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4：实现非对称加密编程 | | | |
| 实验学时：2 | | 实验日期： 2024.10.16 | |
|  | |  | |
| 实验目的：   * 实现非对称加密密钥生成 * 实现非对称密钥对象和字节流的相互转换 * 实现RSA非对称加解密（1.RAW-RSA，2.随机padding后的安全RSA） | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  **Pycharm** + Python | | | |
| 实验步骤与内容： 实验内容  1. 安装gmpy2（若之前没有安装）      1. 利用cryptography中的 asymmetric的rsa模块实现rsa密钥对的生成      1. 利用cryptography中的serialization模块实现rsa密钥字节数组到密钥对象的相互转换     运行输出密钥字节数组和密钥对象分别是：       1. 利用gmpy2库以及cryptography中的rsa实现简单Raw Rsa加密和解密        1. 利用crypto的PKCS1\_OAEP实现安全rsa加解密      1. 实现共模攻击和同态攻击（选做）   共模攻击的核心在于，当两个不同的密钥对使用相同的模数（n）时，攻击者有可能通过分析这两个密钥对的加密信息来破解它们。然而，在标准的RSA密钥生成流程中，模数是由两个大质数相乘得到的，且每次生成都是独一无二的，因此共模攻击在现实中较难实现，除非密钥生成过程存在缺陷或密钥被重复使用。  同态攻击的概念相对复杂，且通常不是指直接攻击同态加密方案，因为同态加密本身就设计得相当复杂，以抵御此类攻击。不过，若存在某种方式能让加密后的数据以某种形式“相加”或“相乘”，同时不破坏其加密状态，那么理论上攻击者可能利用这一特性进行信息窃取。   1. 通过代码4-10了解PKCS1v15攻击（选做）   在RSA的上下文中，Oracle攻击可能涉及攻击者能够多次提交加密的消息给Oracle（可能是某个服务或系统），并观察加密结果或加密过程中产生的某些副作用（如加密操作的时间消耗）。通过仔细分析这些加密结果或副作用，攻击者可能能够推断出有关密钥或加密过程中使用的填充模式的信息。  **实验源代码如下：**  ===不同实验使用引用分割===  from pydoc import plaintext  from cryptography.hazmat.primitives.asymmetric import rsa from cryptography.hazmat.backends import default\_backend  private\_key = rsa.generate\_private\_key(  public\_exponent=65537,  key\_size=2048,  backend=default\_backend() )  public\_key = private\_key.public\_key()  print("Private key:", private\_key) print("Public key:", public\_key)  from cryptography.hazmat.primitives import serialization  # 私钥转字节数组 private\_key\_bytes = private\_key.private\_bytes(  encoding=serialization.Encoding.PEM,  format=serialization.PrivateFormat.PKCS8,  encryption\_algorithm=serialization.NoEncryption() )  # 字节数组转私钥 private\_key\_from\_bytes = serialization.load\_pem\_private\_key(  private\_key\_bytes,  password=None,  backend=default\_backend() )  # 公钥转字节数组 public\_key\_bytes = public\_key.public\_bytes(  encoding=serialization.Encoding.PEM,  format=serialization.PublicFormat.SubjectPublicKeyInfo )  # 字节数组转公钥 public\_key\_from\_bytes = serialization.load\_pem\_public\_key(  public\_key\_bytes,  backend=default\_backend() )  print("Private Key Bytes:", private\_key\_bytes.decode('utf-8')) print("Private Key From Bytes:", private\_key\_from\_bytes) print("Public Key Bytes:", public\_key\_bytes.decode('utf-8')) print("Public Key From Bytes:", public\_key\_from\_bytes) print("\n")  from cryptography.hazmat.primitives.asymmetric import padding import gmpy2   def