

南開大學

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, 我们结合了四轮的概率值, 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	1
二、 实验原理	1
(一) 代换规则 $\pi_s(z)$ 与置换规则 $\pi_p(z)$	1
(二) 偏差与堆积引理	2
(三) S 盒线性逼近	2
(四) SPN 的线性逼近	4
三、 实验过程	5
(一) 基础部分：课本内容复现	5
(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	8
四、 实验结论及心得体会	11
五、 代码实现	11

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：

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

(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 ϵ_i 为：

$$\epsilon_i = P(X_i = 0) - 0.5 \quad (1)$$

由此定义，我们可以得出：

$$P(X_i = 0) = \epsilon_i + 0.5 \quad (2)$$

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, \dots, 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 \oplus 表示异或操作。

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

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

$$\epsilon = 2^{n-1} \prod_{i=1}^n \epsilon_i \quad (4)$$

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。

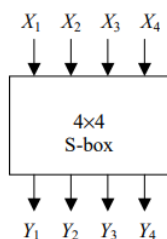


图 1: S-box Mapping

设输入 $X = (X_1, X_2, \dots, X_m)$, 输出为 $Y = (Y_1, Y_2, \dots, Y_n)$, 然后我们可以用形如: $X_1 \oplus X_2 \oplus \dots \oplus X_i \oplus Y_1 \oplus Y_2 \oplus \dots \oplus Y_j$ 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 16×16 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:

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

(四) SPN 的线性逼近

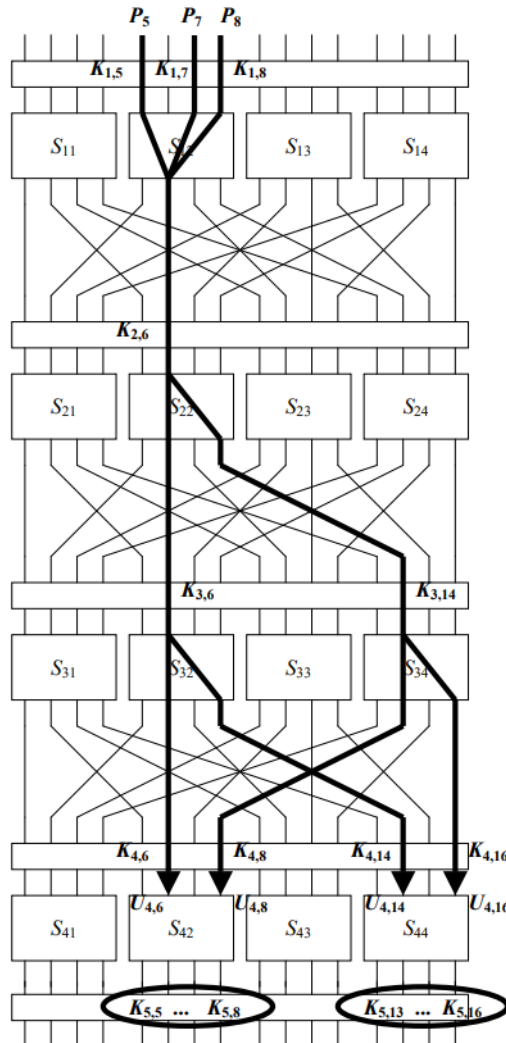


图 2: Sample Linear Approximation

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

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

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

$$V_{2,6} \oplus V_{2,8} = U_{2,6} = V_{1,6} \oplus K_{2,6} \quad (6)$$

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

$$V_{3,6} \oplus V_{3,8} = U_{3,6} = V_{2,6} \oplus K_{3,6} \quad (7)$$

$$V_{3,14} \oplus V_{3,16} = U_{3,14} = V_{2,8} \oplus K_{3,14} \quad (8)$$

第四轮

$$U_{4,6} = V_{3,6} \oplus K_{4,6} \quad (9)$$

$$U_{4,8} = V_{3,14} \oplus K_{4,8} \quad (10)$$

$$U_{4,14} = V_{3,8} \oplus K_{4,14} \quad (11)$$

$$U_{4,16} = V_{3,16} \oplus K_{4,16} \quad (12)$$

联立 (5)-(12) 式得：

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

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

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：

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

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 $[K_5 \dots K_8, K_{13} \dots K_{16}]$ 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“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 2×2^4 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：

