|  |  |
| --- | --- |
| 001 | jsjlogo |

**合肥工业大学**

**计算机与信息学院**

**密码学实验**

|  |  |
| --- | --- |
| **学生姓名：** | 陈若禹 |
| **学 号：**  **邮 箱：** | **2023217422**  1928407107@qq.com |
| **专业班级：** | 信息安全23-1班 |
| **指导教师：** | 童秋云 |

**实验一**

1. **实验题目**

**DES 算法的实现与应用**

1. **实验目的**

1. 理解对称加密算法中 DES的基本原理及加解密流程。

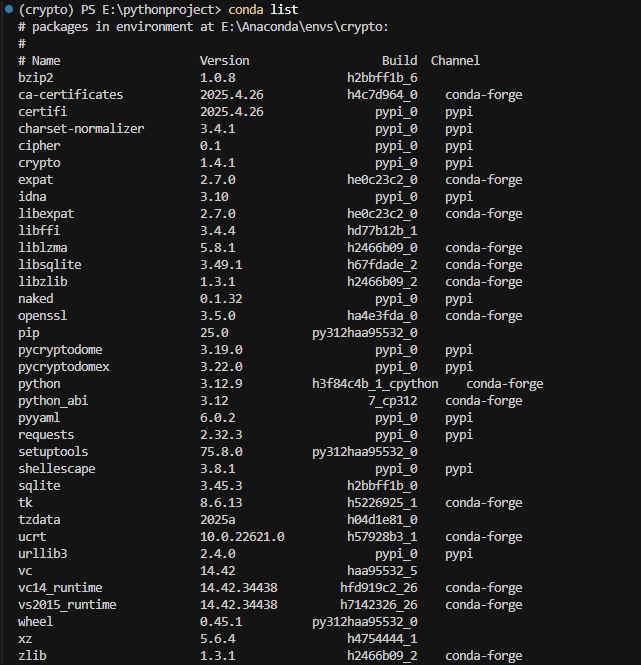
2. 掌握 DES 算法的编码实现方法，理解置换、轮函数和子密钥生成机制。

3. 熟悉使用 DES 加密库进行实际加密解密操作。

4. 学会将密码算法应用于实际数据的加解密处理，增强实际动手能力。

1. **实验环境**

本实验通过python语言完成；建立的环境python版本为3.12.5；其中环境具体库的版本如图所示：



1. **算法原理**

概述：

DES 是一种块加密算法，明文被分成长度固定为 64 位的块进行加密。DES 使用一个 56 位的密钥进行加密，但由于校验位原因，实际输入密钥长度为 64 位，其中每隔 8 位为奇偶校验位，仅 56 位参与加密运算。

