|  |
| --- |
| **Github账号：Ystar0511** |
| **个人博客关于密码学实验的链接：**  **https://github.com/Ystar0511/-/tree/main/%E5%AF%86%E7%A0%81%E5%AD%A6%E5%AE%9E%E9%AA%8C** |
| **实验题目：**  **1、[MTC3 AES 密钥 --- 编码于欧洲电子护照的机读区中]{.mark}** [**https://www.mysterytwisterc3.org/en/challenges/level-2/aes-key--encoded-in-the-machine-readable-zone-of-a-european-epassport**](https://www.mysterytwisterc3.org/en/challenges/level-2/aes-key--encoded-in-the-machine-readable-zone-of-a-european-epassport)  **2、<http://www.cryptopals.com/sets/2>**  **(1) 实现 PKCS#7 填充**  **(2) 实现 CBC 模式**  **(3) ECB/CBC 检测预言机**  **(4) 逐字节 ECB 解密（简单）**  **(5) ECB 剪切与粘贴**  **(6) [逐字节 ECB 解密（更难）]**  **(7) PKCS#7 填充验证**  **(8) [CBC 位翻转攻击]** |
| **实验摘要：**  **第一部分：MTC3挑战**   1. **AES密钥恢复**：分析与解码编码在欧洲电子护照机读区中的AES密钥。   **第二部分：Cryptopals挑战（第二组）**   1. **PKCS#7填充实现**：编写程序实现PKCS#7填充方案。 2. **CBC模式实现**：实现密码块链接模式加解密。 3. **加密模式检测预言机**：设计能够检测加密算法是ECB还是CBC模式的方法。 4. **简单逐字节ECB解密**：通过分析ECB加密特性，逐步恢复未知明文。 5. **ECB剪切粘贴攻击**：利用ECB模式弱点，通过重组密文构造特定明文。 6. **增强逐字节ECB解密**：在更复杂的场景下实现逐字节ECB解密。 7. **PKCS#7填充验证**：编写程序验证解密后数据的填充是否符合PKCS#7标准。 8. **CBC位翻转攻击**：通过修改密文位，在解密时引发目标明文位的改变。 |
| **题目描述：**  **本实验旨在通过一系列实践任务，加深对现代密码学中核心概念与常见攻击方法的理解。实验主要分为两大部分。**  **第一部分是一个来自MTC3平台的具体挑战，要求我们分析与恢复一个嵌入在欧洲电子护照机读区中的AES密钥。这模拟了现实世界中一个经典的密码学应用与安全问题，需要我们理解数据编码格式并运用相应的密码分析技术来提取关键信息。**  **第二部分则是一套系统性的编程练习，源自著名的Cryptopals挑战。这部分要求我们从基础构建块做起，首先亲手实现PKCS#7填充方案和CBC加密模式，这是理解分组密码工作原理的基石。随后，我们将利用这些基础组件，去探索和利用加密算法中的弱点。**  **例如，我们需要创建一个“检测预言机”来区分加密使用的是ECB模式还是CBC模式；我们将在不知道密钥的情况下，通过观察和分析，采用“逐字节”的方式破解ECB加密的明文，这在简单和复杂两种场景下进行。我们还会实践“ECB剪切粘贴攻击”，通过重组密文块来伪造有意义的明文，深刻揭示ECB模式在缺乏完整性保护下的致命缺陷。**  **最后，实验将引导我们完成PKCS#7填充的有效性验证，并实施CBC位翻转攻击。后者通过精心篡改密文，使得解密后的明文发生特定改变，这生动地展示了即使强加密算法在错误的使用模式下（如未经验证的CBC模式）也可能变得脆弱。**  **总而言之，本实验通过“先构建，再破解”的思路，让我们在动手实践中全面掌握对称加密的模式、填充标准以及几种典型且强大的密码分析攻击技术。** |
| **过程：**  **1、代码**  from hashlib import sha1 import codecs import base64 from Crypto.Cipher import AES import binascii   # 求未知数，是到期日的校验位，根据校验规则计算 def Unknown\_Number() -> int:  Unknown\_Number = 0  number = "111116" # 预设的数字  weight = "731" # 权重  for i in range(0, len(number)):  Unknown\_Number += int(number[i]) \* int(weight[i % 3])  return Unknown\_Number % 10 # 返回校验位   # 计算k\_seed def cal\_Kseed() -> str:  MRZ\_information = "12345678<811101821111167" # 护照信息  H\_information = sha1(MRZ\_information.encode()).hexdigest() # 使用SHA1进行哈希  K\_seed = H\_information[0:32] # 取哈希值的前32位作为K\_seed  return K\_seed   def cal\_Ka\_Kb(K\_seed):  c = "00000001"  d = K\_seed + c  H\_d = sha1(codecs.decode(d, "hex")).hexdigest() # 对K\_seed进行哈希  ka = H\_d[0:16] # 取前16位作为ka  kb = H\_d[16:32] # 取后16位作为kb  return ka, kb   # 对Ka和Kb分别进行奇偶校验，得到新的k1和k2 def Parity\_Check(x):  k\_list = []  a = bin(int(x, 16))[2:] # 将16进制转为2进制  for i in range(0, len(a), 8):  # 7位一组分块，计算一个校验位，使1的个数为偶数  if (a[i:i + 7].count("1")) % 2 == 0:  k\_list.append(a[i:i + 7])  k\_list.append('1')  else:  k\_list.append(a[i:i + 7])  k\_list.append('0')  k = hex(int(''.join(k\_list), 2)) # 将2进制转为16进制  return k   if \_\_name\_\_ == "\_\_main\_\_":  K\_seed = cal\_Kseed() # 计算K\_seed  ka, kb = cal\_Ka\_Kb(K\_seed) # 计算ka和kb  k\_1 = Parity\_Check(ka) # 对ka进行奇偶校验  k\_2 = Parity\_Check(kb) # 对kb进行奇偶校验  key = k\_1[2:] + k\_2[2:] # 合并k\_1和k\_2作为最终的密钥  print("密钥是：",key) # 输出密钥   # 待解密的密文  ciphertext = base64.b64decode(  "9MgYwmuPrjiecPMx61O6zIuy3MtIXQQ0E59T3xB6u0Gyf1gYs2i3K9Jxaa0zj4gTMazJuApwd6+jdyeI5iGHvhQyDHGVlAuYTgJrbFDrfB22Fpil2NfNnWFBTXyf7SDI")  IV = '0' \* 32 # 初始化向量   # 使用AES进行解密  m = AES.new(binascii.unhexlify(key), AES.MODE\_CBC, binascii.unhexlify(IV)).decrypt(ciphertext)  print("明文是：",m) # 输出解密后的明文  **输出结果：**  **2、**  **Byte-at-a-time ECB decryption**  **代码**  import os import base64 import random from Crypto.Cipher import AES from AES\_CBC import \*  b64\_string = "Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK" random\_key = os.urandom(16) random\_string = os.urandom(random.randint(0, 255))   def AES128\_harder(text: bytes) -> bytes:  global b64\_string, random\_key, random\_string   secret\_string = base64.b64decode(b64\_string)  plaintext = random\_string + text + secret\_string # 随机前缀 + 攻击者控制 + 目标字节  cipher = AES\_ECB\_encrypt(plaintext, random\_key)  return cipher   def break\_AES\_ECB\_harder(keysize: int, encryptor: callable) -> bytes:  # 寻找前缀长度  padding = 0  random\_blocks = 0  cipher\_length = len(encryptor(b''))  prefix\_length = len(os.path.commonprefix([encryptor(b'AAAA'), encryptor(b'')]))  print("Prefix length: ", prefix\_length)   # 查找随机块的数量  for i in range(int(cipher\_length / keysize)):  if prefix\_length < i \* keysize:  random\_blocks = i  break  print("Random blocks: ", random\_blocks)   # 查找所需的字节填充数  base\_cipher = encryptor(b'')  for i in range(1, keysize):  new\_cipher = encryptor(b'A' \* i)  new\_prefix\_length = len(os.path.commonprefix([base\_cipher, new\_cipher]))  if new\_prefix\_length > prefix\_length:  padding = i - 1  break  base\_cipher = new\_cipher  print("Number of bytes of padding required: ", padding)   deciphered = b""  ciphertext = encryptor(deciphered)  # 添加了填充，增加了一个块  run = len(ciphertext) + keysize   for i in range(keysize \* random\_blocks + 1, run + 1):  template = b'A' \* (run - i + padding)  cipher = encryptor(template)  for j in range(256):  # print(i, j)  text = template + deciphered + j.to\_bytes(1, "little")  c = encryptor(text)  if c[run - keysize:run] == cipher[run - keysize:run]:  deciphered += chr(j).encode()  break  return PKCS7\_unpad(deciphered)   keysize = 16 byte\_text = break\_AES\_ECB\_harder(keysize, AES128\_harder) print("\nDeciphered string:\n") print(byte\_text.decode("utf-8").strip())  **输出结果**    **PKCS#7 padding validation**  **代码**  from AES\_CBC import \*  given\_string = "ICE ICE BABY\x04\x04\x04\x04" target\_string = "ICE ICE BABY"  try:  target\_string.encode() == PKCS7\_unpad(given\_string.encode())  print("Correct Padding!") except:  print("Incorrect Padding!")  **输出结果**  Correct Padding!  **CBC bitflipping attacks**  **代码**  import os import random from Crypto.Cipher import AES from AES\_CBC import \*  prepend\_string = "comment1=cooking%20MCs;userdata=" append\_string = ";comment2=%20like%20a%20pound%20of%20bacon" parameter = b";admin=true;"  keysize = 16 random\_key = os.urandom(keysize) IV = os.urandom(keysize)   def encryptor(text: bytes, IV: bytes, key: bytes) -> bytes:  # 将给定的字符串添加到自定义文本中，并通过AES\_CBC模式进行加密   plaintext = (prepend\_string.encode() + text + append\_string.encode()).replace(b';', b'";"').replace(b'=', b'"="')  ciphertext = AES\_CBC\_encrypt(PKCS7\_pad(plaintext, len(key)), IV, key)  return ciphertext   def decryptor(byte\_string: bytes, IV: bytes, key: bytes) -> bool:  # 通过AES\_CBC模式解密给定的密文并检查admin是否设置为true   decrypted\_string = PKCS7\_unpad(AES\_CBC\_decrypt(byte\_string, IV, key))  if b";admin=true;" in decrypted\_string:  return True  else:  return False   def CBC\_bit\_flipping(parameter: bytes, keysize: int, encryptor: callable) -> bytes:  # 填充   padding = 0  random\_blocks = 0 # 寻找前缀长度  cipher\_length = len(encryptor(b'', IV, random\_key))  prefix\_length = len(os.path.commonprefix([encryptor(b'AAAA', IV, random\_key), encryptor(b'', IV, random\_key)]))  print("Prefix length: ", prefix\_length)   # 查找随机块的数量  for i in range(int(cipher\_length / keysize)):  if prefix\_length < i \* keysize:  random\_blocks = i  break  print("Random blocks: ", random\_blocks)   # 查找所需的字节填充数  base\_cipher = encryptor(b'', IV, random\_key)  for i in range(1, keysize):  new\_cipher = encryptor(b'A' \* i, IV, random\_key)  new\_prefix\_length = len(os.path.commonprefix([base\_cipher, new\_cipher]))  if new\_prefix\_length > prefix\_length:  padding = i - 1  break  base\_cipher = new\_cipher  print("Number of bytes of padding required: ", padding)   # 翻转给定字符串的字节  input\_text = b'A' \* padding + b"heytheremama"  string = parameter  modified\_string = b""  ciphertext = encryptor(input\_text, IV, random\_key)  for i in range(len(string)):  modified\_string += (  ciphertext[i + (random\_blocks - 1) \* keysize] ^ (input\_text[i + padding] ^ string[i])).to\_bytes(1,  "big")   modified\_ciphertext = ciphertext[:(random\_blocks - 1) \* keysize] + modified\_string + ciphertext[(  random\_blocks - 1) \* keysize + len(  modified\_string):]   return modified\_ciphertext   modified\_ciphertext = CBC\_bit\_flipping(parameter, keysize, encryptor) print(AES\_CBC\_decrypt(modified\_ciphertext, IV, random\_key))  **输出结果**  Prefix length: 32  Random blocks: 3  Number of bytes of padding required: 10  b'comment1"="cooking%20MCs";"userd\xc3\x9c\xc3\xaf\x13\xc2\xa0\x10k\xc3\x90\xc2\x94(Q\x14S\xc2\xb4z\xc3\xb2\xc2\x85;admin=true;";"comment2"="%20like%20a%20pound%20of%20bacon\x06\x06\x06\x06\x06\x06' |
| **总结**  **本实验聚焦于对称密码学的核心应用与安全性分析。主要内容包括两部分：一是实战解析欧洲电子护照机读区中的AES密钥，体现密码学在现实安全场景中的应用；二是系统完成Cryptopals挑战，涵盖PKCS#7填充、CBC模式等基础实现，以及ECB/CBC检测、逐字节ECB解密、剪切粘贴攻击、位翻转攻击等关键攻击技术。**  **通过"先实现后攻破"的实践路径，实验让我们深刻理解到：即使采用高强度加密算法，若使用模式不当或缺乏完整性保护，仍会面临严重安全威胁。这对培养密码系统设计与漏洞分析能力具有重要意义。** |
| **参考文献**  **无** |