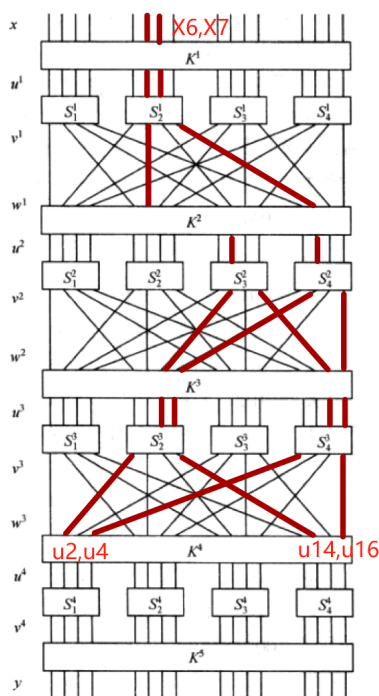


图 7: 构造线性表达式

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

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:

$$X_6 \oplus X_7 \oplus u_2 \oplus u_4 \oplus u_{14} \oplus u_{16} = 0 \quad (17)$$

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

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

需要指出的是, K_1, \dots, K_4 是未知的, 而 K_{13}, \dots, 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, \dots, K_4 , 固定 K_{13}, \dots, 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 $K_1, \dots, K_8, K_{13}, \dots, K_{16}$ 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, \dots, 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:

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

计算得出偏差 $|\epsilon|$ 的值为:

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

现在, 我们已知子密钥 $K_1, \dots, K_8, K_{13}, \dots, K_{16}$ 。显然, 对于未知的子密钥 K_9, \dots, 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:

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

该逼近的偏差 $|\epsilon|$ 可以量化为:

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

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。

```

1 # =====Please modify the parameters here===== #
2     code
3 # =====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

```

1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4
5 using namespace std;
6
7 const int sbbox[] = {0xe, 4, 0xd, 1, 2, 0xf, 0xb, 8, 3, 0xa, 6, 0xc, 5, 9, 0, 7};
8 const int SIZE_SBOX = sizeof(sbbox) / sizeof(sbbox[0]);
9
10 // Compute the linear approximation for a given "input = output" equation
11 int linearApprox(int input_int, int output_int) {
12     int total = 0;
13     for (int ii = 0; ii < SIZE_SBOX; ++ii) {
14         int input_masked = ii & input_int;
15         int output_masked = sbbox[ii] & output_int;
16         if (__builtin_popcount(input_masked) % 2 == __builtin_popcount(
17             output_masked) % 2) {
18             total += 1;
19         }
20     }
21     return total;
22 }

```

```

18     }
19 }
20 return total - (SIZE_SBOX / 2);
21 }
22
23 int main() {
24     vector<int> linear_approx_table(SIZE_SBOX * SIZE_SBOX);
25
26     // Headers
27     cout << "      |";
28     for (int i = 0; i < SIZE_SBOX; ++i) {
29         cout << setw(4) << i;
30     }
31     cout << "\n" << string(SIZE_SBOX * 4 + 6, '-') << "\n";
32
33     for (int row = 0; row < SIZE_SBOX; ++row) {
34         cout << setw(4) << row << " |";
35         for (int col = 0; col < SIZE_SBOX; ++col) {
36             int r = linearApprox(row, col);
37             cout << setw(4) << r;
38             linear_approx_table[row * SIZE_SBOX + col] = r;
39         }
40         cout << "\n";
41     }
42
43     return 0;
44 }

```

明文密文对生成算法.ipynb

```

1 import random
2 import csv
3
4 # S-Box: Used for substitution operation, providing a non-linear substitution step.
5 S_Box = [0xE, 0x4, 0xD, 0x1, 0x2, 0xF, 0xB, 0x8, 0x3, 0xA, 0x6, 0xC, 0x5, 0x9, 0x0,
6           0x7]
7
8 # P-Box: Used for permutation operation, it rearranges the input bits.
9 P_Box = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16]
10
11 # Generate five 16-bit subkeys from a 32-bit master key.
12 def gen_K_list(K):
13     Ks = []
14     for _ in range(5):
15         Ks.append(K & 0xFFFF) # Get the last 16 bits of K
16         K >>= 4 # Right shift by 4 bits
17     return Ks[::-1]
18
19 # Perform substitution operation using the S-Box.
20 def pi_s(s_box, ur):
21     vr = 0
22     for i in range(4):
23         uri = ur & 0xF # Get the last 4 bits of ur
24         vri = s_box[uri] # Get the substitute value from the S-Box
25         vr += vri << (4 * i) # Store the result in vr

```

