# **用户指南 - S-DES系统**

## 1.系统简介

S-DES系统是一款结合DES加解密算法，专为用户设计的工具，旨在为您提供高效的加密与解密以及破解密码服务，使用界面简单直观，即使是小白也能快速上手。

## 2.环境配置

### 2.1安装Python

请确保您的计算机上已经安装了Python。您可以通过访问Python的官方网站来下载并安装。

### 2.2安装Pycharm

为了能顺利使用该工具，您需要借助Pycharm来运行项目。您可以通过访问Pycharm的官方网站来下载并安装。

## 启动系统

运行Pycharm,打开我们的S-DES项目，运行项目文件。

## 功能介绍

系统的功能主要分为：

* **二进制加密：**用户可以对二进制8bit明文进行加密。
* **二进制解密：**用户可以对二进制8bit密文进行解密。
* **ASCII加密：**用户可以对ASCII字符进行加密。
* **ASCII解密：**用户可以对ASCII字符进行解密。
* **暴力破解：**如果您忘记了密钥，可以使用这个功能获取密钥，并且系统能生成可能的所有密钥。

## 使用说明

### 8位二进制数加密

1. 在主界面中点击二进制加解密，定位到“加密”按钮
2. 在提供的输入框中，输入10bit二进制密钥和8bit二进制明文
3. 点击下方加密按钮
4. 在下方，您会看到将该明文加密得到的密文结果
5. 可点击刷新按钮，重新输入

### 8位二进制数解密

1. 在主界面中点击二进制加解密，定位到“解密”按钮
2. 在提供的输入框中，输入10bit二进制密钥和8bit二进制密文
3. 点击下方解密按钮
4. 在下方，您会看到将该密文解密得到的明文结果
5. 可点击刷新按钮，重新输入

### ASCII加密

1. 在主界面中点击ASCII加解密，定位到“加密”按钮
2. 在提供的输入框中，输入10bit二进制密钥和ASCII明文
3. 点击下方加密按钮
4. 在下方，您会看到将该明文加密得到的密文结果
5. 可点击刷新按钮，重新输入

### ASCII解密

1. 在主界面中点击ASCII加解密，定位到“解密”按钮
2. 在提供的输入框中，输入10bit二进制密钥和ASCII密文
3. 点击下方解密按钮
4. 在下方，您会看到将该密文解密得到的明文结果
5. 可点击刷新按钮，重新输入

### 暴力破解

1. 在主界面中点击暴力破解，输入明密文对
2. 得到可能的密钥和破解时间

# **开发手册**

## 引言

在现代信息安全中，加密技术是保护数据隐私的重要手段。AES（高级加密标准）和DES（数据加密标准）是最广泛使用的对称加密算法。本开发手册旨在为开发人员提供基于Python开发的S-DES加密算法应用程序的接口说明，适用于需要对S-DES加密算法进行集成或二次开发的开发人员。

## 技术栈

* **加密算法：**S-DES加密算法
* **数据库：**无（简单的文件存储）
* **编程语言**：Python

## 主要函数

# 初始化置换表和其他固定参数

def \_\_init\_\_(self):  
# 根据给定的置换规则重新排列位

def permute(self, bits, permutation):

#根据给定的顺序进行左移操作

def left\_shift(self, bits, shifts):

# 扩展密钥，生成两个子密钥 K1 和 K2

def key\_expansion(self, key):

# 将二进制位数组转换为整数

def bits\_to\_int(self, bits):

# 将整数转换为指定位数的二进制数组  
def int\_to\_bits(self, value, length):

# 使用 S-Box 进行置换  
def s\_box(self, bits, sbox):

# 轮函数 F  
def f\_function(self, bits, subkey):

# 加密函数  
def encrypt(self, plaintext, key):

# 解密函数  
def decrypt(self, ciphertext, key):

# 检查输入位是否合法（合法长度和位数）  
def is\_valid\_input(bits, expected\_length):

# 加密页面的UI设置

def setup\_encrypt\_tab(self)

# 解密页面的UI设置

def setup\_decrypt\_tab(self):

# 处理加密按钮的操作

def encrypt\_action(self):

