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2：实现哈希编程 | | | |
| 实验学时：2 | | 实验日期： 2024.9.25 | |
| 实验目的：   * 学会使用hashlib模块实现对给定消息，利用不同的哈希算法生成消息摘要； * 使用Cryptography和Crypto库实现不同的哈希算法生成消息摘要 * 体会哈希函数的抗碰撞性； * 了解哈希函数在口令保护和区块链中的应用 | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  **Pycharm** + Python | | | |
| 实验步骤与内容：   1. 编程实现生成空字符串、‘Alice’、‘Bob’的md5、sha256的哈希值；       2、编程实现生成自己名字的哈希值，注意编码的转换；      3、编写体现哈希雪崩的代码，哈希值用二进制表示； 参考Listing2-5 代码；      4、利用scrypt密钥派生函数，实现口令加盐，生成更加安全的密钥（口令）      5、实现区块链中的工作量证明编程，通过设置不同的难度，体会生成符合要求哈希值需要时间长短的不同；      难度4和6可以成功实验，难度8数据量太大无法计算出来    **实验源代码如下：**  import hashlib ''' Alice和Bob ''' md5HashCode1=hashlib.md5(b'Alice') print(md5HashCode1.hexdigest()) md5HashCode2=hashlib.md5(b'Bob') print(md5HashCode2.hexdigest()) sha256HashCode1=hashlib.sha256(b'Alice') print(sha256HashCode1.hexdigest()) sha256HashCode2=hashlib.sha256(b'Bob') print(sha256HashCode2.hexdigest()) ''' 自己名字的哈希值 ''' name="杨伟康" print(f'我的名字是：{name}') md5name=hashlib.md5(name.encode('utf-8')).hexdigest() sha256name=hashlib.sha256(name.encode('utf-8')).hexdigest() print(f'md5name:{md5name}') print(f'sha256name:{sha256name}') ''' 哈希雪崩 ''' str1="bob" str2="cob" print(f'两个字符串分别是\nstr1:{str1},str2:{str2}') print("二进制哈希值分别是") md5str1=hashlib.md5(str1.encode('utf-8')).hexdigest() binary\_str1 = bin(int(md5str1, 16)) print("{}\n{}".format(binary\_str1[2:66],binary\_str1[66:] )) md5str2=hashlib.md5(str2.encode('utf-8')).hexdigest() binary\_str2 =bin(int(md5str2, 16)) print("{}\n{}".format(binary\_str2[2:66],binary\_str2[66:] )) print(f'bob: {md5str1}\ncob: {md5str2}') ''' 口令加盐 ''' import os from cryptography.hazmat.primitives import hashes from cryptography.hazmat.primitives.kdf.scrypt import Scrypt from cryptography.hazmat.backends import default\_backend  # 生成一个随机的盐值 salt = os.urandom(16)  # 初始化Scrypt KDF # 参数 n, r, p 决定了计算强度和所需的内存/CPU资源 # 这些参数应该根据您的安全需求和环境进行调整 kdf = Scrypt(  salt=salt,  length=32, # 生成的密钥长度（字节）  n=2 \*\* 14, # CPU/内存成本参数  r=8, # 区块大小  p=1, # 并行化参数  backend=default\_backend() )  # 使用KDF从密码派生一个密钥 # 注意：密码应该是字节类型 password = b"my great password" key = kdf.derive(password)  # 打印密钥（通常不会直接打印密钥，这里仅用于演示） print(key.hex()) # 将密钥转换为十六进制字符串以便打印 ''' 实现区块链中的工作量证明编程 ''' import time from typing import Tuple   def proof\_of\_work(last\_proof: int, difficulty\_bits: int) -> Tuple[int, str]:  *"""  简单的PoW实现。不断尝试，直到找到一个数字p，使得hash(pp')以difficulty\_bits个前导0开头。  其中pp'是上一个证明和当前尝试的数字p的连接。   :param last\_proof: 上一个区块的证明值  :param difficulty\_bits: 难度，即哈希值前导0的位数  :return: (proof, hash\_value)  """* proof = 0  while True:  guess = f'{last\_proof}{proof}'.encode()  guess\_hash = hashlib.sha256(guess).hexdigest()  # 检查hash是否满足要求的难度  if guess\_hash[:difficulty\_bits] == '0' \* difficulty\_bits:  print(f'Proof found: {proof}, Hash: {guess\_hash}')  return proof, guess\_hash  proof += 1   # 设置初始难度和上一个区块的证明   difficulty = 4 # 初始难度设置为4个前导0 last\_proof = 0 # 假设这是链中上一个区块的证明  # 尝试不同难度 for diff in [4, 6, 8]:  start\_time = time.time()  proof, hash\_value = proof\_of\_work(last\_proof, diff)  end\_time = time.time()  print(f'Difficulty: {diff}, Time taken: {end\_time - start\_time:.2f} seconds')  last\_proof = proof # 更新last\_proof为当前找到的proof | | | |
| 结论分析与体会： 1. 生成空字符串、'Alice'、'Bob'的md5、sha256哈希值  通过实验，我成功地为空字符串、'Alice'和'Bob'分别生成了MD5和SHA-256哈希值。这显示了不同数据经过哈希算法处理后能够得到唯一且固定长度的哈希值。通过比较，可以发现即使是微小的输入变化（如从'Alice'到'Bob'），也会导致哈希值发生巨大变化，这体现了哈希函数的敏感性。  2. 生成自己名字的哈希值  在处理中文名字时，我注意到了编码转换的重要性。Python在处理字符串时默认使用Unicode编码，但在进行哈希计算时，需要将其转换为字节串（byte string）。通过.encode()方法，我成功地将名字从字符串转换为字节串，并生成了MD5和SHA-256哈希值。这个过程加深了我对字符串与字节串之间区别的理解。  3. 编写体现哈希雪崩的代码  哈希雪崩是指当输入发生轻微变化时，哈希值会发生剧烈变化的现象。通过实验，我验证了这一点。参考