主要是为了找到 S 盒的输入和输出之间的线性关系。这种关系在某些情况下比完全随机期望的更为准确。换句话说，线性逼近试图找到 S 盒的一个线性表达，这个表达在某些情况下比随机猜测更准确。

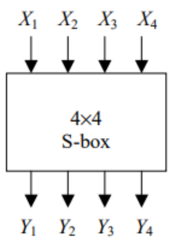


图 1: S-box Mapping

设输入 $X=(X_1,X_2,\ldots,X_m)$，输出为 $Y=(Y_1,Y_2,\ldots,Y_n)$，然后我们可以用形如：$X_1\oplus X_2\oplus\cdots\oplus X_i\oplus Y_1\oplus Y_2\oplus\cdots\oplus Y_j$ 的式子拟合输入与输出之间可能存在的线性关系。针对如图1所示的 S 盒，为了后面描述方便，将明文和密文比特的组合可以表示为（$a_i,b_i$ 表示 $x_i,y_i$ 是否参与运算）：

$$\bigoplus_{i=1}^{4}(a_i\cdot X_i)\oplus\bigoplus_{i=1}^{4}(b_i\cdot Y_i)$$

设 $N_L(a,b)$ 是满足 $\bigoplus_{i=1}^{4}(a_i\cdot X_i)\oplus\bigoplus_{i=1}^{4}(b_i\cdot Y_i)=0$ 的 8 元组 $(X_1,X_2,X_3,X_4,Y_1,Y_2,Y_3,Y_4)$ 对应取值的个数，其中 $(Y_1,Y_2,Y_3,Y_4)=\pi_s(X_1,X_2,X_3,X_4)$ 。例如：

$$X_1\oplus X_4\oplus Y_2\rightarrow N_L(a,b)=N_L(9,4)=8$$

说明由八组 $(x_1,x_2,x_3,x_4,y_1,y_2,y_3,y_4)$ 符合要求，这里 $a=9$ 是因为 $X_1\oplus X_4=(1001)_2=9$，$b=4$ 是因为 $Y_2=(0010)_2=4$。

进而随机变量的偏差可以具体表示为：

$$\epsilon(a,b)=\frac{N_L(a,b)}{16}-\frac{1}{2}$$

$16=2^4$ 代表 $(X_1,X_2,X_3,X_4)$ 可能出现的组合。同时，需要注意的是，偏差的绝对值越大，表示该情况越逼近线性关系。

根据上述描述，遍历 $a$ 从 0000 到 1111 和 $b$ 从 0000 到 1111 的全部 $16\times16$ 种情况，我们可以得到相应的表格（表3)。

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | -2 | -2 | 0 | 0 | -2 | 6 | 2 | 2 | 0 | 0 | 2 | 2 | 0 | 0 |
| 2 | 0 | 0 | -2 | -2 | 0 | 0 | -2 | -2 | 0 | 0 | 2 | 2 | 0 | 0 | -6 | 2 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | -6 | -2 | -2 | 2 | 2 | -2 | -2 |
| 4 | 0 | 2 | 0 | -2 | -2 | -4 | -2 | 0 | 0 | -2 | 0 | 2 | 2 | -4 | 2 | 0 |
| 5 | 0 | -2 | -2 | 0 | -2 | 0 | 4 | 2 | -2 | 0 | -4 | 2 | 0 | -2 | -2 | 0 |
| 6 | 0 | 2 | -2 | 4 | 2 | 0 | 0 | 2 | 0 | -2 | 2 | 4 | -2 | 0 | 0 | -2 |
| 7 | 0 | -2 | 0 | 2 | 2 | -4 | 2 | 0 | -2 | 0 | 2 | 0 | 4 | 2 | 0 | 2 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -2 | 2 | 2 | -2 | 2 | -2 | -2 | -6 |
| 9 | 0 | 0 | -2 | -2 | 0 | 0 | -2 | -2 | -4 | 0 | -2 | 2 | 0 | 4 | 2 | -2 |
| 10 | 0 | 4 | -2 | 2 | -4 | 0 | 2 | -2 | 2 | 2 | 0 | 0 | 2 | 2 | 0 | 0 |
| 11 | 0 | 4 | 0 | -4 | 4 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | -2 | 4 | -2 | -2 | 0 | 2 | 0 | 2 | 0 | 2 | 4 | 0 | 2 | 0 | -2 |
| 13 | 0 | 2 | 2 | 0 | -2 | 4 | 0 | 2 | -4 | -2 | 2 | 0 | 2 | 0 | 0 | 2 |
| 14 | 0 | 2 | 2 | 0 | -2 | -4 | 0 | 2 | -2 | 0 | 0 | -2 | -4 | 2 | -2 | 0 |
| 15 | 0 | -2 | -4 | -2 | -2 | 0 | 2 | 0 | 0 | -2 | 4 | -2 | -2 | 0 | 2 | 0 |

表 3

因此，在使用该表时，只需要找到异或操作种 X 和 Y 对应的 a（Input Sum）和 b（Output Sum)，就可以得到对应随机变量 $\bigoplus_{i=1}^{4}(a_i\cdot X_i)\oplus\bigoplus_{i=1}^{4}(b_i\cdot Y_i)=0$ 的 $(X_1,X_2,X_3,X_4)$ 的个数，这里为何等于 0 参照定义 $\bigoplus_{i=1}^{4}(a_i\cdot X_i)\oplus\bigoplus_{i=1}^{4}(b_i\cdot Y_i)=0$。

在本次实验中我们选择如下随机变量进行分析：

$$\epsilon_1 : X_1 \oplus X_3 \oplus X_4 \oplus Y_2$$

$$\epsilon_2 : X_2 \oplus Y_2 \oplus Y_4$$

$$\epsilon_3 : X_2 \oplus Y_2 \oplus Y_4$$

$$\epsilon_4 : X_2 \oplus Y_2 \oplus Y_4$$

通过查表 3 我们得到以上随机变量异或运算后结果为 0 的数量分别为：$12, 4, 4, 4$ (选择它们作为随机变量的原因是因为偏差的绝对值越大,越逼近线性关系),对应的偏差分别为:$1/4, -1/4, -1/4, -1/4$。然后运用前文给出的堆积引理:

$$\epsilon = 2^{n-1}\prod_{i=1}^{n}\epsilon_i$$

带入 $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$ 得到:

$$\epsilon = \epsilon_1 \times \epsilon_2 \times \epsilon_3 \times \epsilon_4 = 2^3 \times (\frac{1}{4}) \times (-\frac{1}{4})^3 = -\frac{1}{32}$$

