开发手册&用户指南

—— written by 荔枝

1. 简介

1.1 系统概述

该系统是我们“荔枝”团队为用户设计的一款简单、开源的 AES 算法加解密系统，能够满足用户简单的加解密需要（二进制加解密、ASCII 码字符串加解密、多重加解密、文件加解密），同时可以向用户展示对于双重加密 AES 算法的有效破解方案——中间相遇攻击问题。

同时，也欢迎广大程序设计与编码工作者继续完善该加密系统。如有任何疑问和建议，欢迎联系我们团队（Email：1635487611@qq.com）。

1.2 主要功能和特点

该系统主要功能包括对 16 bit 二进制数进行 AES 算法的加解密，对 2 bytes ASCII 码进行 AES 算法的加解密、双重加解密、三重加解密、文件加密、CBC 模式加解密、根据明密文对进行中间相遇攻击破解计算等。

该系统的特点在于简单易懂、容易上手，可以帮助不太了解 AES 算法的用户快速获取加解密结果以及相应的密钥信息，也可以帮助正在学习 AES 算法学生更直观的接触到简单的 AES 应用，感受 AES 加密算法的伟大和奇妙之处。

1.3 目标用户群体

初步学习和希望了解 AES 算法的用户，希望使用 Simple-AES 算法进行加解密的用户。

1.4 使用说明

该系统可以在浏览器运行，无需特殊的环境配置。具体页面及功能详见下文。

2. 开发者团队

2.1 开发者队名：荔枝

2.2 开发者姓名：张芷芮、刘俐莹

2.3 联系信息：1635487611@qq.com

3. 开发环境

3.1 开发工具和技术栈

3.1.1 开发工具

- 集成开发化境（IDE）：PyCharm 2023.3
- 版本控制：Github

3.1.2 技术栈

- 前端技术：语言：HTML5，CSS3，JavaScript

  框架和库：jQuery
- 后端技术：语言：Python

  框架：Flask 框架
- Web 浏览器：Microsoft Edge

3.2 运行

python 文件在根目录下运行：main.py

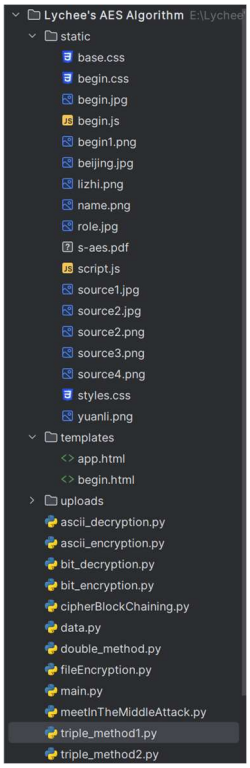
3.3 网址与端口号

该系统在 http://127.0.0.1:6606 上运行

4. 代码结构

4.1 项目的目录结构

项目的目录结构图如下：



- static 文件夹中存放图片、js 文件和 css 样式表等

  （begin.html：启动页面；app.html：主页面）
- templates 文件夹中存放 html 文件
- data.py 文件中存放标准加解密数据和几个基础函数（密钥扩展函数、轮密钥加函数、半字节代替函数、行移位函数、列混淆函数）
- main.py 文件中主要存放连接前后端的路由
- bit_encryption.py 文件中主要存放二进制加密算法
- bit_decryption.py 文件中主要存放二进制解密算法
- ascii_encryption.py 文件中主要存放 ASCII 码的加密算法
- ascii_decryption.py 文件中主要存放 ASCII 码的解密算法
- double_method.py 文件中主要存放两种双重加密算法和普通破解方式的解密算法
- triple_method1.py 文件中主要存放三重加解密算法（其中密钥长度为 32 bit）
- triple_method2.py 文件中主要存放两种三重加解密算法（其中密钥长度为 48 bit）
- meetInTheMiddleAttack.py 文件中存放中间相遇攻击算法
- fileEncryption.py 文件中存放文件加密算法
- cipherBlockChaining.py 文件中存放 CBC 模式加密算法

