1. 简明的介绍和概述：

欢迎使用 S-DES 加密软件用户指南。本软件是一个用于对文本进行加密和解密的工具，基于简化数据加密标准（S-DES）算法。它可以帮助您保护敏感信息，确保您的数据在传输和存储时得到安全保护。

2. 系统要求和安装：

系统要求：
- 操作系统：支持 Windows、macOS 和 Linux。
- Python：需要安装 Python 解释器。
- Tkinter：需要安装 Tkinter 库。
- pyperclip：需要安装 pyperclip 库。

安装步骤：
1. 下载软件源代码到您的计算机。
2. 安装 Python 解释器（如果尚未安装）。
3. 安装 Tkinter 库（如果尚未安装）。
4. 安装 pyperclip 库（如果尚未安装）。
5. 运行源代码。

3. 用户界面和基本操作：

用户界面
用户界面简单易用，包括功能选择界面和操作界面。
功能选择包括二进制数加解密 GUI 按钮、ASCLL 码加解密 GUI 按钮以及中文加解密 GUI 按钮。
操作界面包括随机生成密钥按钮、文本输入框、密钥输入框、加密和解密按钮，以及结果显示区域，其中二进制数加解密 GUI 还有测试按钮。
基本操作
- 在功能选择界面选择二进制数加解密 GUI、ASCLL 码加解密 GUI 或是中文加解密 GUI 以进入操作界面。
- 点击随机生成密钥按钮生成随机密钥。（可选）
- 在文本输入框中输入要加密或解密的文本。
- 在密钥输入框中输入 10 位的二进制密钥。
- 点击加密按钮执行加密操作。
- 点击解密按钮执行解密操作。
- 结果将显示在结果区域。

4. 加密和解密流程：

加密和解密流程如下：
- 输入明文文本或密文文本。
- 输入 10 位的二进制密钥。
- 点击加密按钮以加密文本，或点击解密按钮以解密文本。

算法描述：

加密算法：C=IP^{-1}(f_{k_{2}}(SW(f_{k_{1}}(IP(P)))))

解密算法：P=IP^{-1}(f_{k_{1}}(SW(f_{k_{2}}(IP(C)))))

密钥扩展：k_{i}=P_{8}(Shift^{i}(P_{10}(K))), (i=1,2)

转换装置设定：

密钥扩展置

- P_{10}=(3,5,2,7,4,10,1,9,8,6)
- P_{8}=(6,3,7,4,8,5,10,9)
- Left_Shift^1=(1,1)

初始置换盒

- IP=(2,6,3,1,4,8,5,7)

最终置换盒

- IP^{-1}=(4,1,3,5,7,2,8,6)

轮函数 F

- EPBox=(4,1,2,3,2,3,4,1)
- SBox_{1}=[(1,0,3,2);(3,2,1,0);(0,2,1,3);(3,1,0,2)]
- SBox_{2}=[(0,1,2,3);(2,3,1,0);(3,0,1,2);(2,1,0,3)]
- SPBox=(2,4,3,1)