## （四）　SPN 的线性逼近

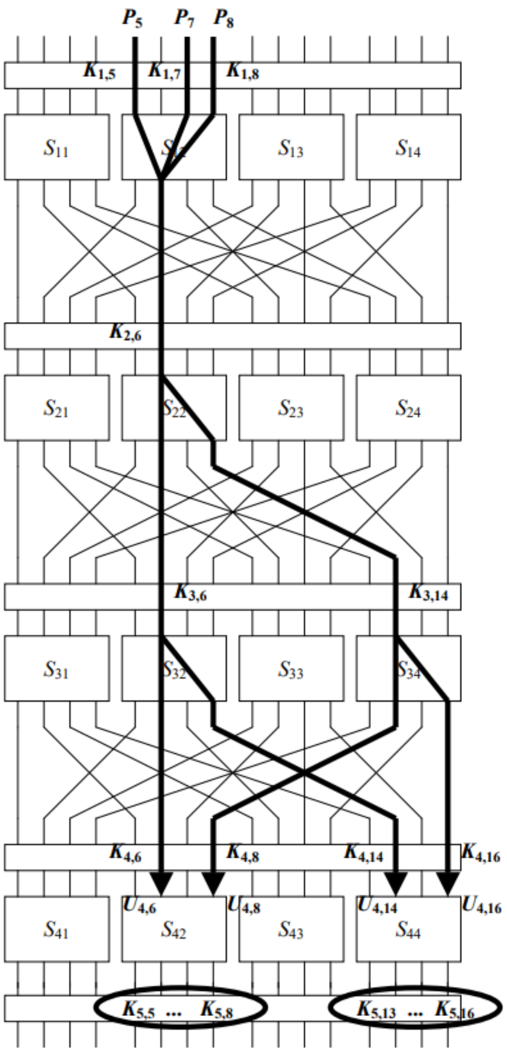


图 2: Sample Linear Approximation

在上一小节中我们讨论了 $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$ 的偏差，它们分别对应 4 组输入输出。需要指出的是，任何一组输入输出都可以对应 SPN 网络中的一个非线性层，也就是 $x$ 和 $\pi_s(x)$。分析多轮迭代的密码体制，一般通过组合多个单轮的线性逼近构成一个多轮的线性逼近，并且按照从下层到上层的顺序进行分析，因为中间层的结果可以通过构造多个线性逼近的时候消除。

具体思路是，先把多层迭代看作是一个整体，分析输入 $X_1$ 输出 $Y_n$，进行单轮的线性逼近，得到密钥 $K_n$。然后利用最后一层的输出，结合密钥 $K_n$ 和输出 $Y_n$ 反解出 $X_{n-1}$，以此类推，得到所有密钥。对应于本次实验的例子，如图2：

**第一轮（参照 $\epsilon_1 : X_1 \oplus X_3 \oplus X_4 \oplus Y_2$）**

$$V_{1,6} = U_{1,5} \oplus U_{1,7} \oplus U_{1,8} = (X_5 \oplus K_{1,5}) \oplus (X_7 \oplus K_{1,7}) \oplus (X_8 \oplus K_{1,8}) \tag{5}$$

**第二轮（参照 $\epsilon_2 : X_2 \oplus Y_2 \oplus Y_4$）**

$$V_{2,6} \oplus V_{2,8} = U_{2,6} = V_{1,6} \oplus K_{2,6} \tag{6}$$

**第三轮（参照 $\epsilon_3 : X_2 \oplus Y_2 \oplus Y_4$ 和 $\epsilon_4 : X_2 \oplus Y_2 \oplus Y_4$）**

$$V_{3,6} \oplus V_{3,8} = U_{3,6} = V_{2,6} \oplus K_{3,6} \tag{7}$$

$$V_{3,14} \oplus V_{3,16} = U_{3,14} = V_{2,8} \oplus K_{3,14} \tag{8}$$

**第四轮**

$$U_{4,6} = V_{3,6} \oplus K_{4,6} \tag{9}$$

$$U_{4,8} = V_{3,14} \oplus K_{4,8} \tag{10}$$

$$U_{4,14} = V_{3,8} \oplus K_{4,14} \tag{11}$$

$$U_{4,16} = V_{3,16} \oplus K_{4,16} \tag{12}$$

**联立 (5)-(12) 式得：**

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus X_5 \oplus X_7 \oplus X_8 \oplus \sum_K = 0 \tag{13}$$

$$\sum_K = K_{1,5} \oplus K_{1,7} \oplus K_{1,8} \oplus K_{2,6} \oplus K_{3,6} \oplus K_{4,6} \oplus K_{4,8} \oplus K_{4,14} \oplus K_{4,16} \tag{14}$$

因为密钥是确定的，所以 $\sum_K$ 是固定值，没有随机性。对于剩余部分：

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus X_5 \oplus X_7 \oplus X_8 = 0 \tag{15}$$

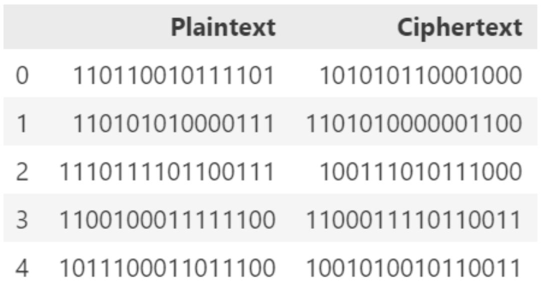
由堆积引理知，偏差值 $\epsilon = -\frac{1}{32}$ 通过上述推导我们得出，如果是正确密钥的话，结果会以偏差 $\epsilon = \pm\frac{1}{32}$ 的概率满足线性逼近，否则 $\epsilon = \pm\frac{1}{2}$

## 三、　实验过程

### (一)　基础部分：课本内容复现

在本小节中，我们将按照课本中提供的思路对子密钥 $[K_5...K_8, K_{13}...K_{16}]$ 进行破解。具体来说，为了保证攻击的可靠性，我们使用密钥 $K = 00111010100101001101011000111111$ 生成了 10000 对明文-密文对，然后通过这些明文-密文对发起已知明文攻击，生成算法对应在“代码实现”中的“明文密文对生成算法.ipynb”。：

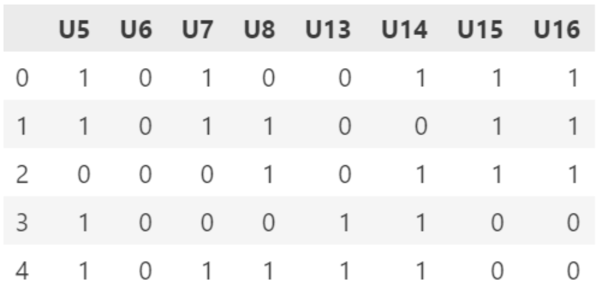
得到结果的前五列如图3所示:



| | Plaintext | Ciphertext |
|---|---|---|
| 0 | 110110010111101 | 101010110001000 |
| 1 | 110101010000111 | 1101010000001100 |
| 2 | 1110111101100111 | 100111010111000 |
| 3 | 1100100011111100 | 1100011110110011 |
| 4 | 1011100011011100 | 1001010010110011 |

图 3: 生成的明文-密文对

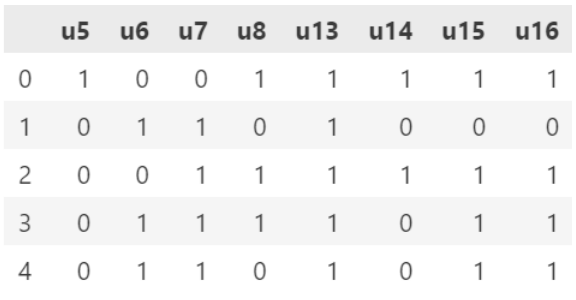
1. 对于目标子密钥 $[K_5...K_8, K_{13}...K_{16}]$, 枚举从 00000000 到 11111111 的 256 个密钥比特流, 通过与密文 $[Y_5...Y_8, Y_{13}...Y_{16}]$ 进行异或, 解出对应的密文得到 $[U_5...U_8, U_{13}...U_{16}]$, 这里我们展示了当 K=11111111 时 Y 与 K 异或结果, 如图4 得到结果的前五列如图4所示:



| | U5 | U6 | U7 | U8 | U13 | U14 | U15 | U16 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 3 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