```

25         ur >>= 4 # Right shift by 4 bits
26     return vr
27
28 # Perform permutation operation using the P-Box.
29 def pi_p(p_box, vr):
30     wr = 0
31     for i in range(15, -1, -1):
32         vri = vr & 1 # Get the last bit of vr
33         vr >>= 1 # Right shift by 1 bit
34         wr |= vri << (16 - p_box[i]) # Rearrange bits based on P-Box value
35     return wr
36
37 # Implement the SPN encryption operation.
38 def do_SPN(x, s_box, p_box, Ks):
39     wr = x
40     # Execute Nr-1 rounds of encryption, each round includes three steps: Key
41     # addition, substitution, and permutation.
42     for r in range(3):
43         ur = wr ^ Ks[r] # Key addition
44         vr = pi_s(s_box, ur) # Substitution
45         wr = pi_p(p_box, vr) # Permutation
46     # The last round does not include permutation.
47     ur = wr ^ Ks[3] # Key addition
48     vr = pi_s(s_box, ur) # Substitution
49     return vr ^ Ks[4] # Output the result after another key addition
50
51 # Encrypt a 16-bit plaintext with a given 32-bit key.
52 def encrypt(K, x):
53     Ks = gen_K_list(K) # Generate subkeys
54     return do_SPN(x, S_Box, P_Box, Ks) # Execute SPN encryption
55
56 # Generate plaintext-ciphertext pairs
57 def generate_pairs():
58     # 0011 1010 1001 0100 1101 0110 0011 1111
59     K = int("00111010100101001101011000111111", 2)
60
61     pairs = []
62     # setting numbers here
63     for _ in range(10000):
64         plaintext = random.randint(0, 0xFFFF)
65         ciphertext = encrypt(K, plaintext)
66         pairs.append((format(plaintext, '016b'), format(ciphertext, '016b')))
67
68     return pairs
69
70 # Save the pairs to a CSV file
71 def save_to_csv(pairs, filename):
72     with open(filename, 'w', newline='') as csvfile:
73         fieldnames = ['Plaintext', 'Ciphertext']
74         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
75         writer.writeheader()
76         for p, c in pairs:
77             writer.writerow({'Plaintext': p, 'Ciphertext': c})
78
79 # Generate and save plaintext-ciphertext pairs to a CSV file

```



```

79 pairs = generate_pairs()
80 filename = "./data/明文_密文对.csv"
81 save_to_csv(pairs, filename)

```

线性分析算法.ipynb

```

1 import csv
2
3 def calculate_bias(K_values):
4     # Read the CSV file and extract the specified bits from the ciphertext
5     def extract_bits_from_ciphertext(filename):
6         extracted_data = []
7
8         with open(filename, 'r') as csvfile:
9             reader = csv.DictReader(csvfile)
10            for row in reader:
11                ciphertext = row['Ciphertext']
12                extracted_bits = {
13                    'Y5': ciphertext[4],
14                    'Y6': ciphertext[5],
15                    'Y7': ciphertext[6],
16                    'Y8': ciphertext[7],
17                    'Y13': ciphertext[12],
18                    'Y14': ciphertext[13],
19                    'Y15': ciphertext[14],
20                    'Y16': ciphertext[15],
21                }
22                extracted_data.append(extracted_bits)
23
24            # Write the extracted bits to a new CSV file
25            output_filename = "./data/特定比特位的密文.csv"
26            with open(output_filename, 'w', newline='') as csvfile:
27                fieldnames = ['Y5', 'Y6', 'Y7', 'Y8', 'Y13', 'Y14', 'Y15', 'Y16']
28                writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
29                writer.writeheader()
30                for row in extracted_data:
31                    writer.writerow(row)
32
33            return output_filename
34
35            # Extract the specified bits from the uploaded file and save to a new CSV file
36            output_file = extract_bits_from_ciphertext("./data/明文_密文对.csv")
37
38            # Define the default values for K bits
39
40
41            # Perform XOR operation on extracted bits and K values
42            def xor_operation(input_filename):
43                xor_results = []
44
45                with open(input_filename, 'r') as csvfile:
46                    reader = csv.DictReader(csvfile)
47                    for row in reader:
48                        xor_row = {
49                            'U5': str(int(row['Y5']) ^ int(K_values['K5'])),

```

