山东大学 软件 学院

**Python高级程序设计(网络空间安全)**课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202200201095 | 姓名：杨伟康 | | 班级： 网安班 |
| 实验题目：实验3：实现对称加密编程 | | | |
| 实验学时：2 | | 实验日期： 2024.10.9 | |
|  | |  | |
| 实验目的：   * 利用cryptography和Crypto库中的Cipher类、algorithms、及modes 模块实现AES在ECB、CBC、CRT模式下的对称加密 * 利用padding实现数据填充； | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  **Pycharm** + Python | | | |
| 实验步骤与内容： 实验内容  1. 利用Cryptography库实现AES-ECB对称加密编程 2. 利用Crypto库实现AES-CBC对称加密编程 3. 利用Cryptography库实现AES-CTR对称加密编程 4. 练习3-12 手工CBC:编程应用AES的ECB模式实现CBC模式 5. 练习3-13 简单CTR模式：编程应用AES的ECB模式实现CTR模式 6. 分析代码3-6、3-7、3-8中的不同之处；   **实验源代码如下：**  *''' AES-ECB '''* from cryptography.hazmat.primitives import padding from cryptography.hazmat.primitives.ciphers import Cipher,algorithms, modes import os  def aes\_ecb\_encrypt(key, plaintext):  #创造一个aes—ecb的密码对象  cipher = Cipher(algorithms.AES(key), modes.ECB())  encryptor = cipher.encryptor()  #填充  padder = padding.PKCS7(algorithms.AES.block\_size).padder()  padded\_data = padder.update(plaintext) + padder.finalize()  #加密结果  ciphertext = encryptor.update(padded\_data) + encryptor.finalize()  return ciphertext  def aes\_ecb\_decrypt(key, ciphertext):  cipher =Cipher(algorithms.AES(key), modes.ECB())  decryptor =cipher.decryptor()  #解密  padded\_plaintext = decryptor.update(ciphertext) + decryptor.finalize()  #去除填充  unpadder = padding.PKCS7(algorithms.AES.block\_size).unpadder()  plaintext = unpadder.update(padded\_plaintext) + unpadder.finalize()  return plaintext  keys = os.urandom(16) plaintext = input("请输入要加密的内容:") btext = plaintext.encode('utf-8') print("AES-ECB加解密结果如下\n") ciphertext = aes\_ecb\_encrypt(keys, btext) print(f'Ciphertext: {ciphertext.hex()}')  decrypted\_text = aes\_ecb\_decrypt(keys, ciphertext) print(f'Decrypted\_text: {decrypted\_text.decode('utf-8')}')  ''' AES-CBC ''' from Crypto.Cipher import AES from Crypto.Util.Padding import pad, unpad   def aes\_cbc\_encrypt(key, plaintext):  iv = os.urandom(16)  cipher = AES.new(key, AES.MODE\_CBC, iv)  #填充成为块的倍数大小  padded\_plaintext = pad(plaintext, AES.block\_size)  #加密  ciphertext = cipher.encrypt(padded\_plaintext)  return iv+ciphertext  def aes\_cbc\_decrypt(key, ciphertext):  iv = ciphertext[:AES.block\_size]  ciphertext = ciphertext[AES.block\_size:]  cipher = AES.new(key, AES.MODE\_CBC, iv)  #解密  padded\_plaintext = cipher.decrypt(ciphertext)  #去除填充  plaintext = unpad(padded\_plaintext, AES.block\_size)  return plaintext print("AES-CBC加解密结果如下:") ciphertext = aes\_cbc\_encrypt(keys, btext) print(f'Ciphertext: {ciphertext.hex()}') decrypted\_text = aes\_cbc\_decrypt(keys, ciphertext) print(f'Decrypted\_text: {decrypted\_text.decode('utf-8')}')  ''' AES-CTR ''' from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes from cryptography.hazmat.backends import default\_backend import os  def aes\_ctr\_encrypt(key, plaintext):  #需要一个随机数  nonce = os.urandom(16)  #创造一个AES-CTR的对象  cipher = Cipher(algorithms.AES(key), modes.CTR(nonce))  encryptor = cipher.encryptor()  #加密  ciphertext = encryptor.update(plaintext) + encryptor.finalize()  return nonce+ciphertext  def aes\_ctr\_decrypt(key, ciphertext):  #将nonce拆下来  nonce = ciphertext[:AES.block\_size]  ciphertext = ciphertext[AES.block\_size:]  cipher = Cipher(algorithms.AES(key), modes.CTR(nonce))  decryptor = cipher.decryptor()  #解密  plaintext = decryptor.update(ciphertext) + decryptor.finalize()  return plaintext print("AES-CTR加解密结果如下:") ciphertext = aes\_ctr\_encrypt(keys, btext) print(f'Ciphertext: {ciphertext.hex()}') decrypted\_text = aes\_ctr\_decrypt(keys, ciphertext) print(f'Decrypted\_text: {decrypted\_text.decode('utf-8')}')  from Crypto.Cipher import AES from Crypto.Util.Padding import pad, unpad from Crypto.Random import get\_random\_bytes   def aes\_ecb\_simulate\_cbc\_encrypt(key, plaintext, iv):  cipher = AES.new(key, AES.MODE\_ECB)  block\_size = AES.block\_size  padded\_plaintext = pad(plaintext, block\_size)  ciphertext = bytearray()   # 第一个块与IV进行XOR  prev\_cipher\_block = iv  current\_plain\_block = padded\_plaintext[:block\_size]  xor\_block = bytearray([a ^ b for a, b in zip(current\_plain\_block, prev\_cipher\_block)])  encrypted\_block = cipher.encrypt(xor\_block)  ciphertext.extend(encrypted\_block)   # 后续的块与前一个密文块进行XOR  for i in range(block\_size, len(padded\_plaintext), block\_size):  current\_plain\_block = padded\_plaintext[i:i + block\_size]  prev\_cipher\_block = ciphertext[-block\_size:] # 取前一个密文块  xor\_block = bytearray([a ^ b for a, b in zip(current\_plain\_block, prev\_cipher\_block)])  encrypted\_block = cipher.encrypt(xor\_block)  ciphertext.extend(encrypted\_block)   return ciphertext   def aes\_ecb\_simulate\_cbc\_decrypt(key, ciphertext, iv):  cipher = AES.new(key, AES.MODE\_ECB)  block\_size = AES.block\_size  decrypted\_plaintext = bytearray()   # 第一个块解密后与IV进行XOR  encrypted\_block = ciphertext[:block\_size]  decrypted\_block = cipher.decrypt(encrypted\_block)  xor\_block = bytearray([a ^ b for a, b in zip(decrypted\_block, iv)])  decrypted\_plaintext.extend(xor\_block)   # 后续的块解密后与前一个密文块进行XOR  for i in range(block\_size, len(ciphertext), block\_size):  encrypted\_block = ciphertext[i:i + block\_size]  decrypted\_block = cipher.decrypt(encrypted\_block)  prev\_cipher\_block = ciphertext[i - block\_size:i] # 取前一个密文块（在解密过程中作为参考）  # 