|  |
| --- |
| **Github账号：**acetyl-lwx |
| **个人博客关于密码学实验的链接：** https://www.cnblogs.com/Yanami-8/articles/18471089 |
| **实验题目：**现代密码学实验二 |
| **实验摘要：**  实验二主要是关于对称密码中的分组密码（块密码），具体有AES的ECB加密模式在大部分情况下都是不安全的，AES的CBC加密模式针对CCA是不安全的，有CBC翻转攻击的可能性。 |
| **题目描述**  T1 题目描述的较为冗长，实际上大致就是护照上有一部分信息泄露了，通过查阅文献，可以通过校验和以及后续很多操作恢复密钥，然后通过密钥恢复AES加密的密文。  T2-1 单纯地实现PKCS#7的填充方式。  T2-2 这里就是让我们实现块密码的CBC模式，利用给出的初始向量IV和密钥k，对文件进行解密。  T2-3 题目意思大概是写一个函数根据随机密钥进行加密，在加密前在明文的前后各添加5到10字节随机值，随机选取加密方式为ECB或CBC，如果是CBC，还需要选择随机的IV。根据加密的信息，判断是使用ECB加密还是CBC加密。  T2-4 题目大体意思就是你可以通过控制明文前的前缀破解ECB加密的密文。  T2-5 题目大意就是给你一个用户名和基于该用户信息加密后的密文，修改密文使其能解密为合法的明文，且该明文中的用户名变成role。  T2-6 题目是大体与T2-4一致，不过在加密时会添加一个固定的随机产生的前缀。  T2-7 就是写一个验证padding是否合法的程序。  T2-8 题目大意就是在CBC加密模式下。可以通过修改IV和密文块实现对明文块的控制。 |
| **过程**  T1  这道题的主要难度就是查阅文献，没有什么特殊技巧，我就简单描述一下解密流程。   1. 首先恢复泄露信息中的?，经过查阅题目提供的文献[2]，可以得知“?”是前面“111116”根据一定权重得到的校验和，根据文献[2]即可恢复“?”。 2. 通过泄露信息中的“12345678”“1110182”“111116?”这三部分（被称为MRZ），计算MRZ的SHA-1哈希值能够得到kseed。 3. 在kseed后面加上'00000001'比特串得到比特串D，计算D的SHA-1哈希值，哈希值的前16位为ka，16-32位为kb。 4. 对3得到的ka或kb，将其第8\*i位改成这位前7个数字的偶校验码，得到最终的ka和kb。 5. 将4得到的ka和kb相接，即得到最终用于AES加密的秘钥Key。 6. 利用秘钥Key解密得到明文。   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\_\_':      K = solve\_num(K)      K\_seed = getK\_seed(K)      Key = getKey(K\_seed)      P = getP(C, Key)      print(P)  T2-1  单纯地实现PKCS#7的填充方式。由于该情况下没有传输错误等情况，因此我就只写了填充和去填充的代码，没有写检验填充是否正确的代码。  代码：  # 添加padding  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  message\_pading = pad(b'YELLOW SUBMARINE', 16)  print(message\_pading)  print(unpad(message\_pading))  T2-2  题目提示我们可以使用这套题前面的ECB加密和异或代码，不过这套题前面两道没有写啊，所以咱自己实现就好了，至于ECB直接使用库中包装好的函数就好了，然后按CBC模式设计即可。  代码：  from Crypto.Cipher import AES  from base64 import b64decode  # 添加padding  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  # AES ECB模式加密  def AES\_ECB\_encrypt(plaintext: bytes, key: bytes) -> bytes:      cipher = AES.new(key, AES.MODE\_ECB)      return cipher.encrypt(pad(plaintext, AES.block\_size))  #AES ECB模式解密  def AES\_ECB\_decrypt(ciphertext: bytes, key: bytes) -> bytes:      cipher = AES.new(key, AES.MODE\_ECB)      return cipher.decrypt(ciphertext)  #异或  def xor(a: bytes, b: bytes) -> bytes:      return bytes(x ^ y for x, y in zip(a, b))  #AES CBC模式加密  def AES\_CBC\_encrypt(plaintext: bytes, key: bytes, iv: bytes) -> bytes:      cipher=b''      prev=iv      plaintext=pad(plaintext, AES.block\_size)      for i in range(0, len(plaintext), AES.block\_size):          current\_plaintext\_block = plaintext[i:i + AES.block\_size]          block\_cipher\_input=xor(current\_plaintext\_block, prev)          block\_cipher\_output=AES\_ECB\_encrypt(block\_cipher\_input, key)          cipher+=block\_cipher\_output          prev=block\_cipher\_output      return cipher  #AES CBC模式解密  def AES\_CBC\_decrypt(ciphertext: bytes, key: bytes, iv: bytes) -> bytes:      plaintext=b''      prev=iv      for i in range(0, len(ciphertext), AES.block\_size):          current\_ciphertext\_block = ciphertext[i:i + AES.block\_size]          block\_plaintext\_input=AES\_ECB\_decrypt(current\_ciphertext\_block, key)          block\_plaintext\_output=xor(block\_plaintext\_input, prev)          plaintext+=block\_plaintext\_output          prev=current\_ciphertext\_block      return plaintext  iv=b'\x00'\* AES.block\_size  key=b'YELLOW SUBMARINE'  with open('10.txt') as plaintext\_file:      plaintext=b64decode(plaintext\_file.read())  print(AES\_CBC\_decrypt(plaintext, key, iv).decode().rstrip())    T2-3  经过学习我们知道ECB加密对于同一条明文加密会得到同一个结果，即使添加了随机前后缀，其大体模样也不会差太多，而CBC加密结果随IV变化很大。