|  |
| --- |
| **Github账号：yetanshenshui** |
| **个人博客关于密码学实验的链接：**https://github.com/yetanshenshui/XDU\_Cryptography |
| **实验题目：**  **1.AES 密钥 - 编码于证件机读区中**  **用损坏的护照信息修复出完整密钥，解密获得隐藏的密码词。**  **2.**一组密码学挑战****  **1）实现PKCS#7填充**  **2）实现CBC模式**  **3）ECB/CBC检测预言机**  **4）一次字节ECB解密（简单）**  **5）ECB剪切粘贴攻击**  **6）一次字节ECB解密（困难）**  **7）PKCS#7填充验证**  **8）CBC位翻转攻击** |
| **实验摘要：**  **本次密码学实验围绕分组密码的核心概念展开编程实践，重点实现了PKCS#7填充算法、ECB和CBC两种加密模式。在技术实现中，我们运用字节级XOR运算处理块间链式操作，通过base64编解码处理数据格式，并利用统计分析方法检测加密模式特征。编程过程中的主要技术挑战包括：在逐字节ECB解密中需要精确控制输入长度以操纵块边界；在剪切粘贴攻击中要巧妙利用ECB的块独立性进行密文重组；在CBC实现中需正确处理初始化向量的链式传递。这些实验不仅验证了ECB模式因缺乏扩散性而导致的选择明文攻击可行性，还揭示了在实际编程中处理数据对齐、边界条件和错误处理的复杂性，体现了密码学算法从理论到实践实现过程中的关键细节。** |
| **题目描述**  **问题 1**  **AES 密钥 - 编码于证件机读区中**  **你收到了一条 AES 加密的消息（CBC 模式，零初始化向量，01-00 填充）。此外，你还收到了相应的密钥——可惜不完整——其形式类似于身份证件上的机读区，例如在欧洲电子护照中使用的那种。**  **12345678<<1110182<111116?<<<<<<<<<<<<4**  **目标是找到以下经过 base64 编码的消息的明文。**  **9MgYwmuPrjiecPMx6106zluy3MtlXQQ0E59T3xB6u0Gyf1gYs2i3K9Jxaa0zj4gTMazJuApwd6+jdyel5iGHvhQyDHGVlAuYTgJrbFDrfB22Fpil2N fNnWFBTXyf7SDI**  **加密时使用了基于基本访问控制协议的密钥 Kenc。(需要查阅文档得知编码方式以及密钥如何计算)**  ****问题2****  ****一组密码学挑战****   1. **实现PKCS#7填充**   **分组密码转换固定大小的明文块（通常为 8 或 16 字节） 成密文。将任何块填充到特定的块长度，方法是将 块末尾的填充字节数。使用PKCS#7将块填充。**   1. **实现CBC模式**   **通过手动实现CBC模式：使用之前编写的ECB函数，使其进行加密而不是解密，并使用XOR函数来组合它们。将文件使用"YELLOW SUBMARINE"作为密钥、IV全为ASCII 0（\x00\x00\x00等）进行CBC解密。**   1. **ECB/CBC检测预言机**   **编写一个在未知密钥下加密数据的函数，这个函数会生成一个随机密钥并用它来加密。在内部，让该函数在明文之前附加5-10个字节（随机选择数量），在明文之后也附加5-10个字节。**  **现在，让该函数在一半时间内选择使用ECB加密，另一半时间使用CBC加密。使用rand(2)来决定使用哪种模式，每次检测函数使用的是哪种分组密码模式。**   1. **一次字节ECB解密（简单）**   **将你的预言机函数复制到一个新函数中，该函数使用一个固定但未知的密钥在ECB模式下加密缓冲区。将给定字符串附加到明文后面，在附加之前先对字符串进行Base64解码。**   1. **ECB剪切粘贴攻击**   **编写一个函数，根据给定的电子邮件地址以该格式编码用户配置文件。现在，还有两个简单的函数，生成一个随机 AES 密钥，然后在密钥下加密编码的用户配置文件，将其"提供"给"攻击者"。仅使用用户对 profile\_for() 的输入（作为生成"有效"密文的预言机）和密文本身，制作一个 role=admin 的配置文件。**   1. **一次字节ECB解密（困难）**   **使用之前的预言机函数，生成一个随机长度的随机字节，并将这个字符串添加到每个明文之前。**   1. **PKCS#7填充验证**   **编写一个函数，该函数接收一个明文，判断其是否具有有效的 PKCS#7 填充，并去除填充。**   1. **CBC位翻转攻击**   **生成一个随机的AES密钥，结合填充代码和CBC代码来编写两个函数。**  **利用CBC模式的错误传播特性，通过精心修改密文来绕过输入验证，最终在解密后的明文中插入;admin=true;字段，从而获得管理员权限。** |
| **实验过程**  **问题1 AES 密钥 - 编码于证件机读区中**  **代码:**  **from hashlib import sha1**  **from base64 import b64decode**  **from Crypto.Cipher import AES**  **# 参数**  **C='9MgYwmuPrjiecPMx61O6zIuy3MtIXQQ0E59T3xB6u0Gyf1gYs2i3K9Jxaa0zj4gTMazJuApwd6+jdyeI5iGHvhQyDHGVlAuYTgJrbFDrfB22Fpil2NfNnWFBTXyf7SDI'**  **K = '12345678<8<<<1110182<111116?<<<<<<<<<<<<<<<4'**  **# 根据文献[2]求?