图 4: Y 与 K 异或结果

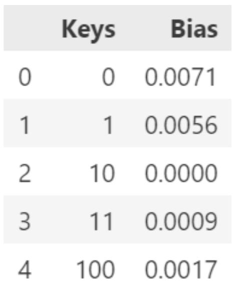
2. 然后, 通过 S 盒的逆变换, 求解出 $[u_5...u_8, u_{13}...u_{16}]$ , 这里我们同样展示前五列结果如图5



| | u5 | u6 | u7 | u8 | u13 | u14 | u15 | u16 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |

图 5: S 盒逆置换结果

3. 依据（15）式, 可以计算出符合该线性表达式的结果数, 从而我们能计算得到每组子密钥下对应的偏差（K=1-5 下对应的偏差绝对值如图6所示）。之所以可以使用这种方法, 是因为如果部分目标子密钥正确, 那么（15）式成立的偏差的绝对值会接近 1/32。而其他不正确的子密钥, 将会造成（15）式概率接近于 0。



| | Keys | Bias |
|---|---|---|
| 0 | 0 | 0.0071 |
| 1 | 1 | 0.0056 |
| 2 | 10 | 0.0000 |
| 3 | 11 | 0.0009 |
| 4 | 100 | 0.0017 |

图 6: 偏差统计结果

上述分析过程的代码实现对应在“代码实现”中的“线性分析算法.ipynb”。这里需要说明的是上面之所以不进行第 5 轮的计算，是因为若 R-1 轮线性分析后，从密文反推会更为容易。进而，可以根据我们计算得到的 256 个密钥对应的偏差的绝对值，找到偏差绝对值最接近 1/32 的对应密钥，这里我们列举出前 20 名，如表4所示：

| **Keys1** | **Bias1** | **Keys2** | **Bias2** |
|---|---|---|---|
| 01101111 | 0.0272 | 01101001 | 0.0148 |
| 01101101 | 0.0237 | 11101010 | 0.0148 |
| 10001101 | 0.0226 | 00010111 | 0.0148 |
| 11111001 | 0.0213 | 11101000 | 0.0148 |
| 00111011 | 0.0209 | 00110101 | 0.0147 |
| 00011001 | 0.0206 | 10011111 | 0.0146 |
| 01101110 | 0.0199 | 00101000 | 0.0144 |
| 01100011 | 0.0192 | 01011010 | 0.0144 |
| 00110110 | 0.0187 | 10101101 | 0.0143 |
| 11111011 | 0.0179 | 00111010 | 0.0139 |
| 00111000 | 0.0161 | 10110100 | 0.0139 |
| 11111010 | 0.0159 | 11101011 | 0.0138 |
| 00101011 | 0.0158 | 11100110 | 0.0134 |
| 00010101 | 0.0156 | 00011011 | 0.0134 |
| 01100111 | 0.0156 | 00101010 | 0.0130 |
| 10101110 | 0.0155 | 01100100 | 0.0130 |
| 10101111 | 0.0152 | 00010001 | 0.0127 |
| 01111010 | 0.0151 | 10111101 | 0.0126 |
| 11110000 | 0.0151 | 00110111 | 0.0125 |
| 11111000 | 0.0151 | 10010011 | 0.0124 |

表 4

从上表中我们可以看到，最接近 1/32 的偏差为 0.027200，对应子密钥：

$$K_5, K_6, K_7, K_8, K_{13}, K_{14}, K_{15}, K_{16} = 01101111$$

而 $K = 0011\ 1010\ 1001\ 0100\ 1101\ 0110\ 0011\ 1111$ 的后 16 字节对应位置上恰好是 0110 1111 证明破解成功。

## （二） 进阶部分：基于部分已破解子密钥破解剩余密钥的时间复杂度优化算法

### 朴素思路：

经过先前的分析，已经确定了子密钥 $K_5 \dots K_8, K_{13} \dots K_{16}$ 的具体值。为了解密其余的密钥部分，一个直观的方法是模仿前面的策略，构建具有显著偏差的线性近似来解密 $K_1 \dots K_4, K_9 \dots K_{12}$。但这需要执行 $2^8$ 次迭代，导致时间复杂度较大，这里不再具体实现。

### 进阶思路：

经过上一节的详细分析，我们成功地推导出了部分子密钥 $K_5 \dots K_8, K_{13} \dots K_{16}$。基于此，我们考虑了一种策略，即是否可以利用已经解密的子密钥来辅助解密尚未破解的子密钥。经过实验，这种策略被证明是有效的。具体就是，结合已知的子密钥和未知的子密钥可以有效减少所需的迭代次数。例如，当 $K_i \dots K_{i+3}$ 和 $K_j \dots K_{j+3}$ 均为未知时，我们需要进行 $2^8$ 次迭代来确定这些密钥。但是，如果我们已经知道 $K_j \dots K_{j+3}$ 中的部分密钥，那么迭代次数将减少到 $2^4$。更进一步，考虑 $K_1 \dots K_4$ 和 $K_9 \dots K_{12}$ 两部分，我们可以将它们与已经解密的 $K_5 \dots K_8$ 和 $K_{13} \dots K_{16}$ 结合，选择偏差较大的进行结合。通过这种方式，我们只需 $2 \times 2^4$ 次迭代就可以解密剩余的子密钥部分。

### 具体实现：

为节省篇幅，本部分专注于 $K_1, \dots, K_4$ 的解密细节。我们提供了图 7 中所示的线性近似，该近似对应的线性表达式为：

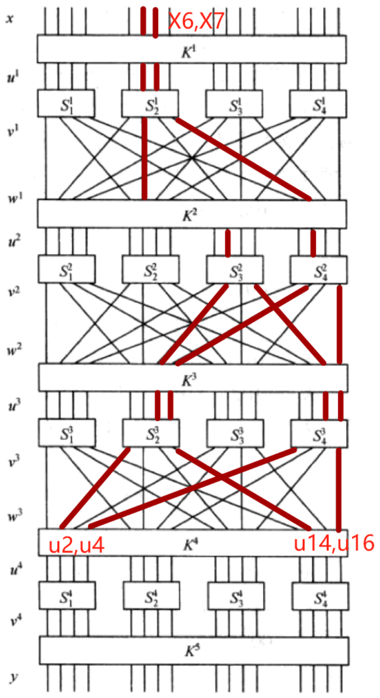


图 7: 构造线性表达式

$$X_6 \oplus X_7 \oplus u_2 \oplus u_4 \oplus u_{14} \oplus u_{16} \oplus K_1 \cdots \oplus K_4 \oplus K_{13} \cdots \oplus K_{16} = 0 \tag{16}$$

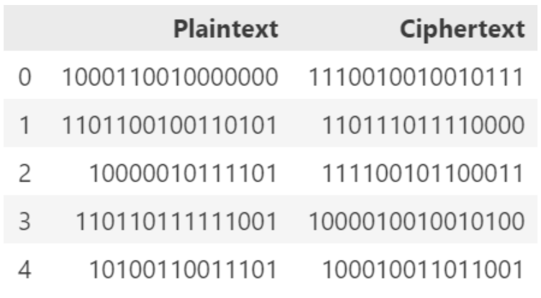
考虑到 $K_1 \cdots \oplus K_4 \oplus K_{13} \cdots \oplus K_{16}$ 是常量，可以将上式重写为:

$$X_6 \oplus X_7 \oplus u_2 \oplus u_4 \oplus u_{14} \oplus u_{16} = 0 \tag{17}$$