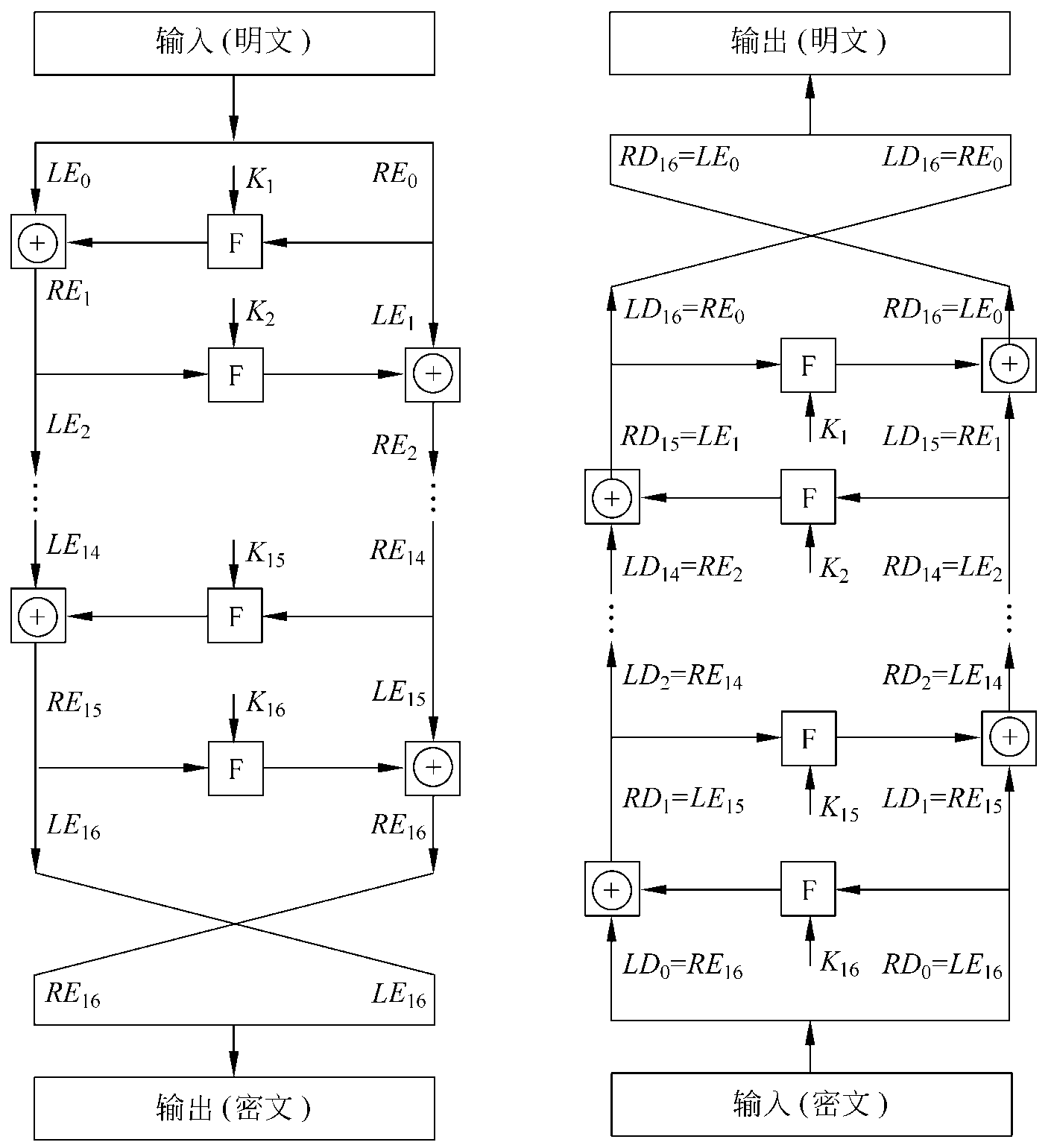


图1 Feistel解密结构

DES 加密由 16 轮 Feistel 结构组成，其核心是 密钥生成 和 轮函数 F。

（1）初始置换（Initial Permutation, IP）：对 64 位明文按固定规则进行重新排列。

（2）16 轮加密：

每轮将 64 位数据分为左右两部分（各 32 位）；使用密钥子集对右半部分进行复杂变换（通过轮函数 F）；将变换结果与左半部分进行异或（XOR）操作；左右两部分交换位置。

最终置换（Inverse Initial Permutation, IP⁻¹）：加密完成后，数据按初始置换的逆过程重新排列。

轮函数 F 包括以下步骤： - 扩展置换（Expansion Permutation）：将右半部分从 32 位扩展为 48 位。 - 密钥混合（Key Mixing）：将扩展后的数据与 48 位子密钥进行 XOR 操作。 - S-盒置换（Substitution through S-boxes）：使用 8 个 S 盒将 48 位数据替换为 32 位数据。 - P-置换（Permutation）：重新排列 S-盒的输出。

