山东大学 软件 学院

Python密码学编程课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202100150209 | 姓名： 杨佳庆 | | 班级： 21级网安班 |
| 实验题目：实验3 | | | |
| 实验学时：2 | | 实验日期： 2023.10.10 | |
| 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 | | | |
| 实验步骤与内容：  定义类AESCrypto，在\_\_init\_\_初始化方法体中定义两个参数：aes\_iv，aes\_key  KEY = os.urandom(32) IV = os.urandom(16)  Key,IV可以通过os内置的随机数方法获取，初始化类时传入即可  class AESCrypto(object):  *"""AESCrypto."""* def \_\_init\_\_(self, aes\_key, aes\_iv):  if not isinstance(aes\_key, bytes):  aes\_key = aes\_key.encode()   if not isinstance(aes\_iv, bytes):  aes\_iv = aes\_iv.encode()   self.aes\_key = aes\_key  self.aes\_iv = aes\_iv  类中定义两个静态方法用来填充和删除填充数据  class AESCrypto(object):  @staticmethod def pkcs7\_padding(data):  *"""pkcs7\_padding."""* padder = padding.PKCS7(algorithms.AES.block\_size).padder()   padded\_data = padder.update(data) + padder.finalize()   return padded\_data  @staticmethod def pkcs7\_unpadding(padded\_data):  *"""pkcs7\_unpadding."""* unpadder = padding.PKCS7(algorithms.AES.block\_size).unpadder()  data = unpadder.update(padded\_data)   try:  uppadded\_data = data + unpadder.finalize()  except ValueError:  raise Exception('无效的加密信息!')  else:  return uppadded\_data  定义encrypto和decrypto函数来包装，实现用户友好型程序，用户可以通过传入mode参数来选择使用哪一种工作模式加密，并检测数据是否已经编码过，如果没有就对数据执行encode()操作。  def encrypt(self, data, mode='cbc'):  *"""encrypt."""* func\_name = '{}\_encrypt'.format(mode)  func = getattr(self, func\_name)  if not isinstance(data, bytes):  data = data.encode()   return func(data)  def decrypt(self, data, mode='cbc'):  *"""decrypt."""* func\_name = '{}\_decrypt'.format(mode)  func = getattr(self, func\_name)   if not isinstance(data, bytes):  data = data.encode()   return func(data)  用户只需要先创建一个类的实例，传入key和iv两个参数  KEY = os.urandom(32) IV = os.urandom(16) crypto = AESCrypto(KEY, IV)  对想要加密解密的数据直接调用encrypt()和decrypt()函数即可  message = "abcdefghjklmnopqrstuvwxyz1234567890"  data1 = crypto.encrypt(message,'ecb') print(data1) print(crypto.decrypt(data1, 'ecb'))  结果如下：     1. 利用Cryptography库实现AES-ECB对称加密编程   def ecb\_encrypt(self, data):  *"""ECB encrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.ECB(),  backend=default\_backend())   return cipher.encryptor().update(data)  def ecb\_decrypt(self, data):  *"""ECB decrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.ECB(),  backend=default\_backend())   return cipher.decryptor().update(data)   1. 利用Crypto库实现AES-CBC对称加密编程   def cbc\_encrypt(self, data):  *"""cbc\_encrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.CBC(self.aes\_iv),  backend=default\_backend())   return cipher.encryptor().update(self.pkcs7\_padding(data))  def cbc\_decrypt(self, data):  *"""cbc\_decrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.CBC(self.aes\_iv),  backend=default\_backend())   uppaded\_data = self.pkcs7\_unpadding(cipher.decryptor().update(data))  return uppaded\_data.decode()   1. 利用Cryptography库实现AES-CTR对称加密编程   def ctr\_encrypt(self, data):  *"""ctr\_encrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.CTR(self.aes\_iv),  backend=default\_backend())   return cipher.encryptor().update(self.pkcs7\_padding(data))  def ctr\_decrypt(self, data):  *"""ctr\_decrypt."""* cipher = Cipher(algorithms.AES(self.aes\_key),  modes.CTR(self.aes\_iv),  backend=default\_backend())   uppaded\_data = self.pkcs7\_unpadding(cipher.decryptor().update(data))  return uppaded\_data.decode()  结果如下图：     1. 