```

50         'U6': str(int(row['Y6']) ^ int(K_values['K6'])),
51         'U7': str(int(row['Y7']) ^ int(K_values['K7'])),
52         'U8': str(int(row['Y8']) ^ int(K_values['K8'])),
53         'U13': str(int(row['Y13']) ^ int(K_values['K13'])),
54         'U14': str(int(row['Y14']) ^ int(K_values['K14'])),
55         'U15': str(int(row['Y15']) ^ int(K_values['K15'])),
56         'U16': str(int(row['Y16']) ^ int(K_values['K16']))
57     }
58     xor_results.append(xor_row)
59
60     # Write the XOR results to a new CSV file
61     output_filename = "./data/Y与K异或结果.csv"
62     with open(output_filename, 'w', newline='') as csvfile:
63         fieldnames = ['U5', 'U6', 'U7', 'U8', 'U13', 'U14', 'U15', 'U16']
64         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
65         writer.writeheader()
66         for row in xor_results:
67             writer.writerow(row)
68
69     return output_filename
70
71     # Perform XOR operation with the specified bits and save to a new CSV file
72     output_xor_file = xor_operation("./data/特定比特位的密文.csv")
73
74     # Convert binary values to hexadecimal and save to a new CSV file
75     def binary_to_hex(binary_str):
76         """Convert a binary string to a hexadecimal string."""
77         return format(int(binary_str, 2), 'x').upper()
78
79     def convert_and_save(input_filename):
80         hex_results = []
81
82         with open(input_filename, 'r') as csvfile:
83             reader = csv.DictReader(csvfile)
84             for row in reader:
85                 H1 = binary_to_hex(row['U5'] + row['U6'] + row['U7'] + row['U8'])
86                 H2 = binary_to_hex(row['U13'] + row['U14'] + row['U15'] + row['U16'])
87                 hex_row = {'H1': H1, 'H2': H2}
88                 hex_results.append(hex_row)
89
90         # Write the hexadecimal results to a new CSV file
91         output_filename = "./data/Y与K异或结果 (16进制).csv"
92         with open(output_filename, 'w', newline='') as csvfile:
93             fieldnames = ['H1', 'H2']
94             writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
95             writer.writeheader()
96             for row in hex_results:
97                 writer.writerow(row)
98
99         return output_filename
100
101     # Convert binary values to hexadecimal and save to a new CSV file
102     output_hex_file = convert_and_save("./data/Y与K异或结果.csv")
103

```

```

104
105 # Define the conversion mapping for H1 to X1
106 conversion_map = {
107     'E': '0', '4': '1', 'D': '2', '1': '3',
108     '2': '4', 'F': '5', 'B': '6', '8': '7',
109     '3': '8', 'A': '9', '6': 'A', 'C': 'B',
110     '5': 'C', '9': 'D', '0': 'E', '7': 'F'
111 }
112
113 # Convert H1 and H2 to X1 and X2 using the provided mapping
114 def convert_H_to_X(input_filename):
115     X_results = []
116
117     with open(input_filename, 'r') as csvfile:
118         reader = csv.DictReader(csvfile)
119         for row in reader:
120             X1 = conversion_map[row['H1']]
121             X2 = conversion_map[row['H2']]
122             X_row = {'X1': X1, 'X2': X2}
123             X_results.append(X_row)
124
125     # Write the X values to a new CSV file
126     output_filename = "./data/S盒逆置换结果（16进制）.csv"
127     with open(output_filename, 'w', newline='') as csvfile:
128         fieldnames = ['X1', 'X2']
129         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
130         writer.writeheader()
131         for row in X_results:
132             writer.writerow(row)
133
134     return output_filename
135
136 # Convert H1 and H2 to X1 and X2 using the provided mapping and save to a new
    CSV file
137 output_X_file = convert_H_to_X("./data/Y与K异或结果（16进制）.csv")
138
139 # Convert hexadecimal values to binary and save to a new CSV file
140 def hex_to_binary(hex_str):
141     """Convert a hexadecimal string to a binary string."""
142     return format(int(hex_str, 16), '04b')
143
144 # Convert X1 and X2 to binary and save to a new CSV file
145 def convert_X_to_u(input_filename):
146     u_results = []
147
148     with open(input_filename, 'r') as csvfile:
149         reader = csv.DictReader(csvfile)
150         for row in reader:
151             u5, u6, u7, u8 = tuple(hex_to_binary(row['X1']))
152             u13, u14, u15, u16 = tuple(hex_to_binary(row['X2']))
153             u_row = {
154                 'u5': u5, 'u6': u6, 'u7': u7, 'u8': u8,
155                 'u13': u13, 'u14': u14, 'u15': u15, 'u16': u16
156             }
157             u_results.append(u_row)

```