4.2 关键组件和模块的描述

4.2.1 密钥扩展函数

1、原理：

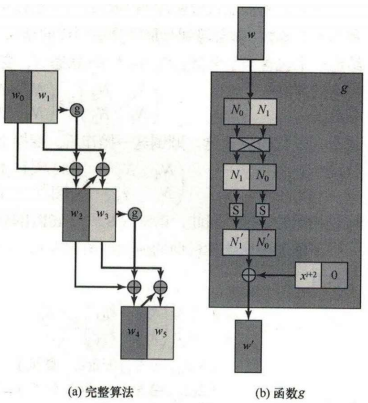


图 D.5 S-AES 密钥扩展

2、代码演示：

密钥扩展操作是根据现有输入的 16 bit 密钥，分为左右各 8 bit 的两部分（初始为

w_0 w_1），通过 g（x）函数和亦或运算，生成 w_2 w_3 w_4 w_5，合并为轮密钥 key2 key3，实现将一个 16 bit 的密钥扩展为 3 个。

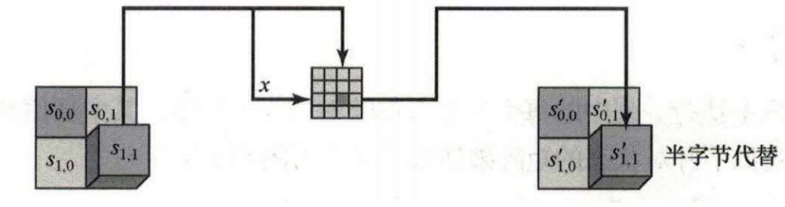
具体代码逻辑如下所示：

```
# 密钥扩展：交换N0，N1.将一个字节的半字节进行循环移位
2 个用法
def circular_shift(word):
    return ((word & 0x0F) << 4) + ((word & 0xF0) >> 4)


# 密钥扩展：用S-BOX替换N0、N1。使用S盒替换一个字节的高4位和低4位
2 个用法
def key_substitution(word, sbox):
    return (sbox[(word >> 4)] << 4) + sbox[word & 0x0F]
```

```
# 密钥扩展
2 个用法
def key_expansion(key):
    w = [None] * 6
    w[0] = (key & 0xFF00) >> 8
    w[1] = key & 0x00FF
    w[2] = w[0] ^ (key_substitution(circular_shift(w[1]), S_BOX) ^ RCON1)
    w[3] = w[2] ^ w[1]
    w[4] = w[2] ^ (key_substitution(circular_shift(w[3]), S_BOX) ^ RCON2)
    w[5] = w[4] ^ w[3]

    return (
        # 第一次轮密钥
        int_to_state((w[0] << 8) + w[1]),
        # 第二次轮密钥
        int_to_state((w[2] << 8) + w[3]),
        # 第三次轮密钥
        int_to_state((w[4] << 8) + w[5])
    )
```

4.2.2 半字节代替函数

1、原理：



2、代码演示：

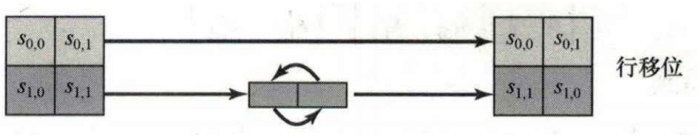
该函数接受 sbox（定义好的 S-BOX 或 S-BOX 的逆）和状态矩阵输入，返回每半字节用对应 sbox 映射得到的结果。

具体代码逻辑如下所示：

```
# 半字节代替：使用S盒替换状态矩阵中的每个字节
# 逆函数调用只需将S_BOX换位逆S_BOX
4 个用法
def half_substitution(sbox, state):
    return [sbox[nibble] for nibble in state]
```

4.2.3 行移位函数

1、原理：



2、代码演示：

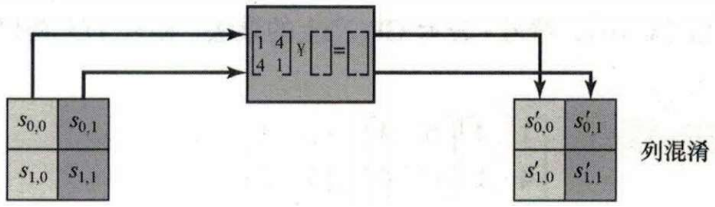
对状态矩阵进行简单变换，其逆函数与之相同

具体代码逻辑如下所示：

```
# 行移位：对状态矩阵进行简单的行移位操作。
# 逆函数与之相同
4 个用法
def line_shift(state):
    return [state[0], state[1], state[3], state[2]]
```