**  **def solve\_num(k):**  **k = list(k)**  **weights = [7, 3, 1, 7, 3, 1]**  **sum = 0**  **for i in range(21, 27):**  **sum = (sum + int(k[i]) \* weights[i - 21]) % 10**  **k[27] = str(sum)**  **return ''.join(k)**  **# 求K\_seed**  **def getK\_seed(k):**  **mrz\_imt = k[:10] + k[13:20] + k[21:28]**  **H\_SHA1 = sha1(mrz\_imt.encode()).hexdigest()**  **return H\_SHA1[:32]**  **# 增加偶校验码以得到ka和kb**  **def getKab(k):**  **kab = []**  **a = bin(int(k, 16))[2:]**  **for i in range(0, len(a), 8):**  **kab.append(a[i:i + 7])**  **if a[i:i + 7].count('1') % 2 == 0:**  **kab.append('1')**  **else:**  **kab.append('0')**  **return hex(int(''.join(kab), 2))[2:]**  **# 根据ka和kb求Key**  **def getKey(k):**  **k = k + '00000001'**  **H = sha1(bytes.fromhex(k)).hexdigest()**  **return getKab(H[:16]) + getKab(H[16:32])**  **# 求明文**  **def getP(C, k):**  **C = b64decode(C)**  **aes = AES.new(bytes.fromhex(k), AES.MODE\_CBC, bytes.fromhex('0' \* 32))**  **return aes.decrypt(C).decode()**  **if \_\_name\_\_ == '\_\_main\_\_':**  **# 计算缺失的字符**  **K = solve\_num(K)**  **# 计算K\_seed**  **K\_seed = getK\_seed(K)**  **# 计算AES密钥**  **Key = getKey(K\_seed)**  **# 解密得到明文**  **P = getP(C, Key)**  **# 输出密钥和明文**  **print(f"密钥: {Key}")**  **print(f"明文: {P}")**  **结果：**    **问题2**   1. **实现PKCS#7填充**   **代码：**  **def pkcs7\_pad(data, block\_size):**  **padding\_len = block\_size - len(data) % block\_size**  **return data + bytes([padding\_len] \* padding\_len)**  **# 示例**  **original = b"YELLOW SUBMARINE"**  **padded = pkcs7\_pad(original, 20)**  **print(padded)**  **结果:**     1. **实现CBC模式**   **代码：**  **from Crypto.Cipher import AES**  **import base64**  **def xor\_bytes(a, b):**  **return bytes([x ^ y for x, y in zip(a, b)])**  **def pkcs7\_pad(data, block\_size):**  **padding\_len = block\_size - len(data) % block\_size**  **return data + bytes([padding\_len] \* padding\_len)**  **def pkcs7\_unpad(data):**  **padding\_len = data[-1]**  **return data[:-padding\_len]**  **def aes\_ecb\_encrypt(plaintext, key):**  **cipher = AES.new(key, AES.MODE\_ECB)**  **return cipher.encrypt(plaintext)**  **def aes\_ecb\_decrypt(ciphertext, key):**  **cipher = AES.new(key, AES.MODE\_ECB)**  **return cipher.decrypt(ciphertext)**  **def aes\_cbc\_encrypt(plaintext, key, iv):**  **block\_size = len(key)**  **plaintext = pkcs7\_pad(plaintext, block\_size)**  **ciphertext = b''**  **prev\_block = iv**  **for i in range(0, len(plaintext), block\_size):**  **block = plaintext[i:i + block\_size]**  **xored = xor\_bytes(block, prev\_block)**  **encrypted\_block = aes\_ecb\_encrypt(xored, key)**  **ciphertext += encrypted\_block**  **prev\_block = encrypted\_block**  **return ciphertext**  **def aes\_cbc\_decrypt(ciphertext, key, iv):**  **block\_size = len(key)**  **plaintext = b''**  **prev\_block = iv**  **for i in range(0, len(ciphertext), block\_size):**  **block = ciphertext[i:i + block\_size]**  **decrypted\_block = aes\_ecb\_decrypt(block, key)**  **plaintext\_block = xor\_bytes(decrypted\_block, prev\_block)**  **plaintext += plaintext\_block**  **prev\_block = block**  **return pkcs7\_unpad(plaintext)**  **# 解密给定的文件**  **def decrypt\_file():**  **# 读取并解码base64文件**  **with open('file\_2.txt', 'r') as f:**  **ciphertext\_b64 = f.read().strip()**  **ciphertext = base64.b64decode(ciphertext\_b64)**  **key = b"YELLOW SUBMARINE"**  **iv = b'\x00' \* 16**  **# CBC解密**  **plaintext = aes\_cbc\_decrypt(ciphertext, key, iv)**  **return plaintext.decode('utf-8', errors='ignore')**  **# 测试CBC加密解密**  **def test\_cbc():**  **test\_key = b"YELLOW SUBMARINE"**  **test\_iv = b'\x00' \* 16**  **test\_plaintext = b"Hello, this is a test message for CBC mode!"**  **# 加密**  **ciphertext = aes\_cbc\_encrypt(test\_plaintext, test\_key, test\_iv)**  **# 解密**  **decrypted = aes\_cbc\_decrypt(ciphertext, test\_key, test\_iv)**  **if \_\_name\_\_ == "\_\_main\_\_":**  **test\_cbc()**  **# 解密文件**  **decrypted\_text = decrypt\_file()**  **print("解密后的文件内容:")**  **print(decrypted\_text[:500] + "..." if len(decrypted\_text) > 500 else decrypted\_text)**  **结果：**  **可以得到解密后的文件内容为：**  **I'm back and I'm ringin' the bell**  **A rockin' on the mike while the fly girls yell**  **In ecstasy in the back of me**  **Well that's my DJ Deshay cuttin' all them Z's**  **Hittin' hard and the girlies goin' crazy**  **Vanilla's on the mike, man I'm not lazy.**  **I'm lettin' my drug kick in**  **It controls my mouth and I begin**  **To just let it flow, let my concepts go**  **My posse's to the side yellin', Go Vanilla Go!**  **Smooth 'cause that's the way I will be**  **And if you don't give a damn, then**  **Why you starin' at me**  **So ...**     1. **ECB/CBC检测预言机**   **代码：**  **import os**  **import random**  **from Crypto.Cipher import AES**  **# 生成随机AES密钥**  **def generate\_random\_key():**  **return os.urandom(16)**  **# 字节异或操作**  **def xor\_bytes(a, b):**  **return bytes([x ^ y for x, y in zip(a, b)])**  **# PKCS#7填充**  **def pkcs7\_pad(data, block\_size):**  **padding\_len = block\_size - len(data) % block\_size**  **return data + bytes([padding\_len] \* padding\_len)**  **# AES ECB加密**  **def aes\_ecb\_encrypt(plaintext, key):**  **cipher = AES.new(key, AES.MODE\_ECB)**  **return cipher.encrypt(plaintext)**  **# AES CBC加密**  **def aes\_cbc\_encrypt(plaintext, key, iv):**  **block\_size = len(key)**  **plaintext = pkcs7\_pad(plaintext, block\_size)**  **ciphertext = b''**  **prev\_block = iv**  **for i in range(0, len(plaintext), block\_size):**  **block = plaintext[i:i + block\_size]**  **xored = xor\_bytes(block, prev\_block)**  **encrypted\_block = aes\_ecb\_encrypt(xored, key)**  **ciphertext += encrypted\_block**  **prev\_block = encrypted\_block**  **return ciphertext**  **# 