# 处理解密按钮的操作

def decrypt\_action(self):

# 清空加密页面的输入框和结果标签

def clear\_encrypt\_fields(self):

# 清空解密页面的输入框和结果标签

def clear\_decrypt\_fields(self):

# 找到有效密钥

def find\_valid\_keys(self, pairs):

# 添加明文和密文对并开始暴力破解

def add\_pair(self):

# 显示错误提示消息框

def show\_error(self, message):

# 开始暴力破解

def start\_brute\_force(self):  
# 清空输入和结果。

def clear\_all(self):

# 加载并设置背景图像

def setup\_background(self):

# 设置主窗口的样式和按钮框

def setup\_window(self):

# 创建功能按钮

def create\_buttons(self):

# 打开加密和解密界面

def open\_encrypt\_decrypt(self):

# 打开ASCII加解密界面

def open\_ascii\_app(self):  
# 打开暴力破解界面

def open\_brute\_force(self):  
# 打开新窗口并加载相应的应用程序

def open\_new\_window(self, app\_class):

## 功能实现

#### 暴力破解

def find\_valid\_keys(self, pairs):  
 valid\_keys = [] # 用于存储有效密钥  
  
 for key\_candidate in range(1024): # 1024 = 2^10  
 key\_bits = self.sdes.int\_to\_bits(key\_candidate, 10) # 将密钥转换为10位二进制列表  
 matches = True # 假设密钥是有效的  
  
 # 检查所有的明密文对  
 for plaintext, ciphertext in pairs:  
 encrypted\_result = self.sdes.encrypt(plaintext, key\_bits)  
 # 如果任何一个加密结果不匹配，设置为 False  
 if encrypted\_result != ciphertext:  
 matches = False  
 break  
  
 if matches: # 如果所有明密文对都匹配  
 valid\_keys.append(key\_bits) # 添加到有效密钥列表  
  
 return valid\_keys

#### 加解密算法实现

"""根据给定的置换规则重新排列位"""  
def permute(self, bits, permutation):  
 return [bits[i - 1] for i in permutation] # 根据置换表返回重新排列的位  
  
"""根据给定的顺序进行左移操作"""  
def left\_shift(self, bits, shifts):  
 return [bits[i - 1] for i in shifts] # 通过索引进行左移位  
  
"""扩展密钥，生成两个子密钥 K1 和 K2"""  
def key\_expansion(self, key):  
 # 使用 P10 置换密钥   
 permuted\_key = self.permute(key, self.P10)  
 left\_part = permuted\_key[:5] # 密钥的左半部分  
 right\_part = permuted\_key[5:] # 密钥的右半部分   
  
 # 左移1位   
 left\_part\_shifted1 = self.left\_shift(left\_part, self.LS1)  
 right\_part\_shifted1 = self.left\_shift(right\_part, self.LS1)  
 K1 = self.permute(left\_part\_shifted1 + right\_part\_shifted1, self.P8) # 生成K1   
  
 # 左移2位   
 left\_part\_shifted2 = self.left\_shift(left\_part\_shifted1, self.LS2)  
 right\_part\_shifted2 = self.left\_shift(right\_part\_shifted1, self.LS2)  
 K2 = self.permute(left\_part\_shifted2 + right\_part\_shifted2, self.P8) # 生成K2   
  
 return K1, K2 # 返回两个子密钥   
  
"""将二进制位数组转换为整数"""  
  
def bits\_to\_int(self, bits):  
 return int("".join(map(str, bits)), 2) # 将位数组逐位拼成字符串并转换为整数   
  
"""将整数转换为指定位数的二进制数组"""  
def int\_to\_bits(self, value, length):  
 return [int(bit) for bit in bin(value)[2:].zfill(length)] # 转换为二进制并填充至指定长度   
  
"""使用 S-Box 进行置换"""  
def s\_box(self, bits, sbox):  
 row = self.bits\_to\_int([bits[0], bits[3]]) # S-Box 行由第一个和最后一个二进制位确定   
 col = self.bits\_to\_int([bits[1], bits[2]]) # S-Box 列由中间两个二进制位确定   
 return self.int\_to\_bits(sbox[row][col], 2) # 返回 S-Box 经过置换的结果（2位二进制）   
  