4.2.4 列混淆函数

1、原理：



2、代码演示：

根据列混淆的算法逻辑，首先定义伽罗瓦有限域上的乘法运算。同时，根据伽罗瓦有限域的数学知识，列混淆的逆运算可以被定义。

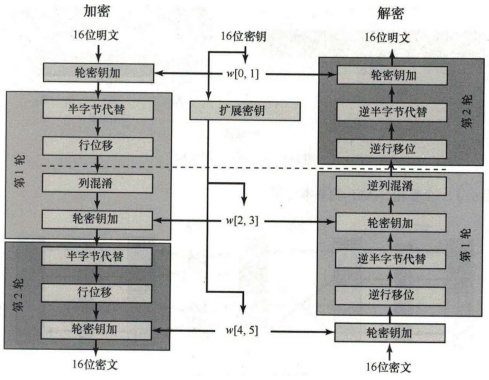
具体代码演示如下：

```
# 列混淆：定义伽罗瓦有限域乘法
12 个用法
def galois_field_multiply(a, b):
    product = 0
    a = a & 0x0F
    b = b & 0x0F

    while a and b:
        if b & 1:
            product ^= a
        a = a << 1
        if a & (1 << 4):
            a ^= 0b10011
        b = b >> 1

    return product
```

```
69      # 列混淆
        1 个用法
70      def mix_columns(state):
71          return [
72              state[0] ^ galois_field_multiply( a: 4, state[2]),
73              state[1] ^ galois_field_multiply( a: 4, state[3]),
74              state[2] ^ galois_field_multiply( a: 4, state[0]),
75              state[3] ^ galois_field_multiply( a: 4, state[1]),
76          ]
```

4.2.5 二进制的加解密算法

1、原理：



2、代码演示：

根据如上加解密流程图，依次调用密钥扩展、轮密钥加（异或运算）函数，然后进行两轮运算，调用半字节代替、行位移、列混淆、轮密钥加函数，最终得到 16 bit 的密文。

具体代码演示如下：

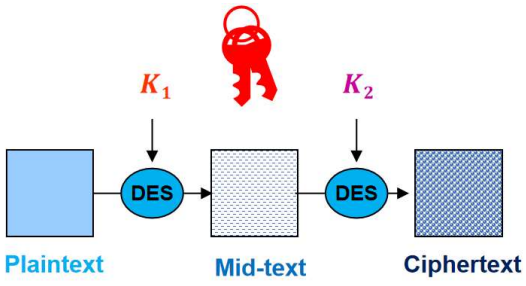
```python
# 二进制加密算法
19 个用法
def encrypt(plaintext, key):
    plaintext = int(plaintext, 2)
    if isinstance(key, str):
        key = int(key, 2)
    key0, key1, key2 = key_expansion(key)
    # 轮密钥加（key0）
    state = add_roundkey(int_to_state(plaintext), key0)
    # 第一轮半字节代替
    state = half_substitution(S_BOX, state)
    # 第一轮行位移
    state = line_shift(state)
    # 第一轮列混淆
    state = mix_columns(state)
    # 第一轮轮密钥加（key1）
    state = add_roundkey(state, key1)
    # 第二轮半字节代替
    state = half_substitution(S_BOX, state)
    # 第二轮行位移
    state = line_shift(state)
    # 第二轮轮密钥加（key2）
    state = add_roundkey(key2, state)
    # 将最终的state转换为十进制数
    encrypted_int = state_to_int(state)
    # 将十进制数转换为16位二进制字符串，左侧补零
    encrypted_binary = bin(encrypted_int)[2:].zfill(16)

    return encrypted_binary
```

4.2.6 ASCII 码的加解密算法

ASCII 码的加解密算法与二进制加解密算法基本一致，只需要加入一个将 ASCII 码转化为二进制数/将二进制数转化为 ASCII 码的过程。故此处不再赘述。

4.2.7 双重加解密算法

1、原理：进行两次 AES 加/解密运算



2、代码演示：

（1）拆分密钥：将输入的 32 bit 密钥按高低位拆分为 key1 key2，执行加解密运算。

```python
# 二重加密
# 拆分输入的32 bit密钥，按高位和低位拆分位key1 key2
4 个用法
def create_key(key):
    # print(key)
    key = int(key, 2)

    # if key < 0 or key > 0xFFFFFFFF:
    #     raise ValueError("错误，密钥输入必须为32 bit")

    key1 = (key & 0xFFFF0000) >> 16  # 密钥的高16位
    key2 = key & 0x0000FFFF  # 密钥的低16位

    return key1, key2
```

