|  |
| --- |
| **Github（或者Coding）账号：phitooo** |
| **个人博客关于密码学实验的链接：** [Phitooo/XD-Crypto-Experiment (github.com)](https://github.com/Phitooo/XD-Crypto-Experiment) |
| **实验题目（中文）：实验二** |
| **实验摘要（中文）：**  本次试验主要是块密码和应用实例。 |
| **题目描述**  **任务一**  转发了一条 AES 加密信息（CBC 模式带零初始化向量和 01-00 填充）。此外，还收到了相应的密钥，遗憾的是，该密钥并不完整，其形式类似于身份证件上的机器可读区 (MRZ)。我们的目标是找出以下基 64 编码信息的明文。我们的目标是找到以下 base64 编码信息的明文。  9MgYwmuPrjiecPMx61O6zIuy3MtIXQQ0E59T3xB6u0Gyf1gYs2i3K9Jx  aa0zj4gTMazJuApwd6+jdyeI5iGHvhQyDHGVlAuYTgJrbFDrfB22Fpil2NfNnWFBTXyf7SDI  用于加密的密钥 KENC 是基于基本访问控制（BAC）协议生成的。  (BAC) 协议的 KENC 密钥。解密时传输了以下字符，从中可以推导出 KENC  这些字符的编码方式见不幸的是，在传输过程中丢失了一个字符。  我们可以使用 CrypTool 1.4.30 进行加密操作。在解密前对 base64 代码进行解码  **任务二**    **任务三** |
| **过程（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）**  **任务一**  一个密码应用实例，思路很好理解，主要考察对密码组件的应用，较难的地方在于对文档的理解。   1. from hashlib import sha1 2. import codecs 3. import base64 4. from Crypto.Cipher import AES 5. import binascii 7. """求未知数，是到期日的校验位，根据校验规则计算""" 8. def Unknown\_Number() -> int: 9. Unknown\_Number = 0 10. number = "111116" 11. weight = "731" 12. for i in range(0, len(number)): 13. Unknown\_Number += int(number[i]) \* int(weight[i % 3]) 14. return Unknown\_Number % 10  17. """计算k\_seed""" 18. def cal\_Kseed() -> str: 19. MRZ\_information = "12345678<811101821111167" 20. *# 护照号码+校验位+出生日期+校验位+到期日+校验位(包括"<"符号)* 21. H\_information = sha1(MRZ\_information.encode()).hexdigest() 22. *# 十六进制字符串* 23. K\_seed = H\_information[0:32] 24. *# 最高有效16字节用作K\_seed* 25. return K\_seed  28. def cal\_Ka\_Kb(K\_seed): 29. c = "00000001" 30. d = K\_seed + c 31. H\_d = sha1(codecs.decode(d, "hex")).hexdigest() 32. *# 十六进制先变为二进制散列再转换成十六进制* 33. ka = H\_d[0:16] 34. kb = H\_d[16:32] 35. return ka, kb  38. """对Ka和Kb分别进行奇偶校验，得到新的k1和k2""" 39. def Parity\_Check(x): 40. k\_list = [] 41. a = bin(int(x, 16))[2:] 42. *# 16进制字符串转2进制字符串* 43. for i in range(0, len(a), 8): 44. *# 7位一组分块，计算一个校验位，使1的个数为偶数* 45. *# 舍弃原来的第8位* 46. if (a[i:i + 7].count("1")) % 2 == 0: 47. k\_list.append(a[i:i + 7]) 48. k\_list.append('1') 49. else: 50. k\_list.append(a[i:i + 7]) 51. k\_list.append('0') 53. k = hex(int(''.join(k\_list), 2)) 54. return k  57. if \_\_name\_\_ == "\_\_main\_\_": 58. K\_seed = cal\_Kseed() 59. ka, kb = cal\_Ka\_Kb(K\_seed) 60. *# print(ka, kb)* 61. k\_1 = Parity\_Check(ka) 62. k\_2 = Parity\_Check(kb) 63. *# print(k\_1, k\_2)* 64. key = k\_1[2:] + k\_2[2:] 65. print(key) 67. ciphertext = base64.b64decode( 68. "9MgYwmuPrjiecPMx61O6zIuy3MtIXQQ0E59T3xB6u0Gyf1gYs2i3K9Jxaa0zj4gTMazJuApwd6+jdyeI5iGHvhQyDHGVlAuYTgJrbFDrfB22Fpil2NfNnWFBTXyf7SDI") 69. IV = '0' \* 32 71. m = AES.new(binascii.unhexlify(key), AES.MODE\_CBC, binascii.unhexlify(IV)).decrypt(ciphertext) 72. print(m)   **结果：**    **任务二**  求target-bytes的长度和加密块的大小。  （1）通过不断对"A"、"AA"、"AAA"……加密，通过密文块的长度以及填充规则，可以得到random-prefix + target-bytes的总长度以及加密块的大小。  （2）加密两个块大小的"A"列表，然后不断的添加"A"，当出现两个相同的块的时候，可以得到单独random-prefix的填充长度。  （3）加密三个块大小的"A"，找到其中的两个相同块，则第一个相同块前面部分为random-prefix及其填充，从而可以得到random-prefix的长度；从总长度中减掉random-prefix长度，即为target-bytes的长度。   1. import os 2. import base64 3. import random 4. from Crypto.Cipher import AES 5. from AES\_CBC import \* 7. b64\_string = "Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkgaGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK" 8. random\_key = os.urandom(16) 9. random\_string = os.urandom(random.randint(0, 255))  12. def AES128\_harder(text: bytes) -> bytes: 13. global b64\_string, random\_key, random\_string 15. secret\_string = base64.b64decode(b64\_string) 16. plaintext = random\_string + text + secret\_string  *# 随机前缀 + 攻击者控制 + 目标字节* 17. cipher = AES\_ECB\_encrypt(plaintext, random\_key) 18. return cipher  21. def