通过对照表 3 并利用堆积引理，计算得出偏差 $|\epsilon|$ 的值为:

$$|\epsilon| = 2^4 \times \left(\frac{1}{4}\right)^3 \times \left(\frac{3}{8}\right)^2 = \frac{9}{256} \tag{18}$$

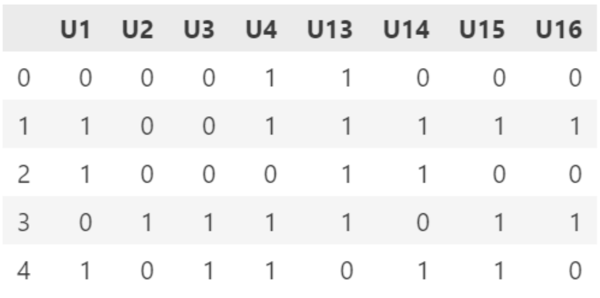
需要指出的是，$K_1, \ldots, K_4$ 是未知的，而 $K_{13}, \ldots, K_{16}$ 是已知的。我们生成了 10,000 对的明文-密文对（如图8），并从中提取了明文的 $X_6, X_7$ 和密文的 $Y_2, Y_4, Y_{14}, Y_{16}$。



| | Plaintext | Ciphertext |
|---|---|---|
| 0 | 1000110010000000 | 1110010010010111 |
| 1 | 1101100100110101 | 110111011110000 |
| 2 | 10000010111101 | 111100101100011 |
| 3 | 110110111111001 | 1000010010010100 |
| 4 | 10100110011101 | 100010011011001 |

图 8: 生成的明文-密文对（$K = (FF)_{16}$）

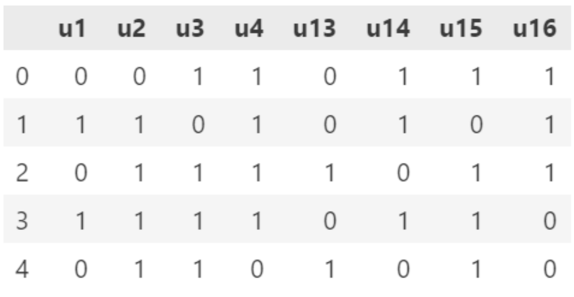
我们从 0000 到 1111 迭代生成了子密钥 $K_1, \ldots, K_4$，固定 $K_{13}, \ldots, K_{16}$ 的值为 1111，并与对应的密文进行异或操作，从而获得了 $U_2, U_4, U_{14}, U_{16}$ 的值（如图9）。



| | U1 | U2 | U3 | U4 | U13 | U14 | U15 | U16 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 4 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

图 9: Y 与 K 异或结果（$K = (FF)_{16}$）

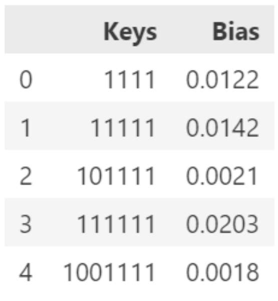
接下来，我们使用之前获得的 S 盒的逆操作得到 $u_2, u_4, u_{14}, u_{16}$（如图10）。



| | u1 | u2 | u3 | u4 | u13 | u14 | u15 | u16 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

图 10: S 盒逆置换结果（$K = (FF)_{16}$）

统计上述线性表达式在不同密钥下结果为 0 的次数，从而为每个密钥计算得出相应的偏差（如图11）。



| | Keys | Bias |
|---|---|---|
| 0 | 1111 | 0.0122 |
| 1 | 11111 | 0.0142 |
| 2 | 101111 | 0.0021 |
| 3 | 111111 | 0.0203 |
| 4 | 1001111 | 0.0018 |

图 11: 偏差统计结果（$K=(FF)_{16}$）

在 16 组偏差中，我们观察到当 $K_1K_2K_3K_4 = 1101$ 时，偏差的绝对值 0.0389 最接近 9/256，因此，我们得出了该子密钥（如表5），以上部分的代码详见“代码实现”的“进阶部分的代码”。

| **Keys1** | **Bias1** | **Keys2** | **Bias2** |
|---|---|---|---|
| 11011111 | 0.0389 | 11111111 | 0.0246 |
| 11101111 | 0.0226 | 00111111 | 0.0203 |
| 11001111 | 0.0165 | 00011111 | 0.0142 |
| 00001111 | 0.0122 | 01011111 | 0.0097 |
| 01101111 | 0.0081 | 10101111 | 0.0071 |
| 10011111 | 0.0055 | 10111111 | 0.0044 |
| 00101111 | 0.0021 | 01001111 | 0.0018 |
| 01111111 | 0.0014 | 10001111 | 0.0012 |

表 5

现在，已经得到了子密钥 $K_1,...,K_8,K_{13},...,K_{16}$ 的结果。在这里，我们既可以用已经得到的子密钥辅助求解未知密钥 $K_9,...,K_{12}$，也可以不借助已知密钥直接求解，这里均只需要尝试 $2_4$ 次便可破解出密钥。因此，寻找较大的偏差便成为了我们构造线性逼近的主要依据。构造线性逼近:

$$X_2 \oplus X_3 \oplus X_4 \oplus X_6 \oplus X_7 \oplus X_8 \oplus u_{11} \oplus K_9 \oplus K_{10} \oplus K_{11} \oplus K_{11} = 0 \tag{19}$$

计算得出偏差 $|\epsilon|$ 的值为:

$$|\epsilon| = 2^4 \times \left(\frac{1}{4}\right)^5 = \frac{1}{64} \tag{20}$$

现在，我们已知子密钥 $K_1,...,K_8,K_{13},...,K_{16}$。显然，对于未知的子密钥 $K_9,...,K_{12}$，在已知和未知的子密钥上的尝试次数在这种情境下是相同的，即 $2^4$。这是因为无论我们是否使用已知的子密钥信息，尝试所有可能的子密钥组合仍然需要 $2^4$ 次尝试。为了优化攻击效率，我们通常寻找具有较大偏差的线性逼近。在此情境下，线性逼近表达式为:

$$X_2 \oplus X_3 \oplus X_4 \oplus X_6 \oplus X_7 \oplus X_8 \oplus u_{11} \oplus K_9 \oplus K_{10} \oplus K_{11} \oplus K_{11} = 0 \tag{21}$$

该逼近的偏差 $|\epsilon|$ 可以量化为:

$$|\epsilon| = 2^4 \times \left(\frac{1}{4}\right)^5 = \frac{1}{64} \tag{22}$$

在实际的代码实现中，我们可以通过调整在“进阶部分代码.ipynb”中的参数来计算并验证该线性逼近。

```
# =============Please modify the parameters here============= #
    code
# =============Please modify the parameters here============= #
```