"""轮函数 F"""  
def f\_function(self, bits, subkey):  
 expanded\_bits = self.permute(bits, self.EPBox) # 扩展输入位   
 xor\_result = [bit ^ key\_bit for bit, key\_bit in zip(expanded\_bits, subkey)] # 与子密钥进行异或操作   
 left\_sbox\_result = self.s\_box(xor\_result[:4], self.SBox1) # 对左半部进行 S-Box 置换   
 right\_sbox\_result = self.s\_box(xor\_result[4:], self.SBox2) # 对右半部进行 S-Box 置换   
 sbox\_result = left\_sbox\_result + right\_sbox\_result # 连接 S-Box 的结果   
 return self.permute(sbox\_result, self.P4) # 最后进行 P4 置换后返回   
  
"""加密函数"""  
def encrypt(self, plaintext, key):  
 # 生成两个子密钥   
 K1, K2 = self.key\_expansion(key)  
 permuted\_plaintext = self.permute(plaintext, self.IP) # 进行初始置换   
 left\_part = permuted\_plaintext[:4] # 左半部分   
 right\_part = permuted\_plaintext[4:] # 右半部分   
  
 # 第一轮   
 f\_output1 = self.f\_function(right\_part, K1) # 调用 F 函数   
 left\_xor\_f1 = [l\_bit ^ f1\_bit for l\_bit, f1\_bit in zip(left\_part, f\_output1)] # 左半部与 F 输出进行异或   
 left\_part, right\_part = right\_part, left\_xor\_f1 # 交换部分   
  
 # 第二轮   
 f\_output2 = self.f\_function(right\_part, K2) # 调用 F 函数   
 left\_xor\_f2 = [l\_bit ^ f2\_bit for l\_bit, f2\_bit in zip(left\_part, f\_output2)] # 左半部与 F 输出进行异或   
 final\_bits = left\_xor\_f2 + right\_part # 拼接成加密后的位   
  
 return self.permute(final\_bits, self.IP\_inv) # 进行逆初始置换，返回加密结果   
  
"""解密函数"""  
def decrypt(self, ciphertext, key):  
 # 生成两个子密钥  
 K1, K2 = self.key\_expansion(key)  
 permuted\_ciphertext = self.permute(ciphertext, self.IP) # 进行初始置换   
 left\_part = permuted\_ciphertext[:4] # 左半部分   
 right\_part = permuted\_ciphertext[4:] # 右半部分   
  
 # 第一轮（使用 K2）   
 f\_output1 = self.f\_function(right\_part, K2) # 调用 F 函数   
 left\_xor\_f1 = [l\_bit ^ f1\_bit for l\_bit, f1\_bit in zip(left\_part, f\_output1)] # 左半部与 F 输出进行异或   
 left\_part, right\_part = right\_part, left\_xor\_f1 # 交换部分   
  
 # 第二轮（使用 K1）   
 f\_output2 = self.f\_function(right\_part, K1) # 调用 F 函数   
 left\_xor\_f2 = [l\_bit ^ f2\_bit for l\_bit, f2\_bit in zip(left\_part, f\_output2)] # 左半部与 F 输出进行异或   
 final\_bits = left\_xor\_f2 + right\_part # 拼接成解密后的位   
  
 return self.permute(final\_bits, self.IP\_inv) # 进行逆初始置换，返回解密结果

## 性能优化

1. 可以考虑使用更高效的算法或数据结构来实现置换、S 盒替换等操作，以提高加密和解密的速度。
2. 对于暴力破解功能，可以考虑使用更优化的搜索算法，以减少破解时间。

## 安全考虑

1. S-DES 是一种相对简单的加密算法，安全性有限。不建议在对安全性要求较高的场景中单独使用。
2. 在使用暴力破解功能时，应注意其计算复杂度可能较高，尤其是对于较长的明文和密文。
3. 虽然使用了加密算法，但在实际应用中，应注意保护密钥的安全，避免在网络传输中泄露。
4. 对于暴力破解功能，应考虑限制其使用频率，以防止恶意攻击。