```

158
159     # Write the u values to a new CSV file
160     output_filename = "./data/S盒逆置换结果.csv"
161     with open(output_filename, 'w', newline='') as csvfile:
162         fieldnames = ['u5', 'u6', 'u7', 'u8', 'u13', 'u14', 'u15', 'u16']
163         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
164         writer.writeheader()
165         for row in u_results:
166             writer.writerow(row)
167
168     return output_filename
169
170 # Convert X1 and X2 to u values and save to a new CSV file
171 output_u_file = convert_X_to_u("./data/S盒逆置换结果 (16进制) .csv")
172
173 # Extract specified bits from plaintext and save to a new CSV file
174 def extract_bits_from_plaintext(filename):
175     extracted_data = []
176
177     with open(filename, 'r') as csvfile:
178         reader = csv.DictReader(csvfile)
179         for row in reader:
180             plaintext = row['Plaintext']
181             extracted_bits = {
182                 'x5': plaintext[4],
183                 'x7': plaintext[6],
184                 'x8': plaintext[7]
185             }
186             extracted_data.append(extracted_bits)
187
188 # Write the extracted bits to a new CSV file
189 output_filename = "./data/特定比特位的明文.csv"
190 with open(output_filename, 'w', newline='') as csvfile:
191     fieldnames = ['x5', 'x7', 'x8']
192     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
193     writer.writeheader()
194     for row in extracted_data:
195         writer.writerow(row)
196
197     return output_filename
198
199 # Extract specified bits from plaintext and save to a new CSV file
200 output_plaintext_bits_file = extract_bits_from_plaintext("./data/明文_密文对.
    csv")
201
202 # Merge extracted plaintext bits and u results, then calculate the XOR result
203 def merge_and_calculate_xor(plaintext_file, u_file):
204     with open(plaintext_file, 'r') as plaintext_csv, open(u_file, 'r') as u_csv
205         :
206         plaintext_reader = csv.DictReader(plaintext_csv)
207         u_reader = csv.DictReader(u_csv)
208
209     merged_data = []
210     for plaintext_row, u_row in zip(plaintext_reader, u_reader):
211         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

```

```
262
263     return count
264
265 # Count the total number of entries in the 'aws' column of the answer.csv file
266 total_count = count_total_entries_in_aws("./data/随机变量异或逻辑表.csv")
267
268 # Calculate the bias
269 bias = zero_count / total_count - 0.5
270
271 return bias
272
273 # Initialize the K_values dictionary
274 K_values = {
275     'K5': '0',
276     'K6': '0',
277     'K7': '0',
278     'K8': '0',
279     'K13': '0',
280     'K14': '0',
281     'K15': '0',
282     'K16': '0'
283 }
284
285 # Initialize a list to store biases for each iteration
286 biases = []
287
288 # Iterate through 256 values (0 to 255)
289 for i in range(256):
290     # Convert the current value of 'i' to an 8-bit binary string
291     binary_str = format(i, '08b')
292
293     # Update the K_values dictionary with the binary string
294     K_values['K5'] = binary_str[0]
295     K_values['K6'] = binary_str[1]
296     K_values['K7'] = binary_str[2]
297     K_values['K8'] = binary_str[3]
298     K_values['K13'] = binary_str[4]
299     K_values['K14'] = binary_str[5]
300     K_values['K15'] = binary_str[6]
301     K_values['K16'] = binary_str[7]
302
303     # Calculate the bias for the current K_values
304     bias = calculate_bias(K_values)
305
306     # Append the bias to the list
307     biases.append(bias)
308
309 # Create and write the results to a CSV file
310 output_filename = "./data/偏差统计结果.csv"
311 with open(output_filename, 'w', newline='') as csvfile:
312     fieldnames = ['Keys', 'Bias']
313     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
314     writer.writeheader()
315     for i, bias in enumerate(biases):
316         keys_binary = format(i, '08b')
```

```

317         writer.writerow({'Keys': keys_binary, 'Bias': abs(bias)})
318
319 print("Results saved to", output_filename)
320
321 # Load the biases from the CSV file
322 biases = []
323 with open("./data/偏差统计结果.csv", 'r') as csvfile:
324     reader = csv.DictReader(csvfile)
325     for row in reader:
326         # Parse the Keys and Bias columns from the CSV
327         keys = row['Keys']
328         bias = float(row['Bias'])
329         biases.append((keys, bias))
330
331 # Initialize variables to keep track of the closest bias and its corresponding keys
332 closest_bias = float('inf') # Initialize with positive infinity
333 closest_keys = None
334
335 # Iterate through the biases and find the closest one to 1/32
336 for keys, bias in biases:
337     # Calculate the absolute difference from 1/32
338     diff = abs(bias - 1/32)
339
340     # Check if this bias is closer than the previous closest
341     if diff < abs(closest_bias - 1/32):
342         closest_bias = bias
343         closest_keys = keys
344
345 # Print the keys and bias that are closest to 1/32
346 print(f"Keys: {closest_keys}")
347 print(f"Closest Bias: {closest_bias:.6f}")

```

进阶部分代码.ipynb

```

1 import csv
2
3 def calculate_bias(K_values):
4     # Read the CSV file and extract the specified bits from the ciphertext
5     def extract_bits_from_ciphertext(filename):
6         extracted_data = []
7
8         with open(filename, 'r') as csvfile:
9             reader = csv.DictReader(csvfile)
10            for row in reader:
11                ciphertext = row['Ciphertext']
12                extracted_bits = {
13
14                    # =====Please modify the parameters here===== #
15                    'Y1': ciphertext[0],
16                    'Y2': ciphertext[1],
17                    'Y3': ciphertext[2],
18                    'Y4': ciphertext[3],
19                    'Y13': ciphertext[12],
20                    'Y14': ciphertext[13],
21                    'Y15': ciphertext[14],

```