因此根据这一点，我们选择连续三个块内容一致的明文，根据是否有密文块重复，判断其是否为ECB加密。  代码：  import os  import random  import Crypto.Cipher.AES as AES  #随机密钥  def random\_key():      return os.urandom(16)  #随机前后缀  def random\_padding():      return os.urandom(random.randint(5, 10))  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  #选择加密机  def encryption\_oracle(key, message):      mode = random.choice([AES.MODE\_ECB, AES.MODE\_CBC])      plaintext = random\_padding() + message + random\_padding()      plaintext = pad(plaintext, 16)      match mode:          case AES.MODE\_ECB:              return AES.new(key, mode).encrypt(plaintext), mode          case AES.MODE\_CBC:              iv = random\_key()              return AES.new(key, mode, iv).encrypt(plaintext), mode      assert False, "unreachable"  #检测预言机  def detect\_mode\_oracle(ciphertext):      blocks = [ciphertext[i : i + 16] for i in range(0, len(ciphertext), 16)]      if len(blocks) != len(set(blocks)):          return AES.MODE\_ECB      return AES.MODE\_CBC  key = random\_key()  msg = b"\x00" \* 16 \* 3  encrypted = [encryption\_oracle(key, msg) for \_ in range(1000)]  accr = sum(detect\_mode\_oracle(ciphertext) == mode for ciphertext, mode in encrypted)  print(f"{accr / len(encrypted):.2%}")  T2-4  题目下面已经给出了解题思路：   1. 通过不断改变前缀的长度，试探出块的大小（比较懒所以省略了）。 2. 用T2-3中的预言机来检测这是一个ECB加密（这步我永也省略了）。 3. 设计一个比块长度少一比特的前缀X，那么第一个块的最后一比特就是明文的第一比特，记录此时第一块的密文。 4. 改变前缀为X+Y,Y是1比特的字符，遍历Y，直到第一块密文与已知密文一致。 5. 那么，Y即为明文第一个字母。 6. 针对下一字母继续破解，直到破译全部明文。   代码：  import base64  import os  import Crypto.Cipher.AES as AES  import string    # 添加padding  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  #加密函数  def AES\_ECB\_encrpt(control\_text:bytes):      key = os.urandom(16)      plaintext = pad(control\_text + base64.b64decode("""  Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkg  aGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBq  dXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUg  YnkK"""), 16)      return AES.new(key, AES.MODE\_ECB).encrypt(plaintext)  #枚举得到明文长度  init\_unk\_strlen = len(AES\_ECB\_encrpt(b""))  unk\_strlen = init\_unk\_strlen  for i in range(16):      if len(AES\_ECB\_encrpt(b"A" \* i)) != init\_unk\_strlen:          unk\_strlen = init\_unk\_strlen - i          break  #DFS得到明文  plain\_space = string.printable.encode()  def dfs(known\_text: bytes):      while True:          partial = known\_text[-15:]          partial = b"\x00" \* (15 - len(partial)) + partial          current = []          for i in plain\_space:              oracle = partial + bytes([i]) + b"\x00" \* (15 - len(known\_text) % 16)              enc = AES\_ECB\_encrpt(oracle)              if enc[15] == enc[len(known\_text) // 16 \* 16 + 31]:                  current.append(i)          if len(current) == 1:              known\_text += bytes(current)              if len(known\_text) == unk\_strlen: # 达到预期长度，成功退出                  print(known\_text.decode())                  return True              continue          elif len(current) == 0:              return False          else:              for c in current:                  if dfs(known\_text + bytes([c])):                      return True  dfs(b'')  T2-5   1. 构造一个邮箱地址,这个邮箱的加密的第一个明文块为“email=xxxxxxxxxx”，第二个明文块为“admin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b”，这样能得到第二个明文块对应的密文块，即以admin结尾的密文块，我构造了如下的地址：   xxxxxxxxxxadmin\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b   1. 