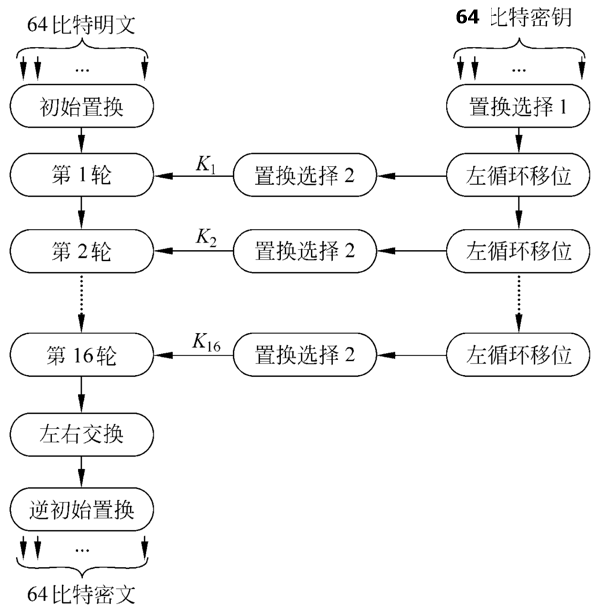


图2 DES算法框图

此外,DES还引入了扩散和混淆：

扩散(diffusion)：将明文及密钥的影响尽可能迅速地散布到较多输出的密文中，使比特0、1均匀分布；

混淆(confusion)：使明文、密钥和密文之间的关系复杂化。

尽管 DES 由于密钥长度限制已逐渐被 AES 等更安全的算法取代，但其经典的 Feistel 结构和设计思想为现代加密算法提供了宝贵的经验。

1. **算法设计**

本脚本最核心的特点是提供了两种截然不同的DES实现方式：第一种是“自主实现版”(manual)，它从零开始，完整地用Python代码构建了DES的全部核心逻辑，包括Feistel网络、S盒代换、P盒置换以及复杂的16轮密钥调度算法，旨在深入揭示算法的内部工作原理；第二种是“库调用版”(lib)，它通过调用专业、高效的第三方加密库pycryptodome来执行加密，代表了在实际项目开发中的标准做法。

同时为了保证安全性，两种实现都采用了比基础ECB模式更安全的CBC（密码块链接）模式进行加密，并正确处理了数据的PKCS#5填充。

在用户交互方面，脚本提供了极大的灵活性。用户既可以通过命令行参数进行精确控制，也可以在不带任何参数的情况下进入一个用户友好的交互式问答模式。该模式会引导用户输入密钥、初始化向量（IV）和数据，并内置了严格的8字节长度验证，确保输入的有效性。此外，脚本还包含一个专门的test()函数，它不仅能验证两种实现的加解密自洽性，还能通过断言确保我们手动实现的结果与标准库完全一致，从而为代码的正确性提供了强有力的证明。

#### 方法1：自主实现基础版DES

**1.Feistel网络与块处理**

我通过Feistel来处理一个64位的独立数据块，分为五个部分：

(1)初始置换 (IP)：输入的数据块首先经过IP表进行一次固定的位-对-位重排，打乱原始数据的顺序。

(2)数据分半：置换后的64位数据被平均分为左右两半，L0和R0，各32位。

(3)16轮迭代：循环执行16次加密轮。在每一轮中：

将当前右半部分right直接作为下一轮的左半部分new\_left。

将当前右半部分right送入复杂的f\_function函数，并与本轮的子密钥k进行运算。

f函数的结果与当前左半部分left进行异或(XOR)操作，其结果作为下一轮的右半部分new\_right。

(4)最终交换与逆置换：16轮结束后，左右两半并不交换（这是Feistel结构的一个特性，实际上在最后一轮循环内部已经完成了交换），直接合并，然后通过IP\_INV逆初始置换表，恢复数据位的顺序，最终得到64位的密文块。

(5)解密逻辑：该函数通过一个decrypt标志来区分加解密。如果是解密，它唯一要做的就是将16轮的子密钥round\_keys逆序使用。Feistel网络结构的这种对称性设计使得加解密过程可以共用同一套硬件/软件逻辑，极为高效。

def manual\_des\_block(*block\_bytes*: bytes, *round\_keys*: list, *decrypt*=False) -> bytes:

    block\_int = int.from\_bytes(block\_bytes, 'big')

    block\_bits = int\_to\_bits(block\_int, 64)

    permuted = permute(block\_bits, IP)

    left, right = permuted[:32], permuted[32:]

    keys = round\_keys[::-1] *if* decrypt *else* round\_keys

*for* k *in* keys:

        new\_left = right

        fx = f\_function(right, k)

        new\_right = [l ^ f *for* l, f *in* zip(left, fx)]

        left, right = new\_left, new\_right

    combined = right + left

    final\_bits = permute(combined, IP\_INV)

    final\_int = bits\_to\_int(final\_bits)