```

22         'Y16': ciphertext[15],
23         # =====Please modify the parameters here===== #
24
25     }
26     extracted_data.append(extracted_bits)
27
28     # Write the extracted bits to a new CSV file
29     output_filename = "./data/特定比特位的密文.csv"
30     with open(output_filename, 'w', newline='') as csvfile:
31
32         # =====Please modify the parameters here===== #
33         fieldnames = ['Y1', 'Y2', 'Y3', 'Y4', 'Y13', 'Y14', 'Y15', 'Y16']
34         # =====Please modify the parameters here===== #
35
36         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
37         writer.writeheader()
38         for row in extracted_data:
39             writer.writerow(row)
40
41     return output_filename
42
43     # Extract the specified bits from the uploaded file and save to a new CSV file
44     output_file = extract_bits_from_ciphertext("./data/明文_密文对.csv")
45
46     # Define the default values for K bits
47
48
49     # Perform XOR operation on extracted bits and K values
50     def xor_operation(input_filename):
51         xor_results = []
52
53         with open(input_filename, 'r') as csvfile:
54             reader = csv.DictReader(csvfile)
55             for row in reader:
56                 xor_row = {
57
58                     # =====Please modify the parameters here===== #
59                     'U1': str(int(row['Y1']) ^ int(K_values['K1'])),
60                     'U2': str(int(row['Y2']) ^ int(K_values['K2'])),
61                     'U3': str(int(row['Y3']) ^ int(K_values['K3'])),
62                     'U4': str(int(row['Y4']) ^ int(K_values['K4'])),
63                     'U13': str(int(row['Y13']) ^ int(K_values['K13'])),
64                     'U14': str(int(row['Y14']) ^ int(K_values['K14'])),
65                     'U15': str(int(row['Y15']) ^ int(K_values['K15'])),
66                     'U16': str(int(row['Y16']) ^ int(K_values['K16'])),
67                     # =====Please modify the parameters here===== #
68
69                 }
70                 xor_results.append(xor_row)
71
72     # Write the XOR results to a new CSV file
73     output_filename = "./data/Y与K异或结果.csv"
74     with open(output_filename, 'w', newline='') as csvfile:
75
76         # =====Please modify the parameters here===== #

```



```

77         fieldnames = ['U1', 'U2', 'U3', 'U4', 'U13', 'U14', 'U15', 'U16']
78         # =====Please modify the parameters here===== #
79
80         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
81         writer.writeheader()
82         for row in xor_results:
83             writer.writerow(row)
84
85         return output_filename
86
87     # Perform XOR operation with the specified bits and save to a new CSV file
88     output_xor_file = xor_operation("./data/特定比特位的密文.csv")
89
90     # Convert binary values to hexadecimal and save to a new CSV file
91     def binary_to_hex(binary_str):
92         """Convert a binary string to a hexadecimal string."""
93         return format(int(binary_str, 2), 'x').upper()
94
95     def convert_and_save(input_filename):
96         hex_results = []
97
98         with open(input_filename, 'r') as csvfile:
99             reader = csv.DictReader(csvfile)
100             for row in reader:
101
102                 # =====Please modify the parameters here===== #
103                 H1 = binary_to_hex(row['U1'] + row['U2'] + row['U3'] + row['U4'])
104                 H2 = binary_to_hex(row['U13'] + row['U14'] + row['U15'] + row['U16']
105                                     ])
106                 # =====Please modify the parameters here===== #
107
108                 hex_row = {'H1': H1, 'H2': H2}
109                 hex_results.append(hex_row)
110
111         # Write the hexadecimal results to a new CSV file
112         output_filename = "./data/Y与K异或结果(16进制).csv"
113         with open(output_filename, 'w', newline='') as csvfile:
114             fieldnames = ['H1', 'H2']
115             writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
116             writer.writeheader()
117             for row in hex_results:
118                 writer.writerow(row)
119
120         return output_filename
121
122     # Convert binary values to hexadecimal and save to a new CSV file
123     output_hex_file = convert_and_save("./data/Y与K异或结果.csv")
124
125     # Define the conversion mapping for H1 to X1
126     conversion_map = {
127         'E': '0', '4': '1', 'D': '2', '1': '3',
128         '2': '4', 'F': '5', 'B': '6', '8': '7',
129         '3': '8', 'A': '9', '6': 'A', 'C': 'B',
130         '5': 'C', '9': 'D', '0': 'E', '7': 'F'

```