部分代码：

```
def initial_permutation(data):
    permuted_data = [data[ip - 1] for ip in IP]
    return permuted_data

# 对 8 位数据进行逆初始置换（IP^-1）
def inverse_initial_permutation(data):
    permuted_data = [data[ip - 1] for ip in IP_INV]
    return permuted_data

# 将 4 位数据扩展为 8 位数据（使用扩展表 E）
def expand(data):
    expanded_data = [data[e - 1] for e in E]
    return expanded_data

# 对两个 8 位数据块进行异或运算
def xor(data1, data2):
    result = []
    for i in range(len(data1)):
        result.append(data1[i] ^ data2[i])
```

```
    return result

# 将一个 8 位数据块分成两个 4 位数据块
def split(data):
    split_index = len(data) // 2
    first_half = data[:split_index]
    second_half = data[split_index:]

    return first_half, second_half

# 使用 S 盒对 4 位数据块进行替代
def s_box(data, s_box_table):
    row = int(''.join(map(str, [data[0], data[3]])), 2)
    col = int(''.join(map(str, data[1:3])), 2)
    return [int(x) for x in format(s_box_table[row][col], '02b')]

# 使用给定的置换表对 4 位数据块进行置换
def permute(data, perm_table):
    return [data[p - 1] for p in perm_table]

# 从 10 位密钥生成轮密钥
def generate_round_keys(key):
    key = permute(key, P10)
    left, right = split(key)

    round_keys = []
    for i in range(2):
        left = left[LS_1[i]:] + left[:LS_1[i]]
        right = right[LS_1[i]:] + right[:LS_1[i]]
        round_key = permute(left + right, P8)
        round_keys.append(round_key)

    return round_keys

# 执行 F 函数（Feistel 函数）
def feistel(data, sub_key):
    data = expand(data)
    data = xor(data, sub_key)
    left, right = split(data)
    left = s_box(left, S0)
    right = s_box(right, S1)
    data = permute(left + right, P4)
    return data
```

```
# 使用 10 位密钥加密 8 位明文
def encrypt(plaintext, key):
    round_keys = generate_round_keys(key)
    plaintext = initial_permutation(plaintext)
    left, right = split(plaintext)

    for i in range(2):
        left, right = right, xor(left, feistel(right,
round_keys[i]))

    ciphertext = inverse_initial_permutation(right + left)
    return ciphertext

# 使用 10 位密钥解密 8 位密文
def decrypt(ciphertext, key):
    round_keys = generate_round_keys(key)
    ciphertext = initial_permutation(ciphertext)
    left, right = split(ciphertext)

    for i in range(2):
        left, right = right, xor(left, feistel(right, round_keys[1
- i]))

    plaintext = inverse_initial_permutation(right + left)
    return plaintext

# 将二进制字符串转换为整数列表
def binary_string_to_list(binary_string):
    return [int(bit) for bit in binary_string]

# 将整数列表转换为二进制字符串
def list_to_binary_string(data_list):
    return ''.join(map(str, data_list))

# 生成一个随机的 10 位密钥
def generate_random_key():
    random_numbers = [random.randint(0, 1) for _ in range(10)]
    random_key = ''.join(map(str, random_numbers))  # 将随机数列表转
换为字符串
    pyperclip.copy(random_key)  # 将生成的随机数复制到剪贴板
    return random_key

# 将 ASCII 字符转换为 8 位的二进制字符串
def ascii_to_binary(text):
```

```
    binary_text = ''.join(format(ord(char), '08b') for char in
text)
    return binary_text

# 将 8 位的二进制字符串转换为 ASCII 字符
def binary_to_ascii(binary_text):
    ascii_text = ''.join(chr(int(binary_text[i:i+8], 2)) for i in
range(0, len(binary_text), 8))
    return ascii_text

# 将 ASCII 编码的字符串分组成 8 位数据块，然后加密
def encrypt_ascii_text(ascii_text, key):
    binary_text = ascii_to_binary(ascii_text)
    encrypted_binary_text = ''

    for i in range(0, len(binary_text), 8):
        plaintext_block = list(map(int, binary_text[i:i+8]))
