开发手册

1.算法原理

Simplified DES方案，简称S-DES方案，是DES算法的简化版。它是一个供教学而非安全的加密算法，它与DES的特性和结构类似，但参数小。

加密算法涉及五个函数：

(1)初始置换IP(initial permutation)

(2)复合函数fk1，它是由密钥K确定的，具有置换和代换的运算。

(3)置换函数SW

(4)复合函数fk2

(5)初始置换IP的逆置换IP-1

# 通用置换函数  
def permute(input\_str, table):  
 output\_str = ""  
 for bit\_position in table:  
 output\_str += input\_str[bit\_position - 1]  
 return output\_str  
  
  
# 循环左移函数  
def ls(key, n):  
 # 将密钥分成两段并循环左移 n 位  
 left\_half = key[:5]  
 right\_half = key[5:]  
 shifted\_left = left\_half[n:] + left\_half[:n]  
 shifted\_right = right\_half[n:] + right\_half[:n]  
 return shifted\_left + shifted\_right  
  
  
# 子密钥生成  
def generate\_key(k, p10\_table, p8\_table):  
 # 执行 P10 置换  
 p10\_key = permute(k, p10\_table)  
 # 对结果进行左移操作和P8置换，得到 K1  
 k1 = permute(ls(p10\_key, 1), p8\_table)  
 # 再次对上一步结果进行左移操作h和P8置换，得到 K2  
 k2 = permute(ls(ls(p10\_key, 1), 2), p8\_table)  
 return k1, k2  
  
  
# S-DES 的 F 函数  
def F(right\_half, k):  
 # 对右半部分进行 E/P 扩展置换  
 expanded = permute(right\_half, ep\_table)  
 # 对结果与 K1 进行异或操作  
 xored = '{0:08b}'.format(int(expanded, 2) ^ int(k, 2))  
 # 将结果分为两组，并根据 S-box 进行替换  
 s0\_input = xored[:4]  
 s1\_input = xored[4:]  
 # 根据S盒规则行列查找  
 s0\_row = int(s0\_input[0] + s0\_input[-1], 2)  
 s0\_col = int(s0\_input[1:-1], 2)  
 s1\_row = int(s1\_input[0] + s1\_input[-1], 2)  
 s1\_col = int(s1\_input[1:-1], 2)  
 s0\_output = '{0:02b}'.format(sbox0[s0\_row][s0\_col])  
 s1\_output = '{0:02b}'.format(sbox1[s1\_row][s1\_col])  
 # 对两个输出串进行 P4 置换得到最终结果  
 s\_output = s0\_output + s1\_output  
 return permute(s\_output, p4\_table)  
  
  
# 加密过程  
def encrypt(p, k1, k2):  
 # 执行初始置换  
 p = permute(p, ip\_table)  
 # 进行两轮 Feistel 加密  
 l0 = p[:4]  
 r0 = p[4:]  
 l1 = r0  
 # 第一轮的P4  
 f\_result = F(r0, k1)  
 # p41和L0异或  
 r1 = '{0:04b}'.format(int(l0, 2) ^ int(f\_result, 2))  
 # 第二轮的P4  
 f\_result = F(r1, k2)  
 # p42和L1异或  
 r2 = '{0:04b}'.format(int(l1, 2) ^ int(f\_result, 2))  
 # 逆置换并返回结果(左边R2右边R1)  
 return permute(r2 + r1, ip\_ni\_table)  
  
  
# 解密过程  
def decrypt(c, k1, k2):  
 # 执行初始置换  
 c = permute(c, ip\_table)  
 # 进行两轮 Feistel 解密（注意子密钥的使用顺序）  
 r2 = c[:4]  
 l2 = c[4:]  
 # 第一轮的P4  
 f\_result = F(l2, k2)  
 # p41和R2异或  
 l1 = '{0:04b}'.format(int(r2, 2) ^ int(f\_result, 2))  
 # 第二轮的P4  
 f\_result = F(l1, k1)  
 # p42和R1异或  
 r1 = '{0:04b}'.format(int(l2, 2) ^ int(f\_result, 2))  
 # 逆置换并返回明文  
 return permute(r1 + l1, ip\_ni\_table)  
  