*return* final\_int.to\_bytes(8, 'big')

1. **轮函数核心：F函数**

在原有的代码进行验收的时候，我的F函数仅仅只是通过了简单的异或函数来实现，无法保证DES算法的安全性，这一点在实验验收的时候助教也是指了出来。故我重新修改了F函数，其设计分为四个步骤：

（1）扩展置换 (Expansion)：输入的32位右半部分r\_half通过E表扩展为48位。这一步既增加了数据长度以匹配48位的子密钥，也让部分位（16个）在后续步骤中影响两个S盒，从而加速了雪崩效应（扩散）。

（2）密钥混合：扩展后的48位数据与本轮的48位子密钥subkey进行异或操作。这是密钥真正参与到数据加密的环节。

（3）S盒代换 (Substitution)：这是DES中唯一的非线性操作，也是其安全性的关键。混合后的48位数据被分为8个6位的块，每个块分别送入一个对应的S盒（S\_BOXES）。每个S盒根据6位输入（第1和第6位决定行，中间4位决定列）输出一个4位的值。这个代换过程是不可逆的，有效地抵抗了差分密码分析。

（4）P盒置换 (Permutation)：从S盒输出的8个4位结果（共32位）经过P盒进行一次置换。这一步纯粹是扩散操作，它将单个S盒的输出尽可能均匀地散布到下一轮所有S盒的输入中，进一步增强了雪崩效应。

def f\_function(*r\_half*: list, *subkey*: list) -> list:

*# 1. 扩展置换: 32位 -> 48位*

    expanded\_r = permute(r\_half, E)

*# 2. 与子密钥异或*

    xored = [r ^ k *for* r, k *in* zip(expanded\_r, subkey)]

*# 3. S-盒代换: 48位 -> 32位*

    s\_box\_out = []

*for* i *in* range(8):

        chunk = xored[i\*6 : i\*6+6]

*# 行号由第1位和第6位决定*

        row = (chunk[0] << 1) | chunk[5]

*# 列号由中间4位决定*

        col = bits\_to\_int(chunk[1:5])

*# 从S-盒中查找值*

        val = S\_BOXES[i][row][col]

*# 将4位整数转换为4位比特列表*

        s\_box\_out.extend(int\_to\_bits(val, 4))

*# 4. P-盒置换: 32位*

*return* permute(s\_box\_out, P)

**3.密钥调度算法**

这个函数负责从用户提供的单个64位主密钥中，生成16个不同的48位子密钥，供每一轮加密使用。分为以下三个步骤：

（1）置换选择1 (PC-1)：64位主密钥首先通过PC1表，丢弃8个奇偶校验位，并将其余56个有效位进行置换。

（2）分半与移位：置换后的56位分为左右两半（C和D，各28位）。在16轮的生成过程中，根据LEFT\_SHIFTS表，C和D两部分在每一轮分别进行1位或2位的循环左移。这个设计确保了每一轮的子密钥都与上一轮不同，且主密钥的每一位都会在多轮的子密钥中出现。

（3）置换选择2 (PC-2)：每一轮移位后，合并的56位C和D会通过PC2表。PC2从56位中选择并置换出48位，作为本轮最终的子密钥。

def generate\_round\_keys(*key\_bytes*: bytes) -> list:

*# 从8字节密钥生成16轮的48位子密钥*

    key\_int = int.from\_bytes(key\_bytes, 'big')

    key\_bits = int\_to\_bits(key\_int, 64)

*# PC-1置换: 64位 -> 56位*

    permuted\_key\_56 = permute(key\_bits, PC1)

*# 分为左右两半 C0, D0*

    c, d = permuted\_key\_56[:28], permuted\_key\_56[28:]

    round\_keys = []

*for* i *in* range(16):

*# 循环左移*

        shift = LEFT\_SHIFTS[i]

        c = c[shift:] + c[:shift]

        d = d[shift:] + d[:shift]

*# 合并并通过PC-2置换: 56位 -> 48位*

        combined\_cd = c + d

        subkey\_48 = permute(combined\_cd, PC2)

        round\_keys.append(subkey\_48)