```

131     }
132
133     # Convert H1 and H2 to X1 and X2 using the provided mapping
134     def convert_H_to_X(input_filename):
135         X_results = []
136
137         with open(input_filename, 'r') as csvfile:
138             reader = csv.DictReader(csvfile)
139             for row in reader:
140                 X1 = conversion_map[row['H1']]
141                 X2 = conversion_map[row['H2']]
142                 X_row = {'X1': X1, 'X2': X2}
143                 X_results.append(X_row)
144
145         # Write the X values to a new CSV file
146         output_filename = "./data/S盒逆置换结果 (16进制) .csv"
147         with open(output_filename, 'w', newline='') as csvfile:
148             fieldnames = ['X1', 'X2']
149             writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
150             writer.writeheader()
151             for row in X_results:
152                 writer.writerow(row)
153
154         return output_filename
155
156     # Convert H1 and H2 to X1 and X2 using the provided mapping and save to a new
157     # CSV file
158     output_X_file = convert_H_to_X("./data/Y与K异或结果 (16进制) .csv")
159
160     # Convert hexadecimal values to binary and save to a new CSV file
161     def hex_to_binary(hex_str):
162         """Convert a hexadecimal string to a binary string."""
163         return format(int(hex_str, 16), '04b')
164
165     # Convert X1 and X2 to binary and save to a new CSV file
166     def convert_X_to_u(input_filename):
167         u_results = []
168
169         with open(input_filename, 'r') as csvfile:
170             reader = csv.DictReader(csvfile)
171             for row in reader:
172                 u1, u2, u3, u4 = tuple(hex_to_binary(row['X1']))
173
174                 # =====Please modify the parameters here===== #
175                 u13, u14, u15, u16 = tuple(hex_to_binary(row['X2']))
176                 u_row = {
177                     'u1': u1, 'u2': u2, 'u3': u3, 'u4': u4,
178                     'u13': u13, 'u14': u14, 'u15': u15, 'u16': u16
179                 }
180                 # =====Please modify the parameters here===== #
181
182                 u_results.append(u_row)
183
184         # Write the u values to a new CSV file
185         output_filename = "./data/S盒逆置换结果.csv"

```

```

185     with open(output_filename, 'w', newline='') as csvfile:
186
187         # =====Please modify the parameters here===== #
188         fieldnames = ['u1', 'u2', 'u3', 'u4', 'u13', 'u14', 'u15', 'u16']
189         # =====Please modify the parameters here===== #
190
191         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
192         writer.writeheader()
193         for row in u_results:
194             writer.writerow(row)
195
196     return output_filename
197
198 # Convert X1 and X2 to u values and save to a new CSV file
199 output_u_file = convert_X_to_u("./data/S盒逆置换结果 (16进制) .csv")
200
201 # Extract specified bits from plaintext and save to a new CSV file
202 def extract_bits_from_plaintext(filename):
203     extracted_data = []
204
205     with open(filename, 'r') as csvfile:
206         reader = csv.DictReader(csvfile)
207         for row in reader:
208             plaintext = row['Plaintext']
209             extracted_bits = {
210
211                 # =====Please modify the parameters here===== #
212                 'x6': plaintext[5],
213                 'x7': plaintext[6]
214                 # =====Please modify the parameters here===== #
215
216             }
217             extracted_data.append(extracted_bits)
218
219 # Write the extracted bits to a new CSV file
220 output_filename = "./data/特定比特位的明文.csv"
221 with open(output_filename, 'w', newline='') as csvfile:
222
223     # =====Please modify the parameters here===== #
224     fieldnames = ['x6', 'x7']
225     # =====Please modify the parameters here===== #
226
227     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
228     writer.writeheader()
229     for row in extracted_data:
230         writer.writerow(row)
231
232     return output_filename
233
234 # Extract specified bits from plaintext and save to a new CSV file
235 output_plaintext_bits_file = extract_bits_from_plaintext("./data/明文_密文对.
    csv")
236
237 # Merge extracted plaintext bits and u results, then calculate the XOR result
238 def merge_and_calculate_xor(plaintext_file, u_file):

```