练习3-12 手工CBC:编程应用AES的ECB模式实现CBC模式   def ecbTocbc(self, data):  if not isinstance(data, bytes):  data = data.encode()  iv = self.aes\_iv  block\_size = self.block\_size  cipherText = b''  cipher = Cipher(algorithms.AES(self.aes\_key),  modes.ECB(),  backend=default\_backend())   for i in range(0, len(data), block\_size):  block = data[i:i + block\_size]   # 异或操作，将IV与明文块结合  xor\_result = bytes(x ^ y for x, y in zip(block, iv))   # 填充并使用ECB加密  padded\_data = self.pkcs7\_padding(xor\_result)  encryptor = cipher.encryptor()  encrypted\_block = encryptor.update(padded\_data)   # 存储加密后的块，并将其作为下一个块的IV  cipherText += encrypted\_block  iv = encrypted\_block  return cipherText   1. 练习3-13 简单CTR模式：编程应用AES的ECB模式实现CTR模式   def ecbToctr(self, data):  if not isinstance(data, bytes):  data = data.encode()  nonce = self.aes\_iv  block\_size = self.block\_size  # 初始化计数器  counter = 0  cipherText = b''   cipher = Cipher(algorithms.AES(self.aes\_key),  modes.ECB(),  backend=default\_backend())   for i in range(0, len(data), block\_size):  block = data[i:i + block\_size]   # 组合Nonce和计数器，然后加密  counter\_bytes = counter.to\_bytes(length=16, byteorder='big', signed=False)  counter\_nonce = nonce + counter\_bytes  encryptor = cipher.encryptor()  encrypted\_counter = encryptor.update(counter\_nonce)   # 异或操作，将加密后的计数器块与明文块结合  xor\_result = bytes(x ^ y for x, y in zip(block, encrypted\_counter))   # 存储加密后的块  cipherText += xor\_result  # 更新计数器  counter += 1  return cipherText   1. 分析代码3-6、3-7、3-8中的不同之处；   相同之处：   1. 使用Cryptography库提供的primitives.ciphers模块中的Cipher和algorithms以及modes模块实现AES加密和解密。 2. 都使用了default\_backend()函数指定默认的加密后端。 3. 使用了padding模块提供的PKCS7填充方式对数据进行填充和去填充。   不同之处：   1. Code3.6在全局定义了加密和解密的对象，而Code3.7和Code3.8使用了一个名为EncryptionManager的类来封装加密和解密的方法。 2. Code3.6 在循环明文数据时，没有每次填充，而是在所有数据都添加到一个padder结构内，最后进行一次填充并销毁；并每次将padder内容添加到密文列表中，因此通过密文恢复明文时，由于padder并不输出不足需求的数据，所以解密的结果会与期望不符，同时会有冗余填充的问题，使得输出最终还多出一个空白数据。 3. Code3.7解决了解密的结果不符预期的问题，通过在循环明文数据加密时，每个循环内都创建一个padder对象，对每次的数据进行填充并销毁，下次重新创建，使得数据不会混合。Code3.8的解决方法与Code3.7基本一致，但是增加了一个判断来解决冗余填充的问题：   if paddedmsg == b'':  paddedmsg = self.padder.finalize() else:  paddedmsg += self.padder.finalize()  冗余填充： AES-128的数据块长度是 16bytes，使用PKCS7进行填充时，填充的长度范围是 1 ~ 16。注意，当待加密数据长度为 16 的整数倍时，填充的长度反而是最大的，要填充 16 字节，为什么呢？因为 "PKCS7" 拆包时会按协议取最后一个字节所表征的数值长度作为数据填充长度，如果因真实数据长度恰好为 16 的整数倍而不进行填充，则拆包时会导致真实数据丢失。   1. Code3.7和Code3.8的差别主要是对类方法的处理，Code3.8在初始化方法体中定义了几个全局变量和类的属性变量：   def \_\_init\_\_(self):  key = os.urandom(32)  iv = os.urandom(16)  aesContext = Cipher(algorithms.AES(key),  modes.CBC(iv),  backend=default\_backend())  self.encryptor = aesContext.encryptor()  self.decryptor = aesContext.decryptor()  self.padder = padding.PKCS7(128).padder()  self.unpadder = padding.PKCS7(128).unpadder()  在加密和解密方法中直接调用已经定义好的属性，这样做的好处是方便维护和修改，同时降低了代码的重用率。**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。**  综上所述，Code3.6、Code3.7和Code3.8都实现了AES加密和解密的功能，但它们在实现方式上略有不同。代码一和代码三更为直接，并且代码三使用了一个类来封装加密和解密的方法，使得代码更加结构化和可复用。代码二相比于代码一和代码三更加简洁，没有使用类和实例化对象，而是直接定义了一个加密管理器并调用其方法进行加密和解密。  结果如下图：  补充结果截屏 | | | |
| 结论分析与体会：  使用Cryptography库提供的primitives.ciphers模块中的Cipher和algorithms以及modes模块实现AES加密和解密。ECB模式将待加密的数据分割为多个固定大小的块，然后对每个块进行独立的加密。在编程中，我们需要将数据进行填充，以满足算法要求的块大小。  CBC模式对每个数据块进行加密时，需要使用前一个块的密文与当前块的明文进行异或操作，以增加密码算法的强度。CTR模式将加密器的计数器作为一个伪随机数生成器，并将生成的序列与明文进行异或操作，从而实现加密。 通过本次实验，我们深入了解了对称加密算法中的三种常见模式（ECB、CBC、CTR），并通过编程实现了它们。这些模式在加密数据时提供了不同的保护措施，并在不同的应用场景中有不同的优势。我们学会了如何使用cryptography和Crypto库提供的功能，使用AES算法进行加密，并了解了填充技术在数据保护中的重要性。  此外，在分析代码差异的过程中，我们意识到了不同加密模式的实现方式和性能之间的差异。这对于我们在实际应用中选择适当的加密模式至关重要。  总的来说，本次实验让我们对对称加密算法有了更深入的理解，同时提高了我们的编程能力和对加密技术的应用认识。在实际应用中，我们可以根据需求选择合适的加密模式和填充技术，从而提高数据的安全性和完整性。 | | | |