*return* round\_keys

**4. CBC链接模式与填充**

DES是一个块加密算法，只能处理固定长度的块。CBC函数将其扩展为一个可以处理任意长度数据的实用加密方案。

（1）填充 (Padding)：在加密前，首先调用pad\_pkcs5对明文进行填充，确保其总长度是8字节（64位）的整数倍。

（2）加密链接 (CBC Mode)：

第一个明文块与一个公开的初始化向量(IV)进行异或，从第二个明文块开始，每一个明文块在加密前，都会与前一个块的密文进行异或。prev\_cipher\_block变量用于保存前一个密文块，实现了这种“链接”关系。这个设计确保了即使明文块相同，只要它们在消息中的位置不同，产生的密文块也不同，从而隐藏了明文的统计规律。

（3）解密链接：

解密时，每个密文块首先通过manual\_des\_block进行解密。解密后的结果再与前一个块的密文进行异或，以还原出原始的明文块。所有块解密完成后，调用unpad\_pkcs5移除填充字节，得到原始明文。

def manual\_des\_encrypt(*plaintext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

*assert* len(key) == 8, "密钥必须是 8 字节。"

*assert* len(iv) == 8, "IV必须是 8 字节。"

    round\_keys = generate\_round\_keys(key)

    padded\_plaintext = pad\_pkcs5(plaintext)

    blocks = [padded\_plaintext[i:i+8] *for* i *in* range(0, len(padded\_plaintext), 8)]

    ciphertext = b''

    prev\_cipher\_block = iv

*for* block *in* blocks:

*# CBC mode: P\_i XOR C\_{i-1}*

        block\_to\_encrypt = xor\_bytes(block, prev\_cipher\_block)

        encrypted\_block = manual\_des\_block(block\_to\_encrypt, round\_keys, *decrypt*=False)

        ciphertext += encrypted\_block

        prev\_cipher\_block = encrypted\_block

*return* ciphertext

def manual\_des\_decrypt(*ciphertext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

*assert* len(key) == 8, "密钥必须是 8 字节。"

*assert* len(iv) == 8, "IV必须是 8 字节。"

*assert* len(ciphertext) % 8 == 0, "密文长度必须是 8 字节的倍数。"

    round\_keys = generate\_round\_keys(key)

    blocks = [ciphertext[i:i+8] *for* i *in* range(0, len(ciphertext), 8)]

    decrypted\_padded = b''

    prev\_cipher\_block = iv

*for* block *in* blocks:

        decrypted\_block = manual\_des\_block(block, round\_keys, *decrypt*=True)

*# CBC mode: D(C\_i) XOR C\_{i-1}*

        plaintext\_block = xor\_bytes(decrypted\_block, prev\_cipher\_block)

        decrypted\_padded += plaintext\_block

        prev\_cipher\_block = block

*return* unpad\_pkcs5(decrypted\_padded)

#### 方法2：调用库版DES

pycryptodome库的特点是高度抽象与封装。所有手动版本中涉及的复杂实现（Feistel网络、S盒、密钥调度等）都被封装在一个黑盒中。开发者通过调用DES.new()来创建一个“密码器”对象（cipher）。这个调用是声明式的，你只需要告诉库：你想用的密钥 (key)。

你想用的操作模式 (DES.MODE\_CBC)。该模式所需的初始化向量 (iv)。

库在内部会自动完成所有的初始化工作，包括生成16轮的子密钥并将其与密码器对象绑定，准备好执行CBC模式的加密逻辑。这种设计将开发者与复杂的底层实现隔离开，降低了使用门槛，并减少了因实现错误导致安全漏洞的风险。

方法2的算法逻辑和方法1相似，但是方法2的代码是可以通过库调用实现故较为便捷:

def library\_des\_encrypt(*plaintext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

    cipher = DES.new(key, DES.MODE\_CBC, iv)

    padded = pad\_pkcs5(plaintext)

*return* cipher.encrypt(padded)

def library\_des\_decrypt(*ciphertext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

    cipher = DES.new(key, DES.MODE\_CBC, iv)

    decrypted = cipher.decrypt(ciphertext)