break\_AES\_ECB\_harder(keysize: int, encryptor: callable) -> bytes: 22. *# 寻找前缀长度* 23. padding = 0 24. random\_blocks = 0 25. cipher\_length = len(encryptor(b'')) 26. prefix\_length = len(os.path.commonprefix([encryptor(b'AAAA'), encryptor(b'')])) 27. print("Prefix length: ", prefix\_length) 29. *# 查找随机块的数量* 30. for i in range(int(cipher\_length / keysize)): 31. if prefix\_length < i \* keysize: 32. random\_blocks = i 33. break 34. print("Random blocks: ", random\_blocks) 36. *# 查找所需的字节填充数* 37. base\_cipher = encryptor(b'') 38. for i in range(1, keysize): 39. new\_cipher = encryptor(b'A' \* i) 40. new\_prefix\_length = len(os.path.commonprefix([base\_cipher, new\_cipher])) 41. if new\_prefix\_length > prefix\_length: 42. padding = i - 1 43. break 44. base\_cipher = new\_cipher 45. print("Number of bytes of padding required: ", padding) 47. deciphered = b"" 48. ciphertext = encryptor(deciphered) 49. *# 添加了填充，增加了一个块* 50. run = len(ciphertext) + keysize 52. for i in range(keysize \* random\_blocks + 1, run + 1): 53. template = b'A' \* (run - i + padding) 54. cipher = encryptor(template) 55. for j in range(256): 56. *# print(i, j)* 57. text = template + deciphered + j.to\_bytes(1, "little") 58. c = encryptor(text) 59. if c[run - keysize:run] == cipher[run - keysize:run]: 60. deciphered += chr(j).encode() 61. break 62. return PKCS7\_unpad(deciphered)  65. keysize = 16 66. byte\_text = break\_AES\_ECB\_harder(keysize, AES128\_harder) 67. print("\nDeciphered string:\n") 68. print(byte\_text.decode("utf-8").strip())   **结果：**    **任务三**  CBC的比特翻转攻击，通过添加很长的data构造出一个没有意义的块，然后异或这个块的某个密文字节，这样下一块的明文字节在解密时就会被篡改。   1. from Crypto.Cipher import AES 2. from Crypto import Random 3. import re  6. def pad(value, size): 7. """applies PKCS#7 padding to `value` to make it of size `size` 8. Args: 9. value (bytes): a byte-string to pad 10. size (size): the required size 11. Returns: 12. bytes: a byte-string = `value` + padding, such that its size is `size` 13. """ 14. if len(value) % size == 0: 15. return value 16. padding = size - len(value) % size 17. padValue = bytes([padding]) \* padding 18. return value + padValue  21. class InvalidPaddingError(Exception): 22. """exception class for invalid PKCS#7 padding 23. Args: 24. Exception (class): inherits from the in-built `Exception` class 25. """ 27. def \_\_init\_\_(self, paddedMsg, message="has invalid PKCS#7 padding."): 28. self.paddedMsg = paddedMsg 29. self.message = message 30. super().\_\_init\_\_(self.message) 32. def \_\_repr\_\_(self): 33. return f"{self.paddedMsg} {self.message}"  36. def valid\_padding(paddedMsg, block\_size): 37. """checks if `paddedMsg` has valid PKCS#7 padding for given `block\_size` 38. Args: 39. paddedMsg (bytes): the padded text 40. block\_size (int): the block size that is to be obtained by padding 41. Returns: 42. bool: True, if the padding is valid. False, otherwise. 43. """ 44. *# if the length of the `paddedMsg` is not a multiple of `block\_size`* 45. if len(paddedMsg) % block\_size != 0: 46. return False 48. last\_byte = paddedMsg[-1] 50. *# if the value of the last\_byte is greater than or equal to block\_size* 51. if last\_byte >= block\_size: 52. return False 54. padValue = bytes([last\_byte]) \* last\_byte 55. *# if all the padding bytes are not the same* 56. if paddedMsg[-last\_byte:] != padValue: 57. return False 59. *# if, after removing the padding, the remaining characters are not all printable* 60. if not paddedMsg[:-last\_byte].decode('ascii').isprintable(): 61. return False 63. return True  66. def remove\_padding(paddedMsg, block\_size): 67. """removes padding from `paddedMsg`, displays error-message if padding is invalid 68. Args: 69. paddedMsg (bytes): the message that is padded using PKCS#7 padding 70. block\_size (int): the block size that is obtained by said padding 71. Raises: 72. InvalidPaddingError: if the padding is invalid 73. Returns: 74. (byte): the message after removal of padding, if valid. 