  
# 测试  
  
# 密钥key、明文p、各个置换、S盒  
# key = "1010000010"  
p10\_table = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)  
p8\_table = (6, 3, 7, 4, 8, 5, 10, 9)  
p4\_table = (2, 4, 3, 1)  
# p = "10110101"  
ip\_table = (2, 6, 3, 1, 4, 8, 5, 7)  
ep\_table = (4, 1, 2, 3, 2, 3, 4, 1)  
ip\_ni\_table = (4, 1, 3, 5, 7, 2, 8, 6)  
sbox0 = [  
 [1, 0, 3, 2],  
 [3, 2, 1, 0],  
 [0, 2, 1, 3],  
 [3, 1, 0, 2]  
]  
  
sbox1 = [  
 [0, 1, 2, 3],  
 [2, 3, 1, 0],  
 [3, 0, 1, 2],  
 [2, 1, 0, 3]  
]

2GUI支持和用户交互

创建窗口和按钮

# 创建窗口  
window = tk.Tk()  
l1 = tk.Label(window, text='明文（8位）:')  
l1.pack()  
# 创建输入框和标签  
entry1 = tk.Entry(window)  
entry1.pack()  
  
l2 = tk.Label(window, text='密钥（10位）:')  
l2.pack()  
entry2 = tk.Entry(window)  
entry2.pack()  
  
  
# 创建按钮  
button = tk.Button(window, text="加密（bit）", command=on\_button\_click)  
button.pack()  
  
button1 = tk.Button(window, text="加密（ACII）", command=on\_button1\_click)  
button1.pack()  
  
# 创建输出标签  
output\_label = tk.Label(window, text="结果：")  
output\_label.pack()  
  
output\_label1 = tk.Label(window, text="结果：")  
output\_label1.pack()  
  
# 运行窗口  
window.mainloop()

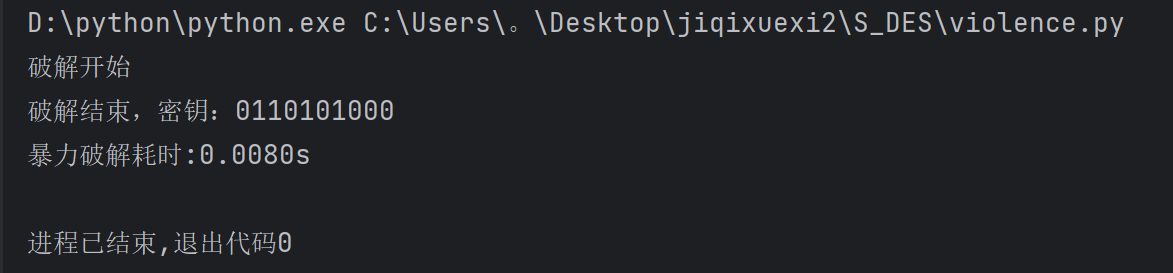


获取数据，输出结果，与用户交互

def on\_button1\_click():  
 # 获取用户输入的两个数据  
 num1 = str(entry1.get())  
 num2 = str(entry2.get())  
  
 # 调用函数进行计算  
 result = calculate1(num1, num2)  
  
 # 显示结果  
 output\_label1.config(text=f"结果：{result}")

3.暴力破解

i = 0  
p10\_table = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)  
p8\_table = (6, 3, 7, 4, 8, 5, 10, 9)  
t = "10101010" #明文  
m = "01101011" #目标密文  
print("破解开始")  
begin = time.time()  
  
  
  
  
while(1):  
 bit = bin(i).replace('0b', '').rjust(10,'0')  
 k1, k2 = generate\_key(bit, p10\_table, p8\_table)  
 result = encrypt(t, k1, k2)  
 if result == m:  
 break  
 # print("密匙"+ bit)  
 i = i + 1  
end = time.time()  
print("破解结束，密钥："+ bin(i).replace('0b', '').rjust(10,'0'))  
print(f'暴力破解耗时:{end - begin:.4f}s')



4.安全性讨论

虽然 S-DES 可以用于学习加密算法的入门课程，但它已经被证明不足以提供足够的安全性。具体来说，S-DES 存在以下安全问题：

密钥长度过短：S-DES 使用的密钥长度只有 10 位，这使得它容易受到暴力破解攻击。

线性结构：S-DES 的轮函数 F 是基于线性变换和置换操作的。由于线性变换可以被表示为矩阵乘法，因此 S-DES 可能会受到差分攻击和线性攻击等攻击方式的威胁。

小型 S 盒：S-DES 使用的 S 盒只有 2 个，每个 S 盒只有 4 个输入和 2 个输出。这使得 S-DES 更容易受到差分攻击和线性攻击等攻击方式的威胁。

以上安全问题都对 S-DES 的安全性产生了影响，因此在实际中不建议使用 S-DES 进行加密。如果需要更强大和安全的加密算法，建议使用更先进的加密算法，比如 AES（Advanced Encryption Standard）。