*return* unpad\_pkcs5(decrypted)

def pad\_pkcs5(*data*: bytes) -> bytes:

    pad\_len = 8 - (len(data) % 8)

*return* data + bytes([pad\_len] \* pad\_len)

def unpad\_pkcs5(*data*: bytes) -> bytes:

    pad\_len = data[-1]

*return* data[:-pad\_len]

def library\_des\_encrypt(*plaintext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

    cipher = DES.new(key, DES.MODE\_CBC, iv)

    padded = pad\_pkcs5(plaintext)

*return* cipher.encrypt(padded)

def library\_des\_decrypt(*ciphertext*: bytes, *key*: bytes, *iv*: bytes) -> bytes:

    cipher = DES.new(key, DES.MODE\_CBC, iv)

    decrypted = cipher.decrypt(ciphertext)

*return* unpad\_pkcs5(decrypted)

1. **原文内容**

我在这里的默认密钥为：HFGYDX25；默认的初始向量IV为：12345678。后面会换用不同的密钥和IV以保证算法的鲁棒性。

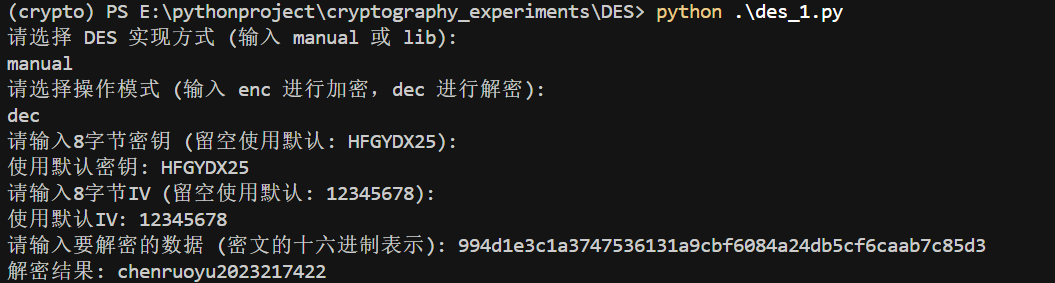
下面我进行加密的明文统一采用我的姓名加学号：chenruoyu2023217422。

1. **结果验证**

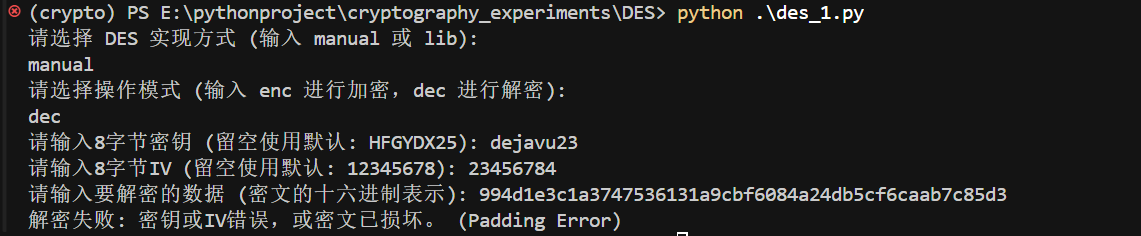
#### 方法1：自主实现基础版DES：



将chenruoyu2023217422采用默认的密钥和IV加密后得到密文：994d1e3c1a3747536131a9cbf6084a24db5cf6caab7c85d3。

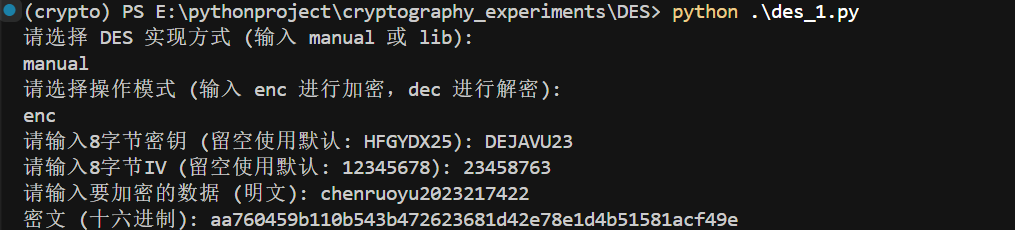


将密文994d1e3c1a3747536131a9cbf6084a24db5cf6caab7c85d3采用默认的密钥和IV解密后得到明文，和明文一致，证明加密解密的正确性。