        ciphertext_block = encrypt(plaintext_block, key)
        encrypted_binary_text += ''.join(map(str,
ciphertext_block))

    return binary_to_ascii(encrypted_binary_text)

# 将 ASCII 编码的字符串分组成 8 位数据块，然后解密
def decrypt_ascii_text(encrypted_ascii_text, key):
    encrypted_binary_text = ascii_to_binary(encrypted_ascii_text)
    decrypted_binary_text = ''

    for i in range(0, len(encrypted_binary_text), 8):
        ciphertext_block = list(map(int,
encrypted_binary_text[i:i+8]))
        plaintext_block = decrypt(ciphertext_block, key)
        decrypted_binary_text += ''.join(map(str, plaintext_block))

    return binary_to_ascii(decrypted_binary_text)

def chinese_to_unicode_binary(text):
    binary_text = ''.join(format(ord(char), '016b') for char in
text)
    return binary_text

# 将 Unicode 编码的二进制字符串转换为中文文本
def unicode_binary_to_chinese(binary_text):
    chinese_text = ''.join(chr(int(binary_text[i:i+16], 2)) for i
```

```
in range(0, len(binary_text), 16))
    return chinese_text

# 将中文文本分组成 8 位数据块，然后加密
def encrypt_chinese_text(chinese_text, key):
    unicode_binary_text = chinese_to_unicode_binary(chinese_text)
    encrypted_binary_text = ''

    for i in range(0, len(unicode_binary_text), 8):
        plaintext_block = list(map(int,
unicode_binary_text[i:i+8]))
        ciphertext_block = encrypt(plaintext_block, key)
        encrypted_binary_text += ''.join(map(str,
ciphertext_block))

    return unicode_binary_to_chinese(encrypted_binary_text)

# 将中文文本分组成 8 位数据块，然后解密
def decrypt_chinese_text(encrypted_chinese_text, key):
    encrypted_binary_text =
chinese_to_unicode_binary(encrypted_chinese_text)
    decrypted_binary_text = ''

    for i in range(0, len(encrypted_binary_text), 8):
        ciphertext_block = list(map(int,
encrypted_binary_text[i:i+8]))
        plaintext_block = decrypt(ciphertext_block, key)
        decrypted_binary_text += ''.join(map(str, plaintext_block))

    return unicode_binary_to_chinese(decrypted_binary_text)

def encrypt_binary_text(binary_text, key):
    encrypted_binary_text = ''

    for i in range(0, len(binary_text), 8):
        plaintext_block = list(map(int, binary_text[i:i + 8]))
        ciphertext_block = encrypt(plaintext_block, key)
        encrypted_binary_text += ''.join(map(str,
ciphertext_block))

    return encrypted_binary_text


def decrypt_binary_text(binary_text, key):
```

```
    decrypted_binary_text = ''

    for i in range(0, len(binary_text), 8):
        ciphertext_block = list(map(int, binary_text[i:i + 8]))
        plaintext_block = decrypt(ciphertext_block, key)
        decrypted_binary_text += ''.join(map(str, plaintext_block))

    return decrypted_binary_text

# 测试部分
def test_encrypt():
    plain = '01010101'
    cipher = '00000100'
    found_keys = []
    key_search = None

    start_time = time.time()  # 记录开始时间

    for i in range(1024):
        cipher_text = ''
        key = format(i, '010b')  # 将整数转换为10位的二进制字符串
        key = list(map(int, key))
        cipher_text = encrypt_binary_text(plain, key)

        if cipher_text == cipher:
            found_keys.append(key)

    end_time = time.time()  # 记录结束时间
    execution_time = end_time - start_time  # 计算执行时间

    if found_keys:
        print("Found Keys:")
        for idx, key_found in enumerate(found_keys, start=1):
            print(f"Key {idx}: {key_found}")
    else:
        print("Keys not found")

    print("Execution Time:", execution_time, "seconds")
```