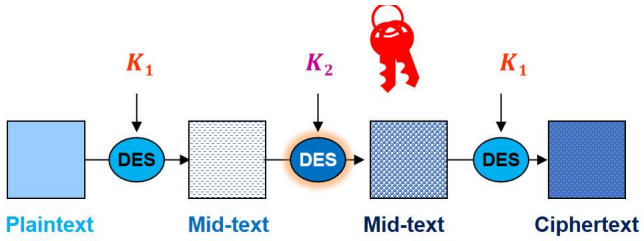
（2）用得到的 key1 key2 迭代的对明文进行两次加密/解密运算

（双重加解密共有两种实现方式，只有函数调用的区别，故在此不做冗余展示）

```python
# 第一种二重加密方式
# 加密过程：key1 key2均执行加密过程
1 个用法
def double_encrypt(plaintext, key):
    key1, key2 = create_key(key)
    mid_text = encrypt(plaintext, key1)  # key1加密
    ciphertext = encrypt(mid_text, key2)  # key2加密
    return ciphertext


# 解密过程：key1 key2均执行解密过程
1 个用法
def double_decrypt(ciphertext, key):
    key1, key2 = create_key(key)
    mid_text = decrypt(ciphertext, key2)  # key2解密
    plaintext = decrypt(mid_text, key1)  # key1解密
    return plaintext
```

4.2.8 三重加解密算法

1、原理：进行三次 AES 加/解密运算。该算法可以有效避免中间相遇攻击



2、代码演示：

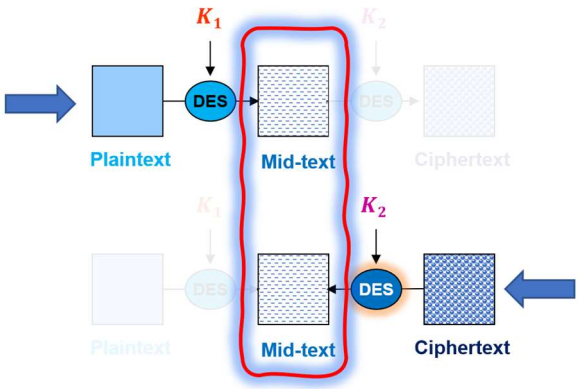
（1）拆分密钥：将输入的 48 bit 密钥按高低位拆分为 key1 key2 key3，执行加解密运算。

（2）用得到的 key1 key2 key3 迭代的对明文进行两次加密/解密运算

（双重加解密共有三种实现方式，与双重加密仅有调用函数顺序和次数的区别，故在此不做代码冗余展示）

4.2.9 中间相遇攻击算法

1.原理；分两头进行蛮力法解密，分别获得 mid-text,直到找到一对相同的 mid-text



2.代码演示

（1）密钥生成：这部分代码负责生成所有可能的前半部分和后半部分密钥。它使用了一个位移操作来确定密钥的数量，然后通过循环遍历所有可能的密钥值，将每个密钥值转换为16位的二进制字符串。

（2）前向表构建：在这个模块中，代码通过遍历所有可能的前半部分密钥，使用一个加密函数（encrypt）对明文进行加密，并将结果存储在一个字典（forward_table）中。这个字典将加密后的中间值作为键，对应的前半部分密钥作为值。