75. """ 76. if not valid\_padding(paddedMsg, block\_size): 77. raise InvalidPaddingError 79. last\_byte = paddedMsg[-1] 80. unpadded = paddedMsg[:-last\_byte] 81. return unpadded  84. *# this is the dictionary for replacements* 85. QUOTE = {b';': b'%3B', b'=': b'%3D'} 87. KEY = Random.new().read(AES.block\_size) 88. IV = bytes(AES.block\_size)  *# for simplicity just a bunch of 0's*  91. def cbc\_encrypt(input\_text): 92. """encrypts a the `input\_text` using AES-128 in CBC mode, 93. aftr replacing `:` with `%3B` and `=` with `%3D` in the `input\_text` 94. Args: 95. input\_text (byte): the input to encrypt 96. Returns: 97. byte: result of AES-128-CBC(prefix+input\_text+suffix) 98. """ 100. prepend = b"comment1=cooking%20MCs;userdata=" 101. append = b";comment2=%20like%20a%20pound%20of%20bacon" 103. for key in QUOTE: 104. input\_text = re.sub(key, QUOTE[key], input\_text) 106. plaintext = prepend + input\_text + append 107. plaintext = pad(plaintext, AES.block\_size) 109. cipher = AES.new(KEY, AES.MODE\_CBC, IV) 110. ciphertext = cipher.encrypt(plaintext) 112. return ciphertext  115. def check(ciphertext): 116. """checks if the `ciphertext` upon decryption has `;admin=true;` in it 117. Args: 118. ciphertext (bytes): the result of an AES-128-CBC encryption 119. Returns: 120. bool: True if the plain-text contains `;admin=true;`, False otherwise 121. """ 123. cipher = AES.new(KEY, AES.MODE\_CBC, IV) 124. plaintext = cipher.decrypt(ciphertext) 125. print(f"Plaintext: {plaintext}") 127. if b";admin=true;" in plaintext: 128. return True 130. return False  133. def test(): 134. """tests the injection of `;admin=true;` into the cipher-text 135. """ 137. *# send two blocks of just A's* 138. input\_string = b'A' \* AES.block\_size \* 2 139. ciphertext = cbc\_encrypt(input\_string) 141. *# replace first block of A's with the `required` plain-text* 142. required = pad(b";admin=true;", AES.block\_size) 143. *# xor each byte of the required with each byte of second block i.e, with 'A'* 144. inject = bytes([r ^ ord('A') for r in required]) 146. *# extra = length of ciphertext - length of injected text - length of prefix* 147. *# = one block of input + suffix* 148. extra = len(ciphertext) - len(inject) - 2 \* AES.block\_size 149. *# keep `inject` fill either side with 0's to match length with original ciphertext* 150. *# xor with 0 does not change value* 151. *# this replaces the first block of input with `required` while the rest is unchanged* 152. inject = bytes(2 \* AES.block\_size) + inject + bytes(extra) 154. *# to craft cipher-text, xor the `inject` bytes with* 155. *# corresponding byte of the ciphertext* 156. crafted = bytes([x ^ y for x, y in zip(ciphertext, inject)]) 158. if check(crafted): 159. print("Admin Found") 160. else: 161. print("Admin Not Found")  164. if \_\_name\_\_ == "\_\_main\_\_": 165. test()   **结果：**  **‘** |
| **总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）**  通过这次实验，学习了对分组密码CBC、ECB模式的多种攻击，包括Padding Oracle Attack，对加密模式有了更加深刻的理解。看懂题目很重要，要善于快速查找资料 知道攻击原理，数学知识很重要，要加强自己的编程能力 直接使用 Python 的内置库会方便很多。 |
| **参考文献（包括参考的书籍，论文，URL等，很重要）**  https://blog.csdn.net/flurry\_rain/article/details/78216841 https://github.com/HPShark/PA2-AES <https://www.jianshu.com/p/833582b2f560>  https://blog.csdn.net/weixin\_48392428/article/details/117227205 https://blog.csdn.net/Koz\_0/article/details/109540921 |