代码段提供了参数调整的位置，这样用户可以根据他们的需求灵活地调整。经过适当的参数调整，我们确定 $K_9K_{10}K_{11}K_{12}$ 的值为 0011。综合起来，我们成功破解了完整的密钥，它是：

$$K = 1101\ 0110\ 0011\ 1111$$

此结果再次验证了密码分析中线性逼近方法的有效性和适用性。

## 四、 实验结论及心得体会

在本次实验中，我深深地体验到了线性密码分析方法的强大和高效。尤其是在进阶部分，通过结合已知的部分子密钥辅助解密尚未破解的子密钥，这种策略的巧妙之处令我印象深刻。首先，基础部分虽然是对课本内容的复现，但它为进阶部分奠定了坚实的基础。当我们掌握了密钥的某一部分后，如何利用这部分信息进行更高效的攻击是一个挑战。在进阶部分，我深刻地体会到了密码分析不仅仅是算法和技术，更多的是策略和思维方式。如何巧妙地利用已有的信息，将问题分解，然后一步步逼近真实的密钥，这是一种既刺激又有趣的经验。尤其值得一提的是，当我固定已知的子密钥并对未知的部分进行迭代时，我体验到了巨大的时间复杂度降低。这种优化思路的启示是，有时候，我们不需要从零开始，可以借助已知的部分信息来大大减少计算的复杂度。此外，我也认识到，尽管线性逼近方法是一个强大的工具，但它也有其局限性。选择一个具有大偏差的线性逼近是关键。在这次实验中，我多次尝试了不同的线性逼近，直到找到了一个满意的结果。这也让我意识到，密码分析需要耐心、细致和创造性。

## 五、 代码实现

在本节中，我们给出复现实验所需要的全部代码。

表 3 的代码实现

```cpp
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;

const int sbox[] = {0xe, 4, 0xd, 1, 2, 0xf, 0xb, 8, 3, 0xa, 6, 0xc, 5, 9, 0, 7};
const int SIZE_SBOX = sizeof(sbox) / sizeof(sbox[0]);

// Compute the linear approximation for a given "input = output" equation
int linearApprox(int input_int, int output_int) {
    int total = 0;
    for (int ii = 0; ii < SIZE_SBOX; ++ii) {
        int input_masked = ii & input_int;
        int output_masked = sbox[ii] & output_int;
        if (__builtin_popcount(input_masked) % 2 == __builtin_popcount(
            output_masked) % 2) {
            total += 1;
        }
    }
    return total - (SIZE_SBOX / 2);
}

int main() {
    vector<int> linear_approx_table(SIZE_SBOX * SIZE_SBOX);

    // Headers
    cout << "      |";
    for (int i = 0; i < SIZE_SBOX; ++i) {
        cout << setw(4) << i;
    }
    cout << "\n" << string(SIZE_SBOX * 4 + 6, '-') << "\n";

    for (int row = 0; row < SIZE_SBOX; ++row) {
        cout << setw(4) << row << " |";
        for (int col = 0; col < SIZE_SBOX; ++col) {
            int r = linearApprox(row, col);
            cout << setw(4) << r;
            linear_approx_table[row * SIZE_SBOX + col] = r;
        }
        cout << "\n";
    }

    return 0;
}
```

明文密文对生成算法.ipynb

```python
import random
import csv

# S-Box: Used for substitution operation, providing a non-linear substitution step.
S_Box = [0xE, 0x4, 0xD, 0x1, 0x2, 0xF, 0xB, 0x8, 0x3, 0xA, 0x6, 0xC, 0x5, 0x9, 0x0,
     0x7]

# P-Box: Used for permutation operation, it rearranges the input bits.
P_Box = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16]

# Generate five 16-bit subkeys from a 32-bit master key.
def gen_K_list(K):
    Ks = []
    for _ in range(5):
        Ks.append(K & 0xFFFF)  # Get the last 16 bits of K
        K >>= 4  # Right shift by 4 bits
    return Ks[::-1]

# Perform substitution operation using the S-Box.
def pi_s(s_box, ur):
    vr = 0
    for i in range(4):
        uri = ur & 0xF  # Get the last 4 bits of ur
        vri = s_box[uri]  # Get the substitute value from the S-Box
        vr += vri << (4 * i)  # Store the result in vr
        ur >>= 4  # Right shift by 4 bits
    return vr

# Perform permutation operation using the P-Box.
def pi_p(p_box, vr):
    wr = 0
    for i in range(15, -1, -1):
        vri = vr & 1  # Get the last bit of vr
        vr >>= 1  # Right shift by 1 bit
        wr |= vri << (16 - p_box[i])  # Rearrange bits based on P-Box value
    return wr

# Implement the SPN encryption operation.
def do_SPN(x, s_box, p_box, Ks):
    wr = x
    # Execute Nr-1 rounds of encryption, each round includes three steps: Key
        addition, substitution, and permutation.
    for r in range(3):
        ur = wr ^ Ks[r]  # Key addition
        vr = pi_s(s_box, ur)  # Substitution
        wr = pi_p(p_box, vr)  # Permutation
    # The last round does not include permutation.
    ur = wr ^ Ks[3]  # Key addition
    vr = pi_s(s_box, ur)  # Substitution
    return vr ^ Ks[4]  # Output the result after another key addition

# Encrypt a 16-bit plaintext with a given 32-bit key.
def encrypt(K, x):
    Ks = gen_K_list(K)  # Generate subkeys
    return do_SPN(x, S_Box, P_Box, Ks)  # Execute SPN encryption

# Generate plaintext-ciphertext pairs
def generate_pairs():
    # 0011 1010 1001 0100 1101 0110 0011 1111
    K = int("00111010100101001101011000111111", 2)

    pairs = []
    # setting numbers here
    for _ in range(10000):
        plaintext = random.randint(0, 0xFFFF)
        ciphertext = encrypt(K, plaintext)
        pairs.append((format(plaintext, '016b'), format(ciphertext, '016b')))

    return pairs

# Save the pairs to a CSV file
def save_to_csv(pairs, filename):
    with open(filename, 'w', newline='') as csvfile:
        fieldnames = ['Plaintext', 'Ciphertext']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for p, c in pairs:
            writer.writerow({'Plaintext': p, 'Ciphertext': c})

# Generate and save plaintext-ciphertext pairs to a CSV file
```

```python
pairs = generate_pairs()
filename = "./data/明文_密文对.csv"
save_to_csv(pairs, filename)
```