5. 安全性和最佳实践：

数据安全至关重要。请遵循以下最佳实践：
- 不要分享密钥或密码。

- 使用强密码来保护您的密钥。
- 将密钥存储在安全的地方。

6. 高级功能和设置：

本软件提供高级功能和设置，支持生成随机密钥，支持对二进制数、ASCII 码以及中文进行加密或解密。

7. 错误处理和故障排除：

常见错误消息和问题的解决方法将在本指南中列出，以帮助您在遇到问题时解决它们。

8. 法律和合规性：

请确保您在使用本软件时遵守适用的法律和合规性要求。我们不对用户的非法或不当使用负责。

9. 更新和支持：

我们会不定期地在 Github 上对该软件进行更新。如果您需要技术支持或有反馈，请联系我们的支持团队。

10. 示例和案例：

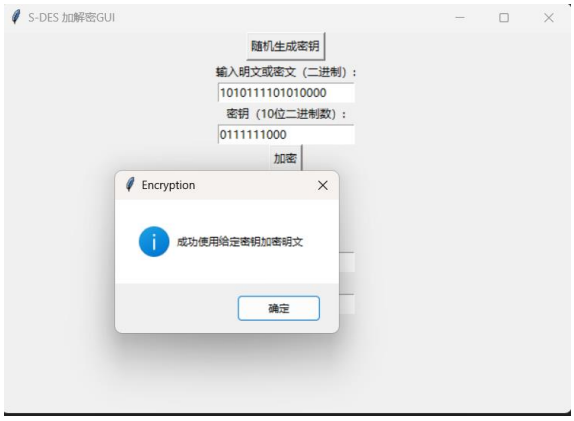
为了帮助您更好地理解软件的用途，我们提供了示例和案例，以展示不同情境下的使用方式。

--运行程序，进入功能选择界面

--点击二进制加解密 GUI，进入操作界面（二进制加解密 GUI）

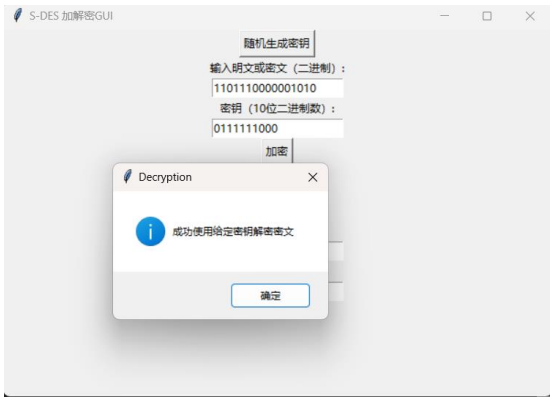
--点击随机生成密钥（密钥已复制到剪切板，直接粘贴到密钥输入框中即可）

--将密钥粘贴到密钥输入框，在明文或密文输入框中输入将要加密的明文(位数须被 8 整除)，点击加密按钮



--得到加密结果，显示在密文显示框中

--对密文进行解密同样如此，在明文或密文输入框中输入将要解密的密文，点击解密按钮



--得到解密结果，显示在明文显示框中

--点击测试按钮，可以找到方法内给定明密文对所对应的密钥，并显示验证密钥数量和查找所花费的时间（可在 test_encrypt()方法内修改明密文对，仍可查找其对应的密钥）

```
Found Keys:
Key 1: [0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
Key 2: [0, 1, 1, 0, 0, 0, 1, 1, 1, 0]
Key 3: [1, 0, 1, 1, 0, 1, 1, 0, 0, 0]
Key 4: [1, 1, 1, 1, 0, 1, 1, 0, 0, 0]
Execution Time: 0.026013851165771484 seconds
```

--同理，在功能选择界面点击 ASCII 加解密 GUI 按钮，进入操作界面（ASCII 加解密 GUI）

--在密钥输入框中输入密钥（可随机生成），并在明文或密文输入框中输入“adwxx”，点击加密按钮，得到密文“µ©,ÉÉ”

--在明文或密文输入加密后得到的密文“µ©,ÉÉ”，保持密钥不变，点击解密按钮，得到明文“adwxx”显示在明文显示框中

--同理，在功能选择界面点击中文加解密 GUI 按钮，进入操作界面（中文加解密 GUI）

--在密钥输入框中输入密钥（可随机生成），并在明文或密文输入框中输入“祝你中秋国庆快乐”，点击加密按钮，得到密文“쾵≈앎쾵꒷偃繩엀”，显示在密文显示框中

--在明文或密文输入加密后得到的密文“쮙≈앎쮕ᵀᵀ傯繩엀”，保持密钥不变，点击解密按钮，得到明文“祝你中秋国庆快乐”显示在明文显示框中

11. 安全注意事项：

强调用户不应分享密钥、密码或加密数据，以及在使用软件时应保持警惕。

12. 测试和校对：

我们经过充分的测试和校对，确保用户指南没有拼写或语法错误。我们的目标是提供清晰和准确的指南，以帮助用户轻松使用本软件。

希望这份用户指南能够帮助您了解并正确使用 S-DES 加密软件。如果您有任何问题或需要进一步的帮助，请随时联系我们的支持团队。