注意：这里应该使用解密前一个块后得到的明文块，但在模拟中我们没有保存这些明文块  # 因此，我们需要重新加密前一个已XOR的明文块来得到前一个密文块（这不是真正的CBC解密过程）  # 但为了这个示例，我们假设我们已经有了正确的prev\_cipher\_block（即实际的CBC模式中的前一个密文块）  xor\_block = bytearray([a ^ b for a, b in zip(decrypted\_block, prev\_cipher\_block)])  decrypted\_plaintext.extend(xor\_block)   # 注意：这里的模拟有一个问题，因为我们没有保存解密过程中的中间明文块  # 在真正的CBC解密中，你不需要重新加密来得到prev\_cipher\_block  # 这个示例只是为了说明如何在ECB的基础上模拟CBC的XOR操作  # 在实际应用中，你应该使用真正的CBC模式   # 去除填充  unpadded\_plaintext = unpad(decrypted\_plaintext, block\_size)  return unpadded\_plaintext   # 测试 key = get\_random\_bytes(16) iv = get\_random\_bytes(AES.block\_size) # 初始化向量 plaintext = btext  # 加密 print("ECB模拟CBC:") ciphertext = aes\_ecb\_simulate\_cbc\_encrypt(key, plaintext, iv) print("Ciphertext:", ciphertext.hex())  # 解密 decrypted\_plaintext = aes\_ecb\_simulate\_cbc\_decrypt(key, ciphertext, iv) print("Decrypted plaintext:", decrypted\_plaintext.decode('utf-8'))  from Crypto.Cipher import AES from Crypto.Util.Padding import pad from Crypto.Util.Padding import unpad from Crypto.Random import get\_random\_bytes   def aes\_ctr\_encrypt(key, plaintext):  # 确保密钥长度为16, 24, 或 32 字节  assert len(key) in [16, 24, 32], "Key must be 16, 24, or 32 bytes long"   # 初始化计数器，前8字节为随机nonce，后8字节为0（可以根据需要调整大小）  nonce = get\_random\_bytes(8)  initial\_counter = b'\x00' \* 8 # 8字节计数器，初始化为0  counter = int.from\_bytes(initial\_counter, 'big') # 将计数器初始化为整数形式以便递增   cipher = AES.new(key, AES.MODE\_ECB) # 使用ECB模式来“加密”计数器块（实际上只是用ECB模式来模拟CTR模式中的密钥流生成）   ciphertext = bytearray()  ciphertext.extend(nonce) # 将nonce添加到密文开头   # 分块加密，块大小与AES块大小一致  block\_size = AES.block\_size  padded\_plaintext = pad(plaintext, block\_size) # 使用PKCS7填充   for i in range(0, len(padded\_plaintext), block\_size):  # 更新计数器  counter += 1 # 递增计数器  counter\_bytes = counter.to\_bytes(8, 'big') # 将计数器转换回字节序列  counter\_block = nonce + counter\_bytes # 组合nonce和计数器作为新的计数器块   # 加密计数器块得到密钥流（在CTR模式中，这实际上是未加密的，因为只是用ECB模式来模拟）  keystream = cipher.encrypt(counter\_block) # 注意：这里实际上并没有使用ECB的加密特性，只是用它来生成与计数器块相同大小的密钥流   # XOR操作  plaintext\_block = padded\_plaintext[i:i + block\_size]  encrypted\_block = bytearray([a ^ b for a, b in zip(plaintext\_block, keystream)])   ciphertext.extend(encrypted\_block) # 将加密后的块添加到密文中   return ciphertext   def aes\_ctr\_decrypt(key, ciphertext):  # 确保密钥长度为16, 24, 或 32 字节  assert len(key) in [16, 24, 32], "Key must be 16, 24, or 32 bytes long"   # 从密文中提取nonce（前8字节）  nonce = ciphertext[:8]  counter\_start = b'\x00' \* 8 # 初始计数器值（与加密时一致）  counter = int.from\_bytes(counter\_start, 'big') # 将计数器初始化为整数形式   cipher = AES.new(key, AES.MODE\_ECB) # 使用ECB模式来“加密”计数器块（模拟CTR模式中的密钥流生成）   plaintext = bytearray()   # 分块解密，块大小与AES块大小一致  block\_size = AES.block\_size   for i in range(8, len(ciphertext), block\_size):  # 更新计数器  counter += 1 # 递增计数器  counter\_bytes = counter.to\_bytes(8, 'big') # 将计数器转换回字节序列  counter\_block = nonce + counter\_bytes # 组合nonce和计数器作为新的计数器块   # 加密计数器块得到密钥流（在CTR模式中，这实际上是未加密的，因为只是用ECB模式来模拟）  keystream = cipher.encrypt(counter\_block) # 注意：这里实际上并没有使用ECB的加密特性，只是用它来生成与计数器块相同大小的密钥流   # XOR操作来解密  encrypted\_block = ciphertext[i:i + block\_size]  decrypted\_block = bytearray([a ^ b for a, b in zip(encrypted\_block, keystream)])   plaintext.extend(decrypted\_block) # 将解密后的块添加到明文中   # 去除填充  unpadded\_plaintext = unpad(plaintext, block\_size)  return unpadded\_plaintext # 注意：解密函数需要类似地处理计数器递增，并且需要能够访问加密时使用的nonce和计数器值。 # 由于这个示例没有实现解密函数，因此没有提供解密部分的代码。 # 在实际应用中，你应该确保解密函数能够正确地重建加密时使用的计数器块。  # 测试 key = get\_random\_bytes(16) plaintext = btext  ciphertext = aes\_ctr\_encrypt(key, plaintext)  bytes\_ciphertext = bytes(ciphertext)   # 如果你想要一个没有引号或前缀的字符串表示，你可以将其编码为十六进制 print("ECB模拟CTR:") hex\_ciphertext = bytes\_ciphertext.hex() print("Ciphertext:",hex\_ciphertext) ciphertext\_bytes = bytes.fromhex(hex\_ciphertext)  # 现在可以正确地解密了 decrypted\_plaintext = aes\_ctr\_decrypt(key, ciphertext\_bytes) print("Decrypted plaintext:",decrypted\_plaintext.decode('utf-8')) # 输出解密后的明文  演示结果截图  **三个代码的不同之处在于：**  代码清单 3-6 和 3-7 和3-8对比  利用填充机制（ Padding中的padder和unpadder ）在明文分组长度不足128bit时的两种处理方式。  代码清单 3-6因为padder和unpadder应用不恰当，结果和预期的不一样；  代码清单3-7对每一个明文单独padder和unpadder后进行单独加密解密，得到结果和预期的一样；代码中存在的问题是每次加密解密都生成新的加密解密对象，不同的加解密对象每次使用相同的IV和key都一样。会导致相同的明文输入得到相同的密文输出；  代码清单3-8 将密钥、IV和加解密对象的生成定义在构造函数中。只要生成新的加密管理器，就会有新的密钥、IV和加解密对象。 | | | |
| 结论分析与体会： 通过本次实验，我深入了解了AES对称加密算法在不同模式（ECB、CBC、CTR）下的实现方法，并掌握了使用cryptography和Crypto库进行加密和解密的基本流程。以下是我对本次实验的一些心得体会：  理解AES算法及其模式：  ECB模式：简单但不安全，因为相同的明文块会产生相同的密文块，容易受到模式分析攻击。  CBC模式：通过引入初始化向量（IV）和链式加密提高了安全性，但需要注意IV的随机性和保密性。  CTR模式：将AES块加密器转换为流密码，通过将计数器块与明文块进行XOR操作，适合处理任意长度的数据。  库的使用：  cryptography库：提供了现代化的API，易于使用且安全性高。通过Cipher、algorithms和modes模块，可以灵活地配置加密模式和算法。  Crypto库：提供了丰富的加密功能，包括填充和随机字节生成等实用工具。使用AES.new()方法可以方便地创建AES加密对象。  数据填充：  在块加密中，填充（Padding）是必要的，因为AES算法要求输入数据必须是块大小的整数倍。  Crypto.Util.Padding模块提供了方便的填充和去填充函数，确保了数据在加密和解密过程中的完整性。  代码实现：  在实现AES-ECB、AES-CBC和AES-CTR加密时，需要注意密钥、IV（如果需要）和加密模式的正确配置。  特别是AES-CTR模式中，nonce（或称为计数器块）的随机性和唯一性对安全性至关重要。  在模拟CBC和CTR模式时，通过ECB模式进行XOR操作，深刻理解了这些模式的工作原理。  代码优化与问题：  在代码实现过程中，发现了一些问题，如padder和unpadder应用不当导致的解密错误。这提醒我在处理数据时，必须仔细考虑数据的完整性和格式。  不同的加解密对象使用相同的IV和key时，会导致相同的明文输入得到相同的密文输出，这在某些情况下可能是一个安全隐患。因此，在实际应用中，需要谨慎管理IV和key的生成和使用。  在模拟CTR模式时，我意识到需要正确管理计数器块和nonce，以确保解密过程的正确性。  代码结构：  通过对比不同的代码实现，我意识到良好的代码结构对于代码的可读性和可维护性至关重要。将密钥、IV和加解密对象的生成封装在构造函数中，可以提高代码的模块化和复用性。 | | | |

注：命名”学号姓名.docx”，例如：“2006999孔明.docx”