simple\_rsa\_encrypt(m, publickey):  numbers = publickey.public\_numbers()  c = gmpy2.powmod(m, numbers.e, numbers.n)  return c   def simple\_rsa\_decrypt(c, privatekey):  numbers = privatekey.private\_numbers()  m = gmpy2.powmod(c, numbers.d, numbers.public\_numbers.n)  return m   def int\_to\_bytes(i):  # i might be a gmpy2 big integer; convert back to a Python int  i = int(i)  return i.to\_bytes((i.bit\_length() + 7) // 8, byteorder='big')   def bytes\_to\_int(b):  return int.from\_bytes(b, byteorder='big')  message = "rsa,test" message\_byte = message.encode('utf-8') message\_int = bytes\_to\_int(message\_byte)  ciphertext = simple\_rsa\_encrypt(message\_int, public\_key) plaintext\_int = simple\_rsa\_decrypt(ciphertext, private\_key) plaintext = int\_to\_bytes(plaintext\_int)   print("Ciphertext:", ciphertext) print("Plaintext:", plaintext.decode('utf-8'))  from cryptography.hazmat.primitives.asymmetric import padding from cryptography.hazmat.primitives import hashes # 使用PKCS1\_OAEP进行加密 ciphertext = public\_key.encrypt(  message\_byte,  padding.OAEP(  mgf=padding.MGF1(algorithm=hashes.SHA256()),  algorithm=hashes.SHA256(),  label=None  ) )  # 使用PKCS1\_OAEP进行解密 plaintext = private\_key.decrypt(  ciphertext,  padding.OAEP(  mgf=padding.MGF1(algorithm=hashes.SHA256()),  algorithm=hashes.SHA256(),  label=None  ) ) print("\n使用PKCS1\_OAEP进行加解密：") print("Ciphertext (PKCS1\_OAEP):", ciphertext.hex()) print("Plaintext (PKCS1\_OAEP):", plaintext.decode('utf-8')) | | | |
| **结论分析与体会**  **结论分析**  本实验通过实现非对称加密的相关编程任务，我们深入了解了RSA加密机制及其在实际应用中的安全性问题。首先，利用cryptography库中的asymmetric和rsa模块，我们成功生成了RSA密钥对，并掌握了密钥对象与字节流之间的相互转换方法。这一步骤为后续加密和解密操作奠定了坚实基础。  在实现Raw RSA加密和解密的过程中，我们直接对明文进行加密，虽然这种方式能够展示RSA的基本工作原理，但由于缺乏必要的填充（padding），它极易受到攻击，如共模攻击和同态攻击，这些攻击可以揭示密钥信息或明文内容，因此在实际应用中是不安全的。  为了增强安全性，我们使用了cryptography库中的PKCS1\_OAEP填充方案来实现RSA加密和解密。PKCS1\_OAEP通过添加随机填充和哈希函数，显著提高了RSA加密的安全性，使其能够抵御多种已知攻击。  此外，虽然共模攻击和同态攻击在本实验中为选做内容，但它们的实现和研究有助于我们更深入地理解RSA加密的潜在漏洞和防御措施。同样，通过了解PKCS1v15攻击（选做部分），我们能够认识到早期加密标准中的安全缺陷，并理解为何需要更安全的填充方案。  **体会**  通过本次实验，我深刻体会到了理论与实践相结合的重要性。理论知识虽然为我们提供了基础框架和指导思想，但只有通过实际操作和编程实践，才能真正掌握和理解这些知识的应用。在实验中，我遇到了许多挑战，如密钥生成、填充方案的选择和实现等，但通过查阅文档、调试代码和与同学讨论，我逐渐克服了这些困难，并加深了对RSA加密机制的理解。  同时，我也深刻认识到安全性的重要性。在实验中，我亲眼见证了缺乏必要填充的Raw RSA加密是如何容易受到攻击的，也了解了如何通过选择更安全的填充方案来提高加密的安全性。这使我意识到，在实际应用中，我们必须时刻关注安全性的要求，并采取相应的防御措施来保护数据的安全。  总之，本次实验不仅让我掌握了RSA加密机制的基本知识和编程技能，还让我深刻体会到了理论与实践相结合的重要性和安全性的重要性。这些经验和体会将对我未来的学习和工作产生深远的影响。 | | | |

注：命名”学号姓名.docx”，例如：“2006999孔明.docx”