线性分析算法.ipynb

```python
import csv

def calculate_bias(K_values):
    # Read the CSV file and extract the specified bits from the ciphertext
    def extract_bits_from_ciphertext(filename):
        extracted_data = []

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                ciphertext = row['Ciphertext']
                extracted_bits = {
                    'Y5': ciphertext[4],
                    'Y6': ciphertext[5],
                    'Y7': ciphertext[6],
                    'Y8': ciphertext[7],
                    'Y13': ciphertext[12],
                    'Y14': ciphertext[13],
                    'Y15': ciphertext[14],
                    'Y16': ciphertext[15],
                }
                extracted_data.append(extracted_bits)

        # Write the extracted bits to a new CSV file
        output_filename = "./data/特定比特位的密文.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['Y5', 'Y6', 'Y7', 'Y8', 'Y13', 'Y14', 'Y15', 'Y16']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in extracted_data:
                writer.writerow(row)

        return output_filename

    # Extract the specified bits from the uploaded file and save to a new CSV file
    output_file = extract_bits_from_ciphertext("./data/明文_密文对.csv")

    # Define the default values for K bits


    # Perform XOR operation on extracted bits and K values
    def xor_operation(input_filename):
        xor_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                xor_row = {
                    'U5': str(int(row['Y5']) ^ int(K_values['K5'])),
                    'U6': str(int(row['Y6']) ^ int(K_values['K6'])),
                    'U7': str(int(row['Y7']) ^ int(K_values['K7'])),
                    'U8': str(int(row['Y8']) ^ int(K_values['K8'])),
                    'U13': str(int(row['Y13']) ^ int(K_values['K13'])),
                    'U14': str(int(row['Y14']) ^ int(K_values['K14'])),
                    'U15': str(int(row['Y15']) ^ int(K_values['K15'])),
                    'U16': str(int(row['Y16']) ^ int(K_values['K16']))
                }
                xor_results.append(xor_row)

        # Write the XOR results to a new CSV file
        output_filename = "./data/Y与K异或结果.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['U5', 'U6', 'U7', 'U8', 'U13', 'U14', 'U15', 'U16']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in xor_results:
                writer.writerow(row)

        return output_filename

    # Perform XOR operation with the specified bits and save to a new CSV file
    output_xor_file = xor_operation("./data/特定比特位的密文.csv")

    # Convert binary values to hexadecimal and save to a new CSV file
    def binary_to_hex(binary_str):
        """Convert a binary string to a hexadecimal string."""
        return format(int(binary_str, 2), 'x').upper()

    def convert_and_save(input_filename):
        hex_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                H1 = binary_to_hex(row['U5'] + row['U6'] + row['U7'] + row['U8'])
                H2 = binary_to_hex(row['U13'] + row['U14'] + row['U15'] + row['U16'
                    ])
                hex_row = {'H1': H1, 'H2': H2}
                hex_results.append(hex_row)

        # Write the hexadecimal results to a new CSV file
        output_filename = "./data/Y与K异或结果（16进制）.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['H1', 'H2']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in hex_results:
                writer.writerow(row)

        return output_filename

    # Convert binary values to hexadecimal and save to a new CSV file
    output_hex_file = convert_and_save("./data/Y与K异或结果.csv")
```

```python

    # Define the conversion mapping for H1 to X1
    conversion_map = {
        'E': '0', '4': '1', 'D': '2', '1': '3',
        '2': '4', 'F': '5', 'B': '6', '8': '7',
        '3': '8', 'A': '9', '6': 'A', 'C': 'B',
        '5': 'C', '9': 'D', '0': 'E', '7': 'F'
    }

    # Convert H1 and H2 to X1 and X2 using the provided mapping
    def convert_H_to_X(input_filename):
        X_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                X1 = conversion_map[row['H1']]
                X2 = conversion_map[row['H2']]
                X_row = {'X1': X1, 'X2': X2}
                X_results.append(X_row)

        # Write the X values to a new CSV file
        output_filename = "./data/S盒逆置换结果（16进制）.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['X1', 'X2']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in X_results:
                writer.writerow(row)

        return output_filename

    # Convert H1 and H2 to X1 and X2 using the provided mapping and save to a new
        CSV file
    output_X_file = convert_H_to_X("./data/Y与K异或结果（16进制）.csv")

    # Convert hexadecimal values to binary and save to a new CSV file
    def hex_to_binary(hex_str):
        """Convert a hexadecimal string to a binary string."""
        return format(int(hex_str, 16), '04b')

    # Convert X1 and X2 to binary and save to a new CSV file
    def convert_X_to_u(input_filename):
        u_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                u5, u6, u7, u8 = tuple(hex_to_binary(row['X1']))
                u13, u14, u15, u16 = tuple(hex_to_binary(row['X2']))
                u_row = {
                    'u5': u5, 'u6': u6, 'u7': u7, 'u8': u8,
                    'u13': u13, 'u14': u14, 'u15': u15, 'u16': u16
                }
                u_results.append(u_row)

        # Write the u values to a new CSV file
        output_filename = "./data/S盒逆置换结果.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['u5', 'u6', 'u7', 'u8', 'u13', 'u14', 'u15', 'u16']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in u_results:
                writer.writerow(row)

        return output_filename

    # Convert X1 and X2 to u values and save to a new CSV file
    output_u_file = convert_X_to_u("./data/S盒逆置换结果（16进制）.csv")

    # Extract specified bits from plaintext and save to a new CSV file
    def extract_bits_from_plaintext(filename):
        extracted_data = []

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                plaintext = row['Plaintext']
                extracted_bits = {
                    'x5': plaintext[4],
                    'x7': plaintext[6],
                    'x8': plaintext[7]
                }
                extracted_data.append(extracted_bits)

        # Write the extracted bits to a new CSV file
        output_filename = "./data/特定比特位的明文.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['x5', 'x7', 'x8']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in extracted_data:
                writer.writerow(row)

        return output_filename

    # Extract specified bits from plaintext and save to a new CSV file
    output_plaintext_bits_file = extract_bits_from_plaintext("./data/明文_密文对.
        csv")

    # Merge extracted plaintext bits and u results, then calculate the XOR result
    def merge_and_calculate_xor(plaintext_file, u_file):
        with open(plaintext_file, 'r') as plaintext_csv, open(u_file, 'r') as u_csv
            :
            plaintext_reader = csv.DictReader(plaintext_csv)
            u_reader = csv.DictReader(u_csv)

            merged_data = []
            for plaintext_row, u_row in zip(plaintext_reader, u_reader):
                xor_result = int(u_row['u6']) ^ int(u_row['u8']) ^ int(u_row['u14'
                    ]) ^ int(u_row['u16']) \
                            ^ int(plaintext_row['x5']) ^ int(plaintext_row['x7']) ^
                                int(plaintext_row['x8'])

                merged_row = {
                    'x5': plaintext_row['x5'],
                    'x7': plaintext_row['x7'],
                    'x8': plaintext_row['x8'],
                    'u6': u_row['u6'],
                    'u8': u_row['u8'],
                    'u14': u_row['u14'],
                    'u16': u_row['u16'],
                    'aws': str(xor_result)
                }
                merged_data.append(merged_row)

        # Write the merged data and XOR result to a new CSV file
        output_filename = "./data/随机变量异或逻辑表.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['x5', 'x7', 'x8', 'u6', 'u8', 'u14', 'u16', 'aws']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in merged_data:
                writer.writerow(row)

        return output_filename

    # Merge extracted plaintext bits and u results, calculate XOR result, and save
        to a new CSV file
    output_answer_file = merge_and_calculate_xor("./data/特定比特位的明文.csv", "./
        data/S盒逆置换结果.csv")

        # Count the number of zeros in the 'aws' column of the answer.csv file
    def count_zeros_in_aws(filename):
        count = 0

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                if row['aws'] == '0':
                    count += 1

        return count

    # Count the number of zeros in the 'aws' column of the answer.csv file
    zero_count = count_zeros_in_aws("./data/随机变量异或逻辑表.csv")

    # Count the total number of entries in the 'aws' column of the answer.csv file
    def count_total_entries_in_aws(filename):
        count = 0

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for _ in reader:
                count += 1

        return count

    # Count the total number of entries in the 'aws' column of the answer.csv file
    total_count = count_total_entries_in_aws("./data/随机变量异或逻辑表.csv")

    # Calculate the bias
    bias = zero_count / total_count - 0.5

    return bias

    # Initialize the K_values dictionary
K_values = {
    'K5': '0',
    'K6': '0',
    'K7': '0',
    'K8': '0',
    'K13': '0',
    'K14': '0',
    'K15': '0',
    'K16': '0'
}

# Initialize a list to store biases for each iteration
biases = []

# Iterate through 256 values (0 to 255)
for i in range(256):
    # Convert the current value of 'i' to an 8-bit binary string
    binary_str = format(i, '08b')

    # Update the K_values dictionary with the binary string
    K_values['K5'] = binary_str[0]
    K_values['K6'] = binary_str[1]
    K_values['K7'] = binary_str[2]
    K_values['K8'] = binary_str[3]
    K_values['K13'] = binary_str[4]
    K_values['K14'] = binary_str[5]
    K_values['K15'] = binary_str[6]
    K_values['K16'] = binary_str[7]

    # Calculate the bias for the current K_values
    bias = calculate_bias(K_values)

    # Append the bias to the list
    biases.append(bias)

# Create and write the results to a CSV file
output_filename = "./data/偏差统计结果.csv"
with open(output_filename, 'w', newline='') as csvfile:
    fieldnames = ['Keys', 'Bias']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for i, bias in enumerate(biases):
        keys_binary = format(i, '08b')
        writer.writerow({'Keys': keys_binary, 'Bias': abs(bias)})

print("Results saved to", output_filename)

# Load the biases from the CSV file
biases = []
with open("./data/偏差统计结果.csv", 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        # Parse the Keys and Bias columns from the CSV
        keys = row['Keys']
        bias = float(row['Bias'])
        biases.append((keys, bias))

# Initialize variables to keep track of the closest bias and its corresponding keys
closest_bias = float('inf')  # Initialize with positive infinity
closest_keys = None

# Iterate through the biases and find the closest one to 1/32
for keys, bias in biases:
    # Calculate the absolute difference from 1/32
    diff = abs(bias - 1/32)

    # Check if this bias is closer than the previous closest
    if diff < abs(closest_bias - 1/32):
        closest_bias = bias
        closest_keys = keys

# Print the keys and bias that are closest to 1/32
print(f"Keys: {closest_keys}")
print(f"Closest Bias: {closest_bias:.6f}")
```