加密预言机：随机选择ECB或CBC模式加密数据**  **def encryption\_oracle(plaintext):**  **key = generate\_random\_key()**  **# 在明文前后添加5-10个随机字节**  **prefix\_len = random.randint(5, 10)**  **suffix\_len = random.randint(5, 10)**  **prefix = os.urandom(prefix\_len)**  **suffix = os.urandom(suffix\_len)**  **modified\_plaintext = prefix + plaintext + suffix**  **# 随机选择加密模式**  **if random.randint(0, 1) == 0:**  **# ECB模式**  **print("Oracle选择: ECB模式")**  **return aes\_ecb\_encrypt(pkcs7\_pad(modified\_plaintext, 16), key), "ECB"**  **else:**  **# CBC模式**  **print("Oracle选择: CBC模式")**  **iv = os.urandom(16)**  **return aes\_cbc\_encrypt(modified\_plaintext, key, iv), "CBC"**  **# 检测加密模式ECB/CBC**  **def detect\_ecb\_cbc(ciphertext):**  **block\_size = 16**  **blocks = [ciphertext[i:i + block\_size] for i in range(0, len(ciphertext), block\_size)]**  **# 检查是否有重复的块**  **unique\_blocks = set(blocks)**  **# 如果存在重复块，很可能是ECB模式**  **if len(blocks) != len(unique\_blocks):**  **return "ECB"**  **else:**  **return "CBC"**  **def advanced\_detection\_oracle():**  **# 创建包含重复块的输入（3个相同的块）**  **test\_input = b'A' \* 48 # 3个完整的AES块**  **ciphertext, actual\_mode = encryption\_oracle(test\_input)**  **detected\_mode = detect\_ecb\_cbc(ciphertext)**  **return actual\_mode, detected\_mode**  **# 演示**  **def simple\_demo():**  **print("ECB/CBC检测器演示")**  **# 创建一个明显包含重复块的输入**  **test\_data = b"AAAAAAAAAAAAAAAA" \* 4 # 4个相同的块**  **ciphertext, actual\_mode = encryption\_oracle(test\_data)**  **detected\_mode = detect\_ecb\_cbc(ciphertext)**  **print(f"实际加密模式: {actual\_mode}")**  **print(f"检测到的模式: {detected\_mode}")**  **print(f"检测结果: {'正确' if actual\_mode == detected\_mode else '错误'}")**  **if \_\_name\_\_ == "\_\_main\_\_":**  **simple\_demo()**  **print("\n")**  **结果：**     1. **一次字节ECB解密（简单）**   **代码：**  **import base64**  **from Crypto.Cipher import AES**  **from Crypto.Random import get\_random\_bytes**  **import os**  **# 全局随机密钥**  **global\_key = get\_random\_bytes(16)**  **# 要解密的未知字符串（base64编码）**  **unknown\_b64="""Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK"""**  **# 解码未知字符串**  **unknown\_string = base64.b64decode(unknown\_b64)**  **def encryption\_oracle(plaintext):**  **# 在用户输入后附加未知字符串**  **full\_plaintext = plaintext + unknown\_string**  **# 使用 PKCS7 填充**  **padding\_length = 16 - (len(full\_plaintext) % 16)**  **if padding\_length == 0:**  **padding\_length = 16**  **full\_plaintext += bytes([padding\_length]) \* padding\_length**  **# ECB 模式加密**  **cipher = AES.new(global\_key, AES.MODE\_ECB)**  **ciphertext = cipher.encrypt(full\_plaintext)**  **return ciphertext**  **def detect\_block\_size():**  **plaintext = b""**  **initial\_length = len(encryption\_oracle(plaintext))**  **# 不断增加输入长度，直到密文长度变化**  **for i in