（3）后向表构建与匹配查找：这个模块首先构建后向表，通过遍历所有可能的后半部分密钥，使用一个解密函数（decrypt）对密文进行解密，并将结果存储在另一个字典（backward_table）中。然后，代码遍历前向表中的每个条目，并在后向表中查找匹配的中间值。

```
# 中间相遇攻击
2 个用法
def meet_in_the_middle_attack(plaintext, ciphertext):
    # 定义可能的密钥数量，这里我们只考虑16位密钥
    num_keys = 1 << 16

    # 创建前向表，枚举所有可能的前半部分密钥
    forward_table = {}
    for key1 in range(num_keys):
        key1_binary = format(key1, '016b')
        intermediate = encrypt(plaintext, key1_binary)
        forward_table[intermediate] = key1_binary

    # 创建后向表，枚举所有可能的后半部分密钥
    backward_table = {}
    for key2 in range(num_keys):
        key2_binary = format(key2, '016b')
        intermediate = decrypt(ciphertext, key2_binary)
        backward_table[intermediate] = key2_binary

    # 查找匹配项，尝试找到正确的密钥组合
    for intermediate, key1_binary in forward_table.items():
        if intermediate in backward_table:
            key2_binary = backward_table[intermediate]
            return key1_binary + key2_binary  # 返回找到的密钥组合

    return None  # 如果没有找到正确的密钥组合，则返回None
```

4.2.10 文件加密算法

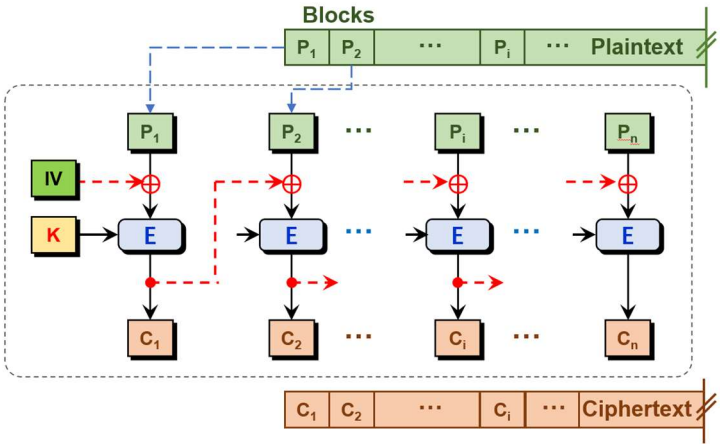
1.原理：先读取前端传入的文件，然后检测文件内容为二进制或是文本，根据类型在后端加密，完成后保存到传入的保存路径中。

2.代码演示

```
def is_binary_data(byte_data):
    binary_count = 0
    total_count = len(byte_data)

    for byte in byte_data:
        # 检查是否为控制字符或者空字节
        if byte == 0x00 or (0x00 < byte < 0x20) or (byte == 0x7F):
            binary_count += 1

    # 根据控制字符的比例判断是否为二进制数据
    if total_count > 0 and (binary_count / total_count) > 0.1:
        return True
    return False

def process_file_upload(file, key, save_path):
    try:
        # 读取文件内容（以二进制模式）
        file_content = file.read()
        print(file_content)

        # 检查文件内容是否仅包含 '0' 和 '1'
        if all(char in b'01' for char in file_content):
            print("文件内容仅包含 '0' 或 '1'，执行 encrypt 函数")
            encrypted_content = encrypt(file_content, key)
        else:
            if is_binary_data(file_content):
                print("true binary")
                encrypted_content = encrypt(file_content, key)
            else:
                print("false binary")
                try:
                    # 尝试将内容解码为ASCII
                    ascii_content = file_content.decode('ascii')
                    print(ascii_content)
                    encrypted_content = encrypt_string(ascii_content, key)
                except UnicodeDecodeError:
                    print(f"可能是二进制文件，解码为ASCII失败")

        # 确保加密后的内容是字节类型，以便可以写入文件
        if isinstance(encrypted_content, str):
            encrypted_content = encrypted_content.encode('utf-8')

        # 保存加密后的文件（以二进制模式写入）
        with open(save_path, 'wb') as encrypted_file:
            encrypted_file.write(encrypted_content)

        return {'status': 'success', 'message': '文件加密保存成功'}
    except Exception as e:
        return {'status': 'error', 'message': str(e)}
```