进阶部分代码.ipynb

```
import csv

def calculate_bias(K_values):
    # Read the CSV file and extract the specified bits from the ciphertext
    def extract_bits_from_ciphertext(filename):
        extracted_data = []

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                ciphertext = row['Ciphertext']
                extracted_bits = {

                    # =============Please modify the parameters here============= #
                    'Y1': ciphertext[0],
                    'Y2': ciphertext[1],
                    'Y3': ciphertext[2],
                    'Y4': ciphertext[3],
                    'Y13': ciphertext[12],
                    'Y14': ciphertext[13],
                    'Y15': ciphertext[14],
                    'Y16': ciphertext[15],
                    # =============Please modify the parameters here============= #

                }
                extracted_data.append(extracted_bits)

        # Write the extracted bits to a new CSV file
        output_filename = "./data/特定比特位的密文.csv"
        with open(output_filename, 'w', newline='') as csvfile:

            # =============Please modify the parameters here============= #
            fieldnames = ['Y1', 'Y2', 'Y3', 'Y4', 'Y13', 'Y14', 'Y15', 'Y16']
            # =============Please modify the parameters here============= #

            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in extracted_data:
                writer.writerow(row)

        return output_filename

    # Extract the specified bits from the uploaded file and save to a new CSV file
    output_file = extract_bits_from_ciphertext("./data/明文_密文对.csv")

    # Define the default values for K bits


    # Perform XOR operation on extracted bits and K values
    def xor_operation(input_filename):
        xor_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                xor_row = {

                    # =============Please modify the parameters here============= #
                    'U1': str(int(row['Y1']) ^ int(K_values['K1'])),
                    'U2': str(int(row['Y2']) ^ int(K_values['K2'])),
                    'U3': str(int(row['Y3']) ^ int(K_values['K3'])),
                    'U4': str(int(row['Y4']) ^ int(K_values['K4'])),
                    'U13': str(int(row['Y13']) ^ int(K_values['K13'])),
                    'U14': str(int(row['Y14']) ^ int(K_values['K14'])),
                    'U15': str(int(row['Y15']) ^ int(K_values['K15'])),
                    'U16': str(int(row['Y16']) ^ int(K_values['K16']))
                    # =============Please modify the parameters here============= #

                }
                xor_results.append(xor_row)

        # Write the XOR results to a new CSV file
        output_filename = "./data/Y与K异或结果.csv"
        with open(output_filename, 'w', newline='') as csvfile:

            # =============Please modify the parameters here============= #
```

```python
            fieldnames = ['U1', 'U2', 'U3', 'U4', 'U13', 'U14', 'U15', 'U16']
            # =============Please modify the parameters here============= #

            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in xor_results:
                writer.writerow(row)

        return output_filename

    # Perform XOR operation with the specified bits and save to a new CSV file
    output_xor_file = xor_operation("./data/特定比特位的密文.csv")

    # Convert binary values to hexadecimal and save to a new CSV file
    def binary_to_hex(binary_str):
        """Convert a binary string to a hexadecimal string."""
        return format(int(binary_str, 2), 'x').upper()

    def convert_and_save(input_filename):
        hex_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:

                # =============Please modify the parameters here============= #
                H1 = binary_to_hex(row['U1'] + row['U2'] + row['U3'] + row['U4'])
                H2 = binary_to_hex(row['U13'] + row['U14'] + row['U15'] + row['U16'
                    ])
                # =============Please modify the parameters here============= #

                hex_row = {'H1': H1, 'H2': H2}
                hex_results.append(hex_row)

        # Write the hexadecimal results to a new CSV file
        output_filename = "./data/Y与K异或结果（16进制）.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['H1', 'H2']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in hex_results:
                writer.writerow(row)

        return output_filename

    # Convert binary values to hexadecimal and save to a new CSV file
    output_hex_file = convert_and_save("./data/Y与K异或结果.csv")


    # Define the conversion mapping for H1 to X1
    conversion_map = {
        'E': '0', '4': '1', 'D': '2', '1': '3',
        '2': '4', 'F': '5', 'B': '6', '8': '7',
        '3': '8', 'A': '9', '6': 'A', 'C': 'B',
        '5': 'C', '9': 'D', '0': 'E', '7': 'F'
    }

    # Convert H1 and H2 to X1 and X2 using the provided mapping
    def convert_H_to_X(input_filename):
        X_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                X1 = conversion_map[row['H1']]
                X2 = conversion_map[row['H2']]
                X_row = {'X1': X1, 'X2': X2}
                X_results.append(X_row)

        # Write the X values to a new CSV file
        output_filename = "./data/S盒逆置换结果（16进制）.csv"
        with open(output_filename, 'w', newline='') as csvfile:
            fieldnames = ['X1', 'X2']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in X_results:
                writer.writerow(row)

        return output_filename

    # Convert H1 and H2 to X1 and X2 using the provided mapping and save to a new
        CSV file
    output_X_file = convert_H_to_X("./data/Y与K异或结果（16进制）.csv")

    # Convert hexadecimal values to binary and save to a new CSV file
    def hex_to_binary(hex_str):
        """Convert a hexadecimal string to a binary string."""
        return format(int(hex_str, 16), '04b')

    # Convert X1 and X2 to binary and save to a new CSV file
    def convert_X_to_u(input_filename):
        u_results = []

        with open(input_filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                u1, u2, u3, u4 = tuple(hex_to_binary(row['X1']))

                # =============Please modify the parameters here============= #
                u13, u14, u15, u16 = tuple(hex_to_binary(row['X2']))
                u_row = {
                    'u1': u1, 'u2': u2, 'u3': u3, 'u4': u4,
                    'u13': u13, 'u14': u14, 'u15': u15, 'u16': u16
                }
                # =============Please modify the parameters here============= #

                u_results.append(u_row)

        # Write the u values to a new CSV file
        output_filename = "./data/S盒逆置换结果.csv"
        with open(output_filename, 'w', newline='') as csvfile:

            # =============Please modify the parameters here============= #
            fieldnames = ['u1', 'u2', 'u3', 'u4', 'u13', 'u14', 'u15', 'u16']