range(1, 100):**  **plaintext = b"A" \* i**  **new\_length = len(encryption\_oracle(plaintext))**  **if new\_length != initial\_length:**  **block\_size = new\_length - initial\_length**  **return block\_size**  **return None**  **def detect\_ecb(block\_size):**  **# 发送两个相同块的明文**  **plaintext = b"A" \* block\_size \* 3**  **ciphertext = encryption\_oracle(plaintext)**  **# 检查是否有重复的块**  **blocks = [ciphertext[i:i + block\_size] for i in range(0, len(ciphertext), block\_size)]**  **# 如果有重复的块，说明是 ECB 模式**  **return len(blocks) != len(set(blocks))**  **def byte\_at\_a\_time\_ecb\_decryption():**  **# 1. 检测块大小**  **block\_size = detect\_block\_size()**  **# 2. 检测 ECB 模式**  **is\_ecb = detect\_ecb(block\_size)**  **if not is\_ecb:**  **print("警告：不是 ECB 模式，攻击可能失败")**  **# 3. 确定未知字符串的长度**  **base\_length = len(encryption\_oracle(b""))**  **# 4. 逐字节解密**  **discovered = b""**  **# 对于每个字节位置进行解密**  **for byte\_pos in range(base\_length):**  **# 计算当前字节所在的块和块内位置**  **block\_index = byte\_pos // block\_size**  **byte\_in\_block = byte\_pos % block\_size**  **# 构造短一个字节的输入**  **padding\_length = block\_size - byte\_in\_block - 1**  **short\_input = b"A" \* padding\_length**  **# 获取目标密文块（我们想要解密的块）**  **target\_ciphertext = encryption\_oracle(short\_input)**  **target\_block = target\_ciphertext[block\_index \* block\_size:(block\_index + 1) \* block\_size]**  **# 构建字典：最后一个字节的所有可能值**  **dictionary = {}**  **# 对于所有可能的最后一个字节**  **for byte\_val in range(256):**  **# 构造测试输入：padding + 已知字节 + 测试字节**  **test\_input = short\_input + discovered + bytes([byte\_val])**  **# 确保我们只取第一个块进行比较**  **test\_ciphertext = encryption\_oracle(test\_input)**  **test\_block = test\_ciphertext[0:block\_size]**  **dictionary[test\_block] = bytes([byte\_val])**  **# 在字典中查找匹配的块**  **if target\_block in dictionary:**  **discovered\_byte = dictionary[target\_block]**  **discovered += discovered\_byte**  **else:**  **break**  **# 移除 PKCS7 填充**  **try:**  **padding\_length = discovered[-1]**  **if padding\_length <= block\_size:**  **# 检查填充是否有效**  **if discovered[-padding\_length:] == bytes([padding\_length]) \* padding\_length:**  **discovered = discovered[:-padding\_length]**  **except:**  **pass**  **return discovered**  **def main():**  **# 执行解密攻击**  **decrypted = byte\_at\_a\_time\_ecb\_decryption()**  **print("解密内容:")**  **print(decrypted.decode('utf-8'))**  **if \_\_name\_\_ == "\_\_main\_\_":**  **main()**  **结果：**     1. **ECB剪切粘贴攻击**   **代码：**  **import os**  **from Crypto.Cipher import AES**  **# 全局随机密钥**  **AES\_KEY = os.urandom(16)**  **def parse\_kv(cookie):**  **return dict(pair.split('=') for pair in cookie.split('&'))**  **# 生成用户配置文件**  **def profile\_for(email):**  **email = email.replace('&', '').replace('=', '')**  **return f"email={email}&uid=10&role=user"**  **# PKCS#7 填充**  **def pkcs7\_pad(data, block\_size):**  **padding\_len = block\_size - len(data) % block\_size**  **return data + bytes([padding\_len] \* padding\_len)**  **# PKCS#7 去除填充**  **def pkcs7\_unpad(data):**  **padding\_len = data[-1]**  **return data[:-padding\_len]**  **# 加密配置文件**  **def encrypt\_profile(profile):**  **cipher = AES.new(AES\_KEY, AES.MODE\_ECB)**  **padded\_data = pkcs7\_pad(profile.encode(), 16)**  **return cipher.encrypt(padded\_data)**  **# 解密密文并解析配置文件**  **def decrypt\_profile(ciphertext):**  **cipher = AES.new(AES\_KEY, AES.MODE\_ECB)**  **decrypted = cipher.decrypt(ciphertext)**  **return parse\_kv(pkcs7\_unpad(decrypted).decode())**  **# ECB 剪切粘贴攻击**  **def ecb\_cut\_and\_paste\_attack():**  **# 创建恰好填充一个块的email**  **block1\_email = "foo@bar.co"**  **profile1 = profile\_for(block1\_email)**  **cipher1 = encrypt\_profile(profile1)**  **# 获取第一个块（包含email部分）**  **block1 = cipher1[:16]**  **# 创建包含"admin"的块**  **block2\_email = "foo@bar.admin" + "\x0b" \* 11**  **profile2 = profile\_for(block2\_email[:13])**  **cipher2 = encrypt\_profile(profile2)**  **# 获取第二个块（包含"admin"和填充）**  **block2 = cipher2[16:32]**  **# 组合攻击**  **block3\_email = "foo@bar.xx"**  **profile3 = profile\_for(block3\_email)**  **cipher3 = encrypt\_profile(profile3)**  **# 构造恶意密文：email部分 + role=部分 + admin块**  **malicious\_cipher = cipher3[:32] + block2**  **# 解密验证**  **result = decrypt\_profile(malicious\_cipher)**  **return result**  **# 测试**  **if \_\_name\_\_ == "\_\_main\_\_":**  **email = "test@example.com"**  **profile = profile\_for(email)**  **print("配置文件:", profile)**  **encrypted = encrypt\_profile(profile)**  **decrypted = decrypt\_profile(encrypted)**  **print("解密结果:", decrypted)**  **结果：**     1. **一次字节ECB解密（困难）**   **代码：**  **import base64**  **from Crypto.Cipher import AES**  **from Crypto.Random import get\_random\_bytes**  **import random**  **# 目标Base64字符串**  **target\_b64 = "Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK"**  **target\_bytes = base64.b64decode(target\_b64)**  **# 随机密钥和前缀**  **key = get\_random\_bytes(16)**  **random\_prefix = get\_random\_bytes(random.randint(1, 100))**  **def encrypt(data):**  **plaintext = random\_prefix + data + target\_bytes**  **cipher = AES.new(key, AES.MODE\_ECB)**  **pad\_len = 16 - len(plaintext) % 16**  **return cipher.encrypt(plaintext + bytes([pad\_len]) \* pad\_len)**  **# 发现块大小**  **block\_size = 16**  **for i in range(1, 33):**  **if len(encrypt(b'A' \* i)) != len(encrypt(b'A' \* (i + 1))):**  **block\_size = len(encrypt(b'A' \* (i + 1))) - len(encrypt(b'A' \* i))**  **break**  **# 发现前缀长度**  **prefix\_len = 0**  **for pad in range(block\_size):**  **ct = encrypt(b'A' \* (pad + block\_size \* 2))**  **blocks = [ct[i:i + block\_size] for i in range(0, len(ct), block\_size)]**  **for i in range(len(blocks) - 1):**  **if blocks[i] == blocks[i + 1]:**  **prefix\_len = i \* block\_size - pad**  **break**  **if prefix\_len: break**  **# 计算对齐填充**  **prefix\_pad = (block\_size - prefix\_len % block\_size) % block\_size**  **offset = prefix\_len + prefix\_pad**  **# 逐字节解密**  **result = b""**  **for i in range(len(encrypt(b'A' \* prefix\_pad)) - offset):**  **shift = block\_size - 1 - (i % block\_size)**  **padding = b'A' \* (prefix\_pad + shift)**  **target\_block = encrypt(padding)[(offset + i) // block\_size \* block\_size:]**  **for byte in range(256):**  **test\_input = padding + result + bytes([byte])**  **test\_block = encrypt(test\_input)[(offset + i) // block\_size \* block\_size:]**  **if test\_block[:block\_size] == target\_block[:block\_size]:**  **result += bytes([byte])**  **break**  **print(result.decode('utf-8'))**  **结果：**    **可以看到第一行解密出的内容与第4题是一致的**   1. **PKCS#7填充验证**   **代码:**  **def validate\_and\_strip\_pkcs7(data):**  **# 验证并去除 PKCS#7 填充**  **if not data:**  **raise ValueError("数据为空")**  **pad\_length = data[-1]**  **# 验证填充长度和内容**  **if pad\_length < 1 or pad\_length > len(data) or data[-pad\_length:] != bytes([pad\_length]) \* pad\_length:**  **raise ValueError("无效的PKCS#7填充")**  **return data[:-pad\_length]**  **# 使用示例**  **if \_\_name\_\_ == "\_\_main\_\_":**  **# 有效填充**  **valid\_data = b"ICE ICE BABY\x04\x04\x04\x04"**  **result = validate\_and\_strip\_pkcs7(valid\_data)**  **print(f"有效填充结果: {result}")**  **# 无效填充**  **invalid\_data1 = b"ICE ICE BABY\x05\x05\x05\x05"**  **invalid\_data2 = b"ICE ICE BABY\x01\x02\x03\x04"**  **try:**  **validate\_and\_strip\_pkcs7(invalid\_data1)**  **except ValueError as e:**  **print(f"无效填充错误: {e}")**  **try:**  **validate\_and\_strip\_pkcs7(invalid\_data2)**  **except ValueError as e:**  **print(f"无效填充错误: {e}")**  **结果：**     1. **CBC位翻转攻击**   **代码：**  **import os**  **import random**  **from Crypto.Cipher import AES**  **from Crypto.Util.Padding import pad, unpad**  **class CBCBitFlippingAttack:**  **def \_\_init\_\_(self):**  **# 生成随机AES密钥**  **self.key = os.urandom(16)**  **self.iv = os.urandom(16) # 为了CBC模式**  **def quote\_special\_chars(self, data):**  **data = data.replace(';', '%3B')**  **data = data.replace('=', '%3D')**  **return data**  **# 第一个函数：加密用户数据**  **def encrypt\_userdata(self, userdata):**  **# 转义特殊字符**  **userdata\_quoted = self.quote\_special\_chars(userdata)**  **# 构建完整字符串**  **prefix = "comment1=cooking%20MCs;userdata="**  **suffix = ";comment2=%20like%20a%20pound%20of%20bacon"**  **plaintext = prefix + userdata\_quoted + suffix**  **# 加密**  **cipher = AES.new(self.key, AES.MODE\_CBC, self.iv)**  **ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block\_size))**  **return ciphertext**  **# 第二个函数：检查是否有admin权限**  **def check\_admin(self, ciphertext):**  **try:**  **# 解密**  **cipher = AES.new(self.key, AES.MODE\_CBC, self.iv)**  **decrypted = unpad(cipher.decrypt(ciphertext), AES.block\_size)**  **plaintext = decrypted.decode('latin-1') # 使用latin-1避免解码错误**  **# 检查是否包含";admin=true;"**  **if ";admin=true;" in plaintext:**  **return True, plaintext**  **return False, plaintext**  **except Exception as e:**  **return False, f"Decryption error: {str(e)}"**  **# 执行CBC位翻转攻击**  **def cbc\_bit\_flip\_attack(self):**  **print("=== CBC Bit Flipping Attack ===")**  **# 1. 首先加密一个已知的用户数据**  **user\_input = "XadminXtrueX" # 使用占位符**  **ciphertext = self.encrypt\_userdata(user\_input)**  **print(f"Original user input: {user\_input}")**  **print(f"Ciphertext length: {len(ciphertext)} bytes")**  **# 2. 检查原始密文是否包含admin权限**  **is\_admin, original\_plaintext = self.check\_admin(ciphertext)**  **print(f"Before attack - Admin access: {is\_admin}")**  **# 3. 分析块结构**  **prefix = "comment1=cooking%20MCs;userdata="**  **print(f"Prefix length: {len(prefix)} characters")**  **print(f"Prefix in hex: {prefix.encode().hex()}")**  **# 4. 执行位翻转攻击**  **ciphertext\_bytearray = bytearray(ciphertext)**  **flip\_0 = ord('X') ^ ord(';')**  **ciphertext\_bytearray[16 + 0] ^= flip\_0 # 影响第三个块的第一个字符**  **# 在位置6: 'X' -> '=' (0x58 -> 0x3D)**  **flip\_6 = ord('X') ^ ord('=')**  **ciphertext\_bytearray[16 + 6] ^= flip\_6 # 影响第三个块的第七个字符**  **# 在位置11: 'X' -> ';' (0x58 -> 0x3B)**  **flip\_11 = ord('X') ^ ord(';')**  **ciphertext\_bytearray[16 + 11] ^= flip\_11 # 影响第三个块的第十二个字符**  **modified\_ciphertext = bytes(ciphertext\_bytearray)**  **print("\nAfter bit flipping attack:")**  **is\_admin\_after, modified\_plaintext = self.check\_admin(modified\_ciphertext)**  **print(f"Admin access: {is\_admin\_after}")**  **return is\_admin\_after, modified\_ciphertext**  **if \_\_name\_\_ == "\_\_main\_\_":**  **# 执行攻击**  **attack = CBCBitFlippingAttack()**  **success, final\_ciphertext = attack.cbc\_bit\_flip\_attack()**  **if success:**  **print("\n Attack successful! Admin access gained!")**  **else:**  **print("\n Attack failed!")**  **结果：** |
| **总结**  **通过本系列密码学实验，我们深入实践了分组密码的核心加解密技术。在编程实现层面，我们系统构建了PKCS#7填充、ECB模式和CBC模式的完整加解密流程，运用字节级XOR运算、块分割与重组等基础密码学操作。在攻击方法探索中，我们基于ECB模式的确定性缺陷，成功实现了通过重复块检测识别加密模式、通过精心构造输入实施逐字节解密的攻击方案，并进一步突破了随机前缀的防护。在CBC模式分析中，我们理解了初始化向量的重要作用和错误传播机制。整个实验过程凸显了密码学实现中的关键细节：数据边界处理、块大小对齐、填充验证安全等，深刻揭示了即使算法本身安全，实现不当仍会导致严重漏洞，为构建安全的密码学系统提供了宝贵的实践经验。** |
| **参考文献**  **《现代密码学》（第四版）（杨波编著）**  http://www.cryptopals.com  https://www.mysterytwisterc3.org  https://www.coursera.org/learn/crypto  http://www2.icao.int |