```

239     with open(plaintext_file, 'r') as plaintext_csv, open(u_file, 'r') as u_csv
240     :
241         plaintext_reader = csv.DictReader(plaintext_csv)
242         u_reader = csv.DictReader(u_csv)
243
244         merged_data = []
245         for plaintext_row, u_row in zip(plaintext_reader, u_reader):
246             # =====Please modify the parameters here===== #
247             xor_result = int(u_row['u2']) ^ int(u_row['u4']) ^ int(u_row['u14']
248                 ]) ^ int(u_row['u16']) ^ int(plaintext_row['x6']) ^ int(
249                 plaintext_row['x7'])
250             merged_row = {
251                 'x6': plaintext_row['x6'],
252                 'x7': plaintext_row['x7'],
253                 'u2': u_row['u2'],
254                 'u4': u_row['u4'],
255                 'u14': u_row['u14'],
256                 'u16': u_row['u16'],
257                 'aws': str(xor_result)
258             }
259             # =====Please modify the parameters here===== #
260
261             merged_data.append(merged_row)
262
263             # Write the merged data and XOR result to a new CSV file
264             output_filename = "./data/随机变量异或逻辑表.csv"
265             with open(output_filename, 'w', newline='') as csvfile:
266
267                 # =====Please modify the parameters here===== #
268                 fieldnames = ['x6', 'x7', 'u2', 'u4', 'u14', 'u16', 'aws']
269                 # =====Please modify the parameters here===== #
270
271                 writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
272                 writer.writeheader()
273                 for row in merged_data:
274                     writer.writerow(row)
275
276             return output_filename
277
278 # Merge extracted plaintext bits and u results, calculate XOR result, and save
279 # to a new CSV file
280 output_answer_file = merge_and_calculate_xor("./data/特定比特位的明文.csv", "./
281 data/S盒逆置换结果.csv")
282
283 # Count the number of zeros in the 'aws' column of the answer.csv file
284 def count_zeros_in_aws(filename):
285     count = 0
286
287     with open(filename, 'r') as csvfile:
288         reader = csv.DictReader(csvfile)
289         for row in reader:
290             if row['aws'] == '0':
291                 count += 1

```

```

289         return count
290
291     # Count the number of zeros in the 'aws' column of the answer.csv file
292     zero_count = count_zeros_in_aws("./data/随机变量异或逻辑表.csv")
293
294     # Count the total number of entries in the 'aws' column of the answer.csv file
295     def count_total_entries_in_aws(filename):
296         count = 0
297
298         with open(filename, 'r') as csvfile:
299             reader = csv.DictReader(csvfile)
300             for _ in reader:
301                 count += 1
302
303         return count
304
305     # Count the total number of entries in the 'aws' column of the answer.csv file
306     total_count = count_total_entries_in_aws("./data/随机变量异或逻辑表.csv")
307
308     # Calculate the bias
309     bias = zero_count / total_count - 0.5
310
311     return bias
312
313 # Initialize the K_values dictionary with K9 to K12 set to '1'
314
315 # =====Please modify the parameters here===== #
316 K_values = {
317     'K1': '0',
318     'K2': '0',
319     'K3': '0',
320     'K4': '0',
321     'K13': '1',
322     'K14': '1',
323     'K15': '1',
324     'K16': '1'
325 }
326 # =====Please modify the parameters here===== #
327
328 # Initialize a list to store biases for each iteration
329 biases = []
330
331 # Iterate through 16 values (0 to 15) for K1 to K4
332 # =====Please modify the parameters here===== #
333 for i in range(16):
334     # Convert the current value of 'i' to a 4-bit binary string
335     binary_str = format(i, '04b')
336
337     # Update the K_values dictionary with the binary string for K1 to K4
338     K_values['K1'] = binary_str[0]
339     K_values['K2'] = binary_str[1]
340     K_values['K3'] = binary_str[2]
341     K_values['K4'] = binary_str[3]
342
343     # Calculate the bias for the current K_values

```

```
344     bias = calculate_bias(K_values)
345
346     # Append the bias to the list
347     biases.append(bias)
348 # =====Please modify the parameters here===== #
349
350
351 # Create and write the results to a CSV file
352 output_filename = "./data/偏差统计结果.csv"
353 with open(output_filename, 'w', newline='') as csvfile:
354     fieldnames = ['Keys', 'Bias']
355     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
356     writer.writeheader()
357     for i, bias in enumerate(biases):
358         keys_binary = format(i, '04b') + '1111' # Appending '1111' for K9 to K12
359         writer.writerow({'Keys': keys_binary, 'Bias': abs(bias)})
360
361 # Print the output filename
362 output_filename
363
364 # Load the biases from the CSV file
365 biases = []
366 with open("./data/偏差统计结果.csv", 'r') as csvfile:
367     reader = csv.DictReader(csvfile)
368     for row in reader:
369         # Parse the Keys and Bias columns from the CSV
370         keys = row['Keys']
371         bias = float(row['Bias'])
372         biases.append((keys, bias))
373
374 # Initialize variables to keep track of the closest bias and its corresponding keys
375 closest_bias = float('inf') # Initialize with positive infinity
376 closest_keys = None
377
378 # Iterate through the biases and find the closest one to 9/256
379 for keys, bias in biases:
380     # Calculate the absolute difference from 9/256
381
382     diff = abs(bias - 9/256)
383
384     # Check if this bias is closer than the previous closest
385     if diff < abs(closest_bias - 9/256):
386         closest_bias = bias
387         closest_keys = keys
388
389 # Print the keys and bias that are closest to 9/256
390 print(f"Keys: {closest_keys}")
391 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.