            # =============Please modify the parameters here============= #

            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in u_results:
                writer.writerow(row)

        return output_filename

    # Convert X1 and X2 to u values and save to a new CSV file
    output_u_file = convert_X_to_u("./data/S盒逆置换结果（16进制）.csv")

    # Extract specified bits from plaintext and save to a new CSV file
    def extract_bits_from_plaintext(filename):
        extracted_data = []

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                plaintext = row['Plaintext']
                extracted_bits = {

                    # =============Please modify the parameters here============= #
                    'x6': plaintext[5],
                    'x7': plaintext[6]
                    # =============Please modify the parameters here============= #

                }
                extracted_data.append(extracted_bits)

        # Write the extracted bits to a new CSV file
        output_filename = "./data/特定比特位的明文.csv"
        with open(output_filename, 'w', newline='') as csvfile:

            # =============Please modify the parameters here============= #
            fieldnames = ['x6', 'x7']
            # =============Please modify the parameters here============= #

            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for row in extracted_data:
                writer.writerow(row)

        return output_filename

    # Extract specified bits from plaintext and save to a new CSV file
    output_plaintext_bits_file = extract_bits_from_plaintext("./data/明文_密文对.
        csv")

    # Merge extracted plaintext bits and u results, then calculate the XOR result
    def merge_and_calculate_xor(plaintext_file, u_file):
```

```python
    with open(plaintext_file, 'r') as plaintext_csv, open(u_file, 'r') as u_csv:
        plaintext_reader = csv.DictReader(plaintext_csv)
        u_reader = csv.DictReader(u_csv)

        merged_data = []
        for plaintext_row, u_row in zip(plaintext_reader, u_reader):

            # =============Please modify the parameters here============= #
            xor_result = int(u_row['u2']) ^ int(u_row['u4']) ^ int(u_row['u14']) ^ int(u_row['u16']) ^ int(plaintext_row['x6']) ^ int(plaintext_row['x7'])
            merged_row = {
                'x6': plaintext_row['x6'],
                'x7': plaintext_row['x7'],
                'u2': u_row['u2'],
                'u4': u_row['u4'],
                'u14': u_row['u14'],
                'u16': u_row['u16'],
                'aws': str(xor_result)
            }
            # =============Please modify the parameters here============= #

            merged_data.append(merged_row)

    # Write the merged data and XOR result to a new CSV file
    output_filename = "./data/随机变量异或逻辑表.csv"
    with open(output_filename, 'w', newline='') as csvfile:

        # =============Please modify the parameters here============= #
        fieldnames = ['x6', 'x7', 'u2', 'u4', 'u14', 'u16','aws']
        # =============Please modify the parameters here============= #

        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for row in merged_data:
            writer.writerow(row)

    return output_filename

# Merge extracted plaintext bits and u results, calculate XOR result, and save to a new CSV file
output_answer_file = merge_and_calculate_xor("./data/特定比特位的明文.csv", "./data/S盒逆置换结果.csv")

    # Count the number of zeros in the 'aws' column of the answer.csv file
def count_zeros_in_aws(filename):
    count = 0

    with open(filename, 'r') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            if row['aws'] == '0':
                count += 1

        return count

    # Count the number of zeros in the 'aws' column of the answer.csv file
    zero_count = count_zeros_in_aws("./data/随机变量异或逻辑表.csv")

    # Count the total number of entries in the 'aws' column of the answer.csv file
    def count_total_entries_in_aws(filename):
        count = 0

        with open(filename, 'r') as csvfile:
            reader = csv.DictReader(csvfile)
            for _ in reader:
                count += 1

        return count

    # Count the total number of entries in the 'aws' column of the answer.csv file
    total_count = count_total_entries_in_aws("./data/随机变量异或逻辑表.csv")

    # Calculate the bias
    bias = zero_count / total_count - 0.5

    return bias

# Initialize the K_values dictionary with K9 to K12 set to '1'

# =============Please modify the parameters here============= #
K_values = {
    'K1': '0',
    'K2': '0',
    'K3': '0',
    'K4': '0',
    'K13': '1',
    'K14': '1',
    'K15': '1',
    'K16': '1'
}
# =============Please modify the parameters here============= #

# Initialize a list to store biases for each iteration
biases = []

# Iterate through 16 values (0 to 15) for K1 to K4
# =============Please modify the parameters here============= #
for i in range(16):
    # Convert the current value of 'i' to a 4-bit binary string
    binary_str = format(i, '04b')

    # Update the K_values dictionary with the binary string for K1 to K4
    K_values['K1'] = binary_str[0]
    K_values['K2'] = binary_str[1]
    K_values['K3'] = binary_str[2]
    K_values['K4'] = binary_str[3]

    # Calculate the bias for the current K_values
    bias = calculate_bias(K_values)

    # Append the bias to the list
    biases.append(bias)
# =============Please modify the parameters here============= #


# Create and write the results to a CSV file
output_filename = "./data/偏差统计结果.csv"
with open(output_filename, 'w', newline='') as csvfile:
    fieldnames = ['Keys', 'Bias']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for i, bias in enumerate(biases):
        keys_binary = format(i, '04b') + '1111'  # Appending '1111' for K9 to K12
        writer.writerow({'Keys': keys_binary, 'Bias': abs(bias)})

# Print the output filename
output_filename

# Load the biases from the CSV file
biases = []
with open("./data/偏差统计结果.csv", 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        # Parse the Keys and Bias columns from the CSV
        keys = row['Keys']
        bias = float(row['Bias'])
        biases.append((keys, bias))

# Initialize variables to keep track of the closest bias and its corresponding keys
closest_bias = float('inf')  # Initialize with positive infinity
closest_keys = None

# Iterate through the biases and find the closest one to 9/256
for keys, bias in biases:
    # Calculate the absolute difference from 9/256

    diff = abs(bias - 9/256)

    # Check if this bias is closer than the previous closest
    if diff < abs(closest_bias - 9/256):
        closest_bias = bias
        closest_keys = keys

# Print the keys and bias that are closest to 9/256
print(f"Keys: {closest_keys}")
print(f"Closest Bias: {closest_bias:.6f}")
```

# 参考文献

[1] Heys H M .A Tutorial on[J].Linear Differential Cryptanalysis, 2001.DOI:doi:10.1109/CISS.2008.4558479.