4.2.11 CBC 模式加密算法

1.原理：先将明文切分成若干小段，然后每一小段与初始块或者上一段的密文段进行异或运算后，再与密钥进行加密。



2.代码演示

```
2 个用法
def cbc_encrypt(plaintext, key, iv):
    binary_plaintext = ''.join([bin(ord(char)).replace( _old: "0b",  _new: "").zfill(8) for char in plaintext])
    encrypted_blocks = []
    previous_cipherblock = iv

    for i in range(0, len(binary_plaintext), 16):
        block = binary_plaintext[i:i + 16]
        xor_block = ''.join([str(int(a) ^ int(b)) for a, b in zip(block.zfill(16), previous_cipherblock.zfill(16))])
        cipherblock = encrypt(xor_block, key)
        encrypted_blocks.append(cipherblock)
        previous_cipherblock = cipherblock

    return ''.join(encrypted_blocks)

2 个用法
def cbc_decrypt(ciphertext, key, iv):
    decrypted_blocks = []
    previous_cipherblock = iv

    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i + 16]
        decrypted_block = decrypt(block, key)
        xor_block = ''.join([str(int(a) ^ int(b)) for a, b in zip(decrypted_block.zfill(16), previous_cipherblock.zfill(16))])
        decrypted_blocks.append(xor_block)
        previous_cipherblock = block

    binary_plain_text = ''.join(decrypted_blocks)
    ascii_plain_text = ''.join([chr(int(binary_plain_text[i:i + 8], 2)) for i in range(0, len(binary_plain_text), 8)])
    return ascii_plain_text
```

（1）加密算法

[1]. 明文转换：先将明文字符串中的每个字符转换为其 ASCII 值的二进制表示，并将每个二进制字符串填充到 8 位。

[2]. 加密过程：循环遍历明文的二进制表示，每次处理 16 位（一个块）。将当前块与前一个加密块进行异或操作。使用加密函数和密钥对异或后的结果进行加密。将加密后的块添加到列表中。更新 previous_cipherblock 为当前的加密块。

（2）解密算法

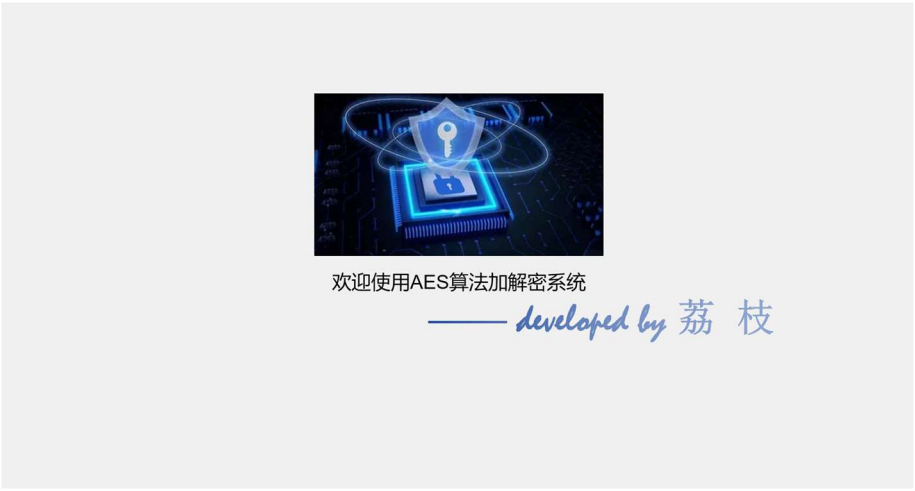
[1]. 解密过程：循环遍历密文，每次处理 16 位（一个块）。使用解密函数和密钥对当前块进行解密。将解密后的块与前一个加密块进行异或操作。将异或后的块添加到列表中。更新 previous_cipherblock 为当前的块。

