**实验三 Simple DES加密算法实现**

**一、实验目的**

1. 熟练掌握Simple DES（SDES）算法的加密与解密原理，理解其核心思想与操作流程。

2. 通过编程实现SDES算法，熟悉现代密码算法的基本设计思想和实现方法。

**二、实验内容**

1. **密钥生成**：

 输入一个10位的主密钥，生成两个子密钥*K1*和*K2*。

 显示密钥生成过程的中间结果，例如P10、P8和左移操作结果。

1. **加密实现**：

 输入一个8位的明文字符串。

 实现SDES加密算法，完整展示初始置换、两轮加密和逆初始置换的过程。

 输出最终的密文结果。

1. **解密实现**：

 输入生成的密文。

 使用SDES解密算法，验证解密结果与原明文是否一致。

 显示解密过程的中间结果。

1. **功能验证**：

 验证不同密钥对相同明文的加密效果，分析密钥的重要性。

 比较明文中单个位的变化如何影响密文（雪崩效应）。

**三、实验代码**

实验代码主要分为三类IP变化、显示类函数定义、定义加密解密方法。

1、首先此处定义了IP变化和S和变化的两个函数，用于实现P10、P8、P4等变化。

|  |
| --- |
| class IP\_box:  def \_\_init\_\_(self,In\_length,Out\_order):  self.In\_length = In\_length  self.Out\_order = Out\_order  def use(self,input):  out=[input[i-1]for i in self.Out\_order]  return out  class S\_box:  def \_\_init\_\_(self,table):  self.table = table  def use(self,input\_data):  *'''  根据S盒进行变换  :param input:4位输入  :return:2位输出  '''* row = int(input\_data[0] + input\_data[3], 2)  col = int(input\_data[1] + input\_data[2], 2)  output\_data = self.table[row][col]  return list(bin(output\_data)[2:].zfill(2)) |

接下来定义了密钥生成类，其过程实现主要如图所示。两次左移变化以及一次P8变化。

|  |
| --- |
| #子密钥生成 Key Generation class KeyGenerator:  def \_\_init\_\_(self):  self.P10 = IP\_box(10,[3,5,2,7,4,10,1,9,8,6])  self.P8 = IP\_box(10,[6,3,7,4,8,5,10,9])  self.LShift1= IP\_box(5,[2,3,4,5,1])  self.LShift2= IP\_box(5,[3,4,5,1,2])#左移2位   def use(self, key):  x = self.P10.use(key)  left, right = x[:5], x[5:]  # 第一次左移1位  left,right = self.LShift1.use(left),self.LShift1.use(right)  x1 = left + right  key1 = self.P8.use(x1)  # 第二次左移2位  left,right = self.LShift2.use(left),self.LShift2.use(right)  x2 = left + right  key2 = self.P8.use(x2)  return key1, key2 |

接下来是轮密钥部分，其主要就是一个S和的函数并针对输入的函数进行异或。

|  |
| --- |
| class RoundFunction:  def \_\_init\_\_(self):  self.EP=IP\_box(4,[4,1,2,3,2,3,4,1])  self.SBox0=S\_box([[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]])  self.SBox1=S\_box([[0,1,2,3],[2,0,1,3],[3,0,1,0],[2,1,0,3]])  self.P4=IP\_box(4,[2,4,3,1])   def xor\_list(self, l1, l2):  return [str(int(a) ^ int(b)) for a, b in zip(l1, l2)]   def use(self,input,subkey):  x=self.EP.use(input)  x\_str = "".join(x)  # print("EP后为:" +x\_str)  x=self.xor\_list(x\_str,subkey)  # print("异或后为:"+"".join(x))  xleft,xright=x[:4],x[4:]#取出左右部分   xleft=self.SBox0.use(xleft)  # print("S0和为:"+"".join(xleft))  xright=self.SBox1.use(xright)  # print("S1和为:"+"".join(xright))   x=self.P4.use(xleft+xright)  # print("P4后为:"+"".join(x))  return x |