构造一个邮箱地址，要求是profile\_for()后最后一块的明文块只包含“user”，具体来说这道题需要一个13比特的邮箱地址，然后得到其密文，我构造了如下的地址：   master@xd.com   1. 取2中得到前两个密文块，再取1中得到的第二密文块，拼接起来。这样构造的密文解密就能将user改成admin。   代码：  from Crypto.Cipher import AES  from Crypto import Random  # 添加padding  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  # AES ECB模式加密  def AES\_ECB\_encrypt(plaintext: bytes, key: bytes) -> bytes:      cipher = AES.new(key, AES.MODE\_ECB)      return cipher.encrypt(pad(plaintext, AES.block\_size))  #AES ECB模式解密  def AES\_ECB\_decrypt(ciphertext: bytes, key: bytes) -> bytes:      cipher = AES.new(key, AES.MODE\_ECB)      return cipher.decrypt(ciphertext)  #profile  def profile\_for(email):      email=email.replace('&','').replace('=','')      return {'email':email, 'uid':10, 'role':'user'}  #字典转字符串  def kv\_encode(dict\_object):      encode\_text=''      for item in dict\_object.items():          encode\_text += item[0] + '=' + str(item[1]) + '&'      return encode\_text[:-1]  #字符串转字典  def kv\_decode(encode\_text):      dict\_object={}      attributes=encode\_text.split('&')      for item in attributes:          dict\_object[item.split('=')[0]]=item.split('=')[1]      return dict\_object  #ECB模式加解密  class ECBoracle:      def \_\_init\_\_(self):          self.key=Random.new().read(AES.key\_size[0])      def encrypt(self, email):          encoded=kv\_encode(profile\_for(email))          bytes\_to\_encrypted=encoded.encode()          return AES\_ECB\_encrypt(bytes\_to\_encrypted, self.key)      def decrypt(self, ciphertext):          return unpad(AES\_ECB\_decrypt(ciphertext, self.key))    #cut and paste攻击  def cut\_and\_paste\_attack(oracle):      prefix\_len = AES.block\_size - len('email=')      suffix\_len = AES.block\_size - len('admin')      email1 = 'x' \* prefix\_len + 'admin' + (chr(suffix\_len)\*suffix\_len)      encrypt1 = oracle.encrypt(email1)      email2 = "master@xd.com"      encrypt2 = oracle.encrypt(email2)      ciphertext = encrypt2[:32] + encrypt1[16:32]      return ciphertext  oracle=ECBoracle()  ciphertext=cut\_and\_paste\_attack(oracle)  decrypt = oracle.decrypt(ciphertext).decode()  plaintext = kv\_decode(decrypt)  print(plaintext)  T2-6  这里我们认为这个随机前缀产生后就不会改变，那么除了求明文长度外，还得求前缀导致的偏移量和需要的补齐长度，这里我们补齐长度确定为填满后再填两个块，只需修改深度优先搜索中的部分代码即可。  代码：  import base64  import os  import Crypto.Cipher.AES as AES  import string  import random  # 先生成随机长度，随机生成的前缀  prefix\_len = random.randint(0, 64)  prefix = os.urandom(prefix\_len)  # 添加padding  def pad(message:bytes, block\_size:int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)  #去除padding  def unpad(message:bytes) -> bytes:      padding = message[-1]      return message[:-padding]  #加密函数  def AES\_ECB\_encrpt(control\_text:bytes):      key = os.urandom(16)      plaintext = pad(prefix + control\_text + base64.b64decode("""  Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkg  aGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBq  dXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUg  YnkK"""), 16)      return AES.new(key, AES.MODE\_ECB).encrypt(plaintext)  #枚举得到明文长度,前缀导致的偏移量和需要的补齐长度  def get\_unklen():      init\_unk\_strlen = len(AES\_ECB\_encrpt(b""))      unk\_strlen = init\_unk\_strlen      for i in range(16):          if len(AES\_ECB\_encrpt(b"A" \* i)) != init\_unk\_strlen:              unk\_strlen = init\_unk\_strlen - i              break      leftlen = 0      while True:          leftlen += 1          enc = AES\_ECB\_encrpt(b"A" \* leftlen)          blocks = [enc[i : i + 16] for i in