[2]. 转换为明文：将所有解密后的块连接成一个二进制字符串。将二进制字符串转换回 ASCII 字符。

5. 基础功能及界面

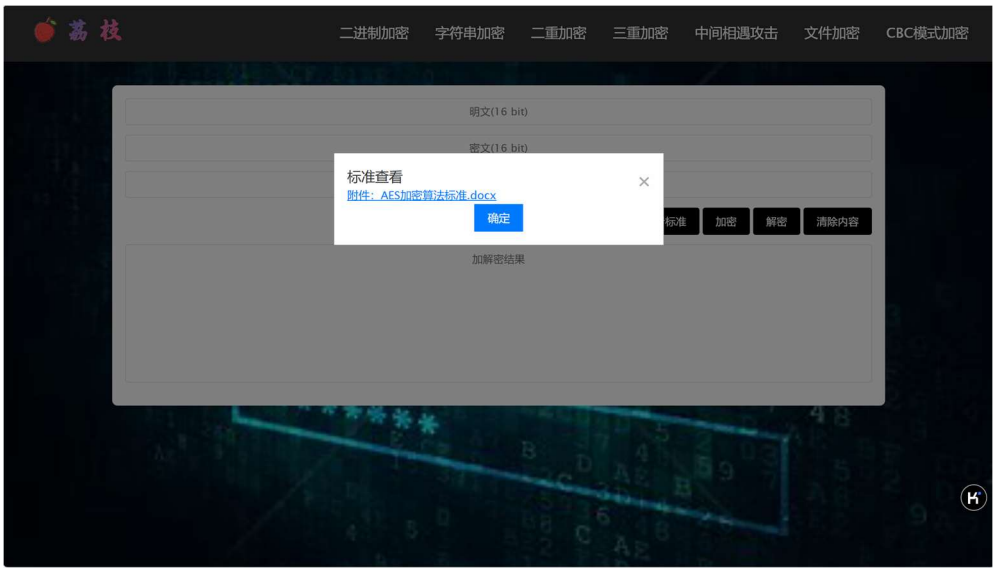
5.1 启动动画

该动画页面主要显示欢迎信息和关于开发者团队的信息，可以帮助大家了解我们开发者团队。



5.2 原理讲解页面

在该页面内可以看到 AES 加解密算法的基础原理。



5.3 二进制加密页面

该页面提供了二进制加解密功能。

1、用户可以在明文文本框和密钥文本框分别输入 16-bits 和 16-bits 的明文和密钥，点击“加密”按钮，执行加密操作。

2、用户可以在密文文本框和密钥文本框输入 16-bits 和 16-bits 的密文和密钥，点击“解密”按钮，执行解密操作。

3、用户可以点击“查看标准”按钮进入模态窗口，查看该 AES 算法中的标准数据信息，同时下载相关文档以获取数据。

4、用户可以点击“清除内容”按钮，清除本页面文本框中的内容。

5.4 ASCII 码加密页面

基础功能与二进制加密页面类似。

1、用户在文本框输入明文为 2 bytes 字符串，密钥为 16 bit 二进制数的内容，点击“加密”按钮，对字符串进行加密操作。

2、用户在文本框输入密文为 16 bit 二进制数，密钥为 16 bit 二进制数的内容，点击“解密”按钮，解密为 ASCII 码内容。

3、加解密结果文本框会同时输出二进制表示和 ASCII 码表示。

5.5 双重加密页面

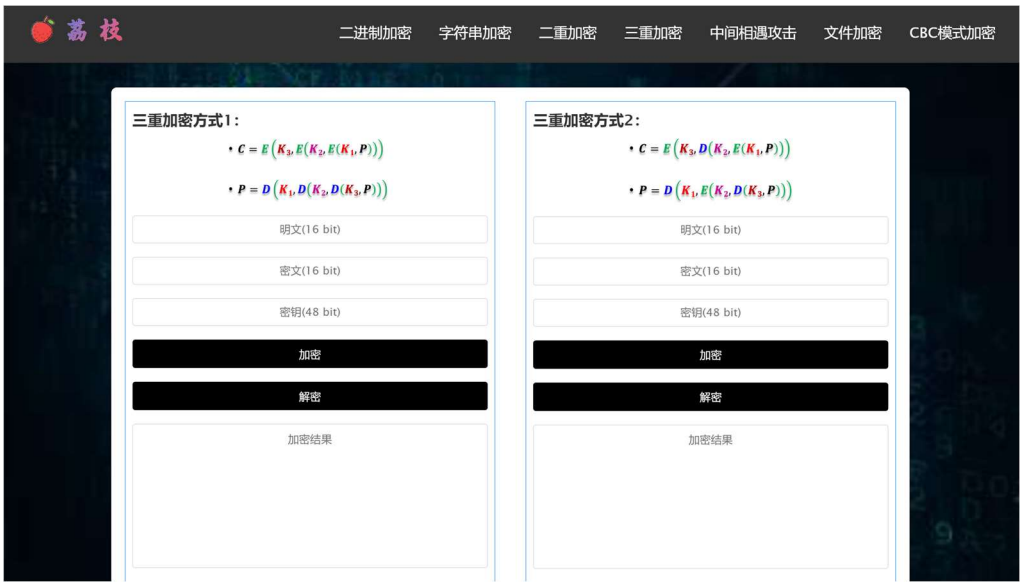
1、该页面接收输入 16 bit 的明文和 32 bit 的密钥，点击“加密”按钮，可以对明文进行双重加密算法的加密运算，点击“解密”按钮，可以对密文实现普通解密算法（不同于中间相遇攻击）的解密运算。