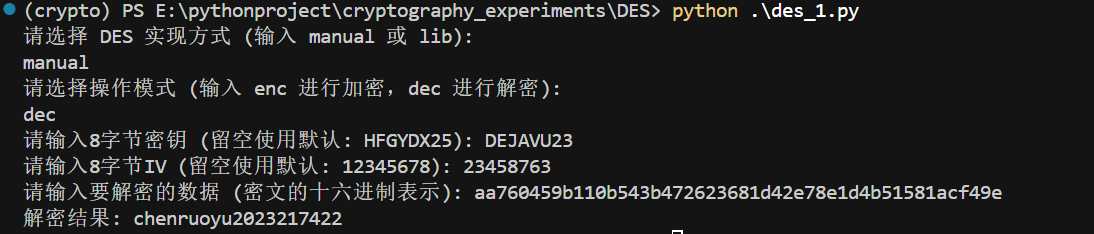


当我采用非默认密钥和IV（在这里密钥选取dejavu23，IV选取23456784）进行解密同一个密文的时候，unpad\_pkcs5 函数会严格检查填充的格式是否符合PKCS#5标准。

如果解密后的数据是乱码，导致填充格式不正确，它会立刻抛出一个带有"Invalid padding"信息的ValueError。



另外我重新进行了密钥和IV的选取加密同一个明文：（密钥选取DEJAVU23,IV选取23458763），得到密文：aa760459b110b543b472623681d42e78e1d4b51581acf49e。



用上述密文进行解密结果显示正确。

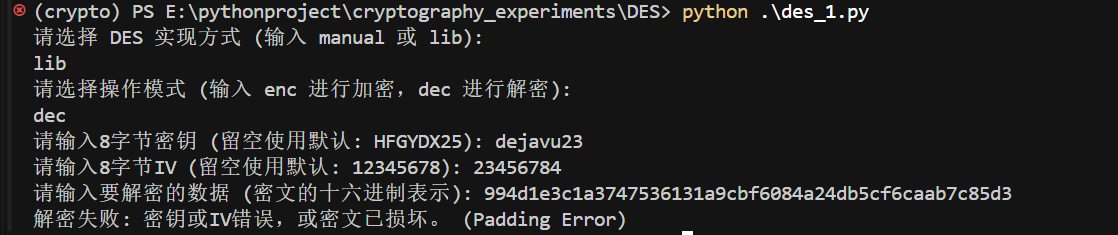
#### 方法2：调用库版DES：



将chenruoyu2023217422采用默认的密钥和IV加密后得到密文：994d1e3c1a3747536131a9cbf6084a24db5cf6caab7c85d3。在这里采用的S盒P盒和方式1一样，故得到密文也与方式1相同。

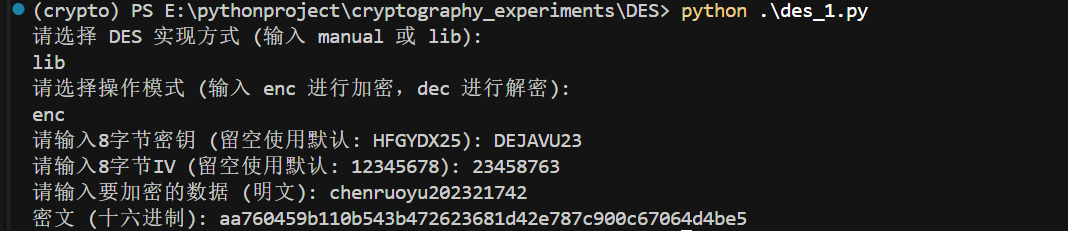


将密文994d1e3c1a3747536131a9cbf6084a24db5cf6caab7c85d3采用默认的密钥和IV解密后得到明文，和明文一致，证明加密解密的正确性。