range(0, len(enc), 16)]          for i in range(len(blocks) - 1):              if blocks[i] == blocks[i + 1]:                  return unk\_strlen - i \* 16 + leftlen % 16, i \* 16, leftlen % 16    unk\_strlen, offset, leftpad = get\_unklen()  leftpad = b"\x00" \* leftpad  #DFS得到明文  plain\_space = string.printable.encode()      def dfs(known\_text):      while True:          partial = known\_text[-15:]          partial = b"\x00" \* (15 - len(partial)) + partial          current = []          for i in plain\_space:              oracle = leftpad + partial + bytes([i]) + b"\x00" \* (15 - len(known\_text) % 16)              enc = AES\_ECB\_encrpt(oracle)[offset:]              if enc[15] == enc[len(known\_text) // 16 \* 16 + 31]:                  current.append(i)          if len(current) == 1:              known\_text += bytes(current)              if len(known\_text) == unk\_strlen:                  print(known\_text.decode())                  return True              continue          elif len(current) == 0:              return False          else:              for c in current:                  if dfs(known\_text + bytes([c])):                      return True  dfs(b"")  T2-7  检查padding最后一个字符，得到字符对应数字，看前面是否有相应数量的字符。  代码：  def pad(message: bytes, block\_size: int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)    def unpad(message\_padded):      padding\_len = message\_padded[-1]      message, padding = message\_padded[:-padding\_len], message\_padded[-padding\_len:]      assert all(x == padding\_len for x in padding)      return message  print(unpad(b"ICE ICE BABY\x04\x04\x04\x04"))  print(unpad(b"ICE ICE BABY\x05\x05\x05\x05"))  T2-8  为了便于操作我就假设明文为":admin<true"，这样需要翻转的位数少一点，根据明文的结构，我们只需要将‘:’改为‘;’，‘<’改为‘=’，即可骗过检测程序。不过，还需要保证明文在同一个加密块中。由于固定前缀长30比特，我们添加2比特使其成为完整块，那么明文":admin<true"一定在第三个块中，然后对比特进行翻转即可。  代码：  import Crypto.Cipher.AES as AES  import os  key = os.urandom(16)  # 填充  def pad(message: bytes, block\_size: int) -> bytes:      padding = block\_size - len(message) % block\_size      return message + bytes([padding] \* padding)    # 去除填充  def unpad(message\_padded):      padding\_len = message\_padded[-1]      message, padding = message\_padded[:-padding\_len], message\_padded[-padding\_len:]      assert all(x == padding\_len for x in padding)      return message  # CBC模式加密  def AES\_CBC\_encrypt(userdata: bytes):      data = (          b"comment1=cooking MCs;userdata="          + userdata.replace(b";", b"%3B").replace(b"=", b"%3D")          + b";comment2= like a pound of bacon"      )      return AES.new(key, AES.MODE\_CBC, os.urandom(16)).encrypt(pad((b"\x00" \* 16) + data, 16))    # CBC模式解密  def AES\_CBC\_decrypt(data: bytes):      data = unpad(AES.new(key, AES.MODE\_CBC, os.urandom(16)).decrypt(data))[16:]      return {          (kv := item.split(b"=", maxsplit=1))[0].decode(): kv[1]          for item in data.split(b";")      }    # 检测函数  def is\_admin(data: bytes):      decrypted = AES\_CBC\_decrypt(data)      return decrypted.get("admin") == b"true"  padlen = 2  userdata = b"A" \* padlen + b":admin<true"  enc = bytearray(AES\_CBC\_encrypt(userdata))  enc[padlen + 30] ^= ord(":") ^ ord(";")  enc[padlen + 36] ^= ord("<") ^ ord("=")  if is\_admin(enc):      print("Success!")  else:      print("Fail!") |
| **总结**  这次实验主要涉及块密码相关知识，需要对AES,以及ECB、CBC加密模式比较熟悉。 |
| **参考文献**  <https://zhuanlan.zhihu.com/p/149989030>  <https://zhuanlan.zhihu.com/p/153069963>  https://koito.fun/posts/24-09-26-crypto-challenges-writeups |