2、接下来是工具类的两个方法定义，分别实现输入单个字母转化为ASCII码后转化为二进制，以及将二进制转化为ASCII码。其定义如下：

|  |
| --- |
| def char\_to\_binary(char: str) -> str:  *'''  将字符转为ASCII码后转化为二进制  :param char:字符  :return:二进制的字符  '''* if len(char) != 1:  raise ValueError("输入必须是单个字符")  # 获取 ASCII 码  ascii\_code = ord(char)  # 转换为 8 位二进制字符串  binary\_str = bin(ascii\_code)[2:].zfill(8)  return binary\_str  def binary\_to\_char(binary\_str):  ascii\_code = chr(int(binary\_str, 2))  return ascii\_code |

3、接下来就是加密类负责实现整个加密解密的过程，其代码如下：在编写的过程中添加了许多中间输出来验证输出的正确性。

|  |
| --- |
| class Simple\_DES:  def \_\_init\_\_(self):  self.keygenerate = KeyGenerator()  self.IP = IP\_box(8, [2,6,3,1,4,8,5,7])  self.IP\_inv = IP\_box(8, [4,1,3,5,7,2,8,6])  self.roundF = RoundFunction()   def xor\_list(self, l1, l2):  return [str(int(a) ^ int(b)) for a, b in zip(l1, l2)]   def encrypt(self, M, key):  key1, key2 = self.keygenerate.use(key)  print("生成的密钥1为:" + "".join(key1))  print("生成的密钥2为:" + "".join(key2))   x = self.IP.use(M)  print("IP变化后的数据为:" + "".join(x))  left, right = x[:4], x[4:]  #对右半部分进行处理  step=self.roundF.use(right, key1)  #进行异或  step1 = self.xor\_list(step, left)  print("异或结果为:" + "".join(step1))  # 注意 step1 = 左右拼接  step1 = step1 + right  print("拼接结果为:" + "".join(step1))  #开始第二次  #交换SW  left, right = step1[4:], step1[:4]  # 对右半部分进行处理  step = self.roundF.use(right, key2)  # 进行异或  step2 = self.xor\_list(step, left)  print("异或结果为:" + "".join(step2))  # 注意 step1 = 左右拼接  step2 = step2 + right  print("拼接结果为:" + "".join(step2))   out = self.IP\_inv.use(step2)  return out   def decrypt(self, C, key):  key1, key2 = self.keygenerate.use(key)  x = self.IP.use(C)  left, right = x[:4], x[4:]  step = self.roundF.use(right, key2)  step1 = self.xor\_list(step, left)  step1 = step1 + right  left, right = step1[4:], step1[:4]  step = self.roundF.use(right, key1)  step2 = self.xor\_list(step, left)  step2 = step2 + right  out = self.IP\_inv.use(step2)  return out |

4、最后则是主函数部分

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  C = input("请输入要加密的单个明文字符:")  C\_bin = char\_to\_binary(C)  print("要加密的明文为:" + C\_bin)   key = "1010000010"  print("密钥为:" + str(key))   sdes=Simple\_DES()  #加密  cipher\_text=sdes.encrypt(C\_bin,key)  print("加密的密文为:"+"".join(cipher\_text))  encrypt\_text=sdes.decrypt(cipher\_text,key)  print("解密的密文为:"+binary\_to\_char("".join(encrypt\_text))) |

文本

AI 生成的内容可能不正确。**四、实验结果**

实验验证选取了课堂上使用过的“C”的例子来进行验证，通过与计算过程的比对，验证了加密解密的可行性与正确性。

**五、实验总结与心得**

在本次实验中，其实花费了大量的时间在每个块编写完成后，的加解密代码的编写，值得注意的两个地方是1、在生成密钥的时候，第一次是左移1位第二次就是左移2位，这个点没注意到导致排查很久。2、在S和的时候，是head和end的和决定了行数，其余的和才决定列，这个知识点也是容易忘记的。

在代码实现部分，比较难受的是Python中的异或“^”,两个列表类型无法直接异或，需要转化为字符串后才能进行处理。

总体来说通过本次Simple\_DES的实验了解了在通过逆变化还原出明文的过程，当S和和IP变化固定时，只要掌握好Key就能实现两个人之间的信息传递。