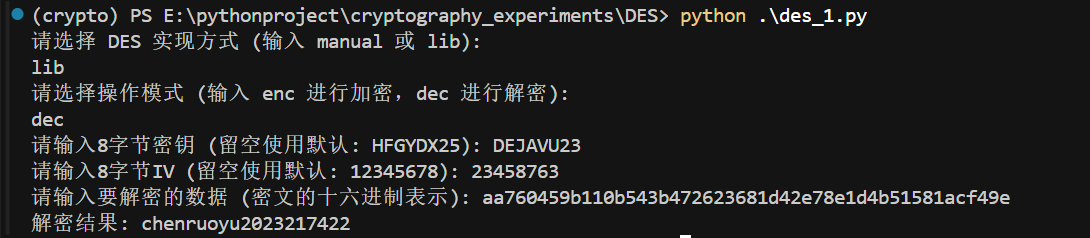


当我采用非默认密钥和IV（在这里密钥选取dejavu23，IV选取23456784）进行解密同一个密文的时候，unpad\_pkcs5 函数会严格检查填充的格式是否符合PKCS#5标准。

如果解密后的数据是乱码，导致填充格式不正确，它会立刻抛出一个带有"Invalid padding"信息的ValueError。



另外我重新进行了密钥和IV的选取加密同一个明文：（密钥选取DEJAVU23,IV选取23458763），得到密文：aa760459b110b543b472623681d42e78e1d4b51581acf49e。



用上述密文进行解密结果显示正确。

1. **实验收获**

手动实现DES算法，首先揭示了其内在的复杂性远超高层理论模型。算法的安全性不仅源于其16轮的Feistel网络结构，更深植于一系列固定置换和代换表的精确数值。这些表，包括初始/逆初始置换（IP/IP-1）、扩展置换（E）、8个S盒以及P盒，包含了大量非直观的、预设的常量。在编码过程中，这些表的转录和使用必须达到100%的准确率。任何单一数值的偏差，都会通过雪崩效应，在加密过程中被迅速放大，导致最终输出的密文与标准结果完全不符。这凸显了从理解算法原理到产出正确工程实现之间存在的显著难度，也体现了密码算法对确定性和精确性的极端依赖。

通过从ECB模式切换到CBC模式，我验证了操作模式（Mode of Operation）的关键作用。ECB模式因其对相同明文块产生相同密文块的确定性，会泄露明文的统计学规律，使其在大多数应用场景下都是不安全的。CBC模式通过引入一个初始化向量（IV）和块间链接机制，消除了这种确定性。此外，如PKCS#5等标准填充方案的必要性也得以体现，它确保了任意长度的明文都能被安全处理。结论是，一个安全的系统必须是算法原语、操作模式、IV管理和数据填充等多个环节均遵循安全实践的有机整体。

DES的Feistel结构在工程实现层面展现了卓越的设计效率。其核心的F函数不需要具备可逆性，这大大降低了F函数的设计难度。更重要的是，这种结构赋予了加解密过程高度的对称性。在我的实现中，解密逻辑与加密逻辑几乎完全复用，唯一的区别在于16轮子密钥的使用顺序是颠倒的。这种设计意味着在硬件上可以用同一套逻辑电路完成加解密，在软件上则可以大量复用代码，显著降低了开发和实现的成本与复杂度。这是算法理论在设计之初就充分考量工程化和经济性需求的经典范例。

对于CBC这样的链接加密模式，初始化向量（IV）的管理是保障安全性的基石。实验过程从使用固定IV演进到为每次加密生成随机IV，清晰地展示了IV的核心作用：打破加密过程的确定性。一个固定的、可预测的IV会使加密失去随机性，如果用同一密钥加密相同的明文，总会得到相同的密文，这违背了现代密码学所追求的语义安全目标。正确的实践是，必须为每一次独立的加密会话（session）生成一个密码学安全的、不可预测的随机IV。此IV无需保密，通常作为协议的一部分，附加在密文前一同传输，供解密方使用。严格遵循这一原则，是防止重放攻击、模式分析等多种威胁的基本前提。

1. **参考文献**

[1] 杨波, 现代密码学(第4版), 清华大学出版社, 2017  
[2] 陈恭亮. 简明信息安全数学基础. 高等教育出版社，2011  
[3] Douglas R.Stinson. 密码学原理与实践. 电子工业出版社, 2009