2、用户可以自行选择使用两种双重加密算法。但值得注意的是，右侧双重加密算法要求 Key1≠key2，即输入密钥的前 16bit 与后 16 bit 应该不完全相同，否则可能会导致加密失败。

5.6 三重加密页面

1、用户可以点击下拉框中的选项，选择 32 bit 密钥的三重加密算法，或 48 bit 密钥的加密算法

2、页面接收输入 16 bit 的明文和 32 bit（或 48 bit）的密钥，点击“加密”按钮，可以对明文进行三重加密算法的加密运算，点击“解密”按钮，可以对密文实现解密运算。（注意，三重加密不存在中间相遇攻击，只能使用普通的、高时间代价的方式进行解密）



5.7 中间相遇攻击界面

1、用户在文本框输入明文为 16 bit 二进制数，密钥为 16 bit 二进制数的内容，点击“执行中间相遇攻击”按钮，对明密文对执行中间相遇攻击操作。

2、中间相遇攻击解密结果文本框会输出得到的密钥对。

5.8 文件加密页面

1、用户在文件选择框选择要加密的 txt 文件，输入密钥为 16 bit 二进制数，在保存地址输入框输入保存加密后文件的地址，点击“加密文件”按钮，对文件和密钥执行加密操作，并将加密后的文件保存在制定的保存地址。

2、结果文本框会输出加密是否成功的结果以及失败的原因。

5.9CBC 模式加密页面

1、用户在文本框输入明文字符串，密钥为 16 bit 二进制数的内容，初始向量 iv 为 16 bit 二进制数的内容，点击“加密”按钮，对字符串进行加密操作。

2、用户在文本框输入密文为二进制数，密钥为 16 bit 二进制数的内容，点击“解密”按钮，解密为明文文本内容。

3、加解密结果文本框会输出相应的明密文内容。

6. 功能扩展

6.1 新功能——文件加密

6.1.1 文件加密

文件加密：读取一个文件的内容，识别文件内容为二进制或文本，使用密钥对其进行相应加密，然后将加密后的内容保存到指定的路径。

6.1.2 文件加密的实现

```
def is_binary_data(byte_data):
    binary_count = 0
    total_count = len(byte_data)

    for byte in byte_data:
        # 检查是否为控制字符或者空字节
        if byte == 0x00 or (0x00 < byte < 0x20) or (byte == 0x7F):
            binary_count += 1

    # 根据控制字符的比例判断是否为二进制数据
    if total_count > 0 and (binary_count / total_count) > 0.1:
        return True
    return False

def process_file_upload(file, key, save_path):
    try:
        # 读取文件内容（以二进制模式）
        file_content = file.read()
        print(file_content)

        # 检查文件内容是否仅包含 '0' 和 '1'
        if all(char in b'01' for char in file_content):
            print("文件内容仅包含 '0' 或 '1'，执行 encrypt 函数")
            encrypted_content = encrypt(file_content, key)
        else:
            if is_binary_data(file_content):
                print("true binary")
                encrypted_content = encrypt(file_content, key)
            else:
                print("false binary")
                try:
                    # 尝试将内容解码为ASCII
                    ascii_content = file_content.decode('ascii')
                    print(ascii_content)
                    encrypted_content = encrypt_string(ascii_content, key)
                except UnicodeDecodeError:
                    print(f"可能是二进制文件，解码为ASCII失败")

        # 确保加密后的内容是字节类型，以便可以写入文件
        if isinstance(encrypted_content, str):
            encrypted_content = encrypted_content.encode('utf-8')

        # 保存加密后的文件（以二进制模式写入）
        with open(save_path, 'wb') as encrypted_file:
            encrypted_file.write(encrypted_content)

        return {'status': 'success', 'message': '文件加密保存成功'}
    except Exception as e:
        return {'status': 'error', 'message': str(e)}
```

6.2 功能展示

传入文件、密钥和保存路径，点击加密文件，加密成功后将返回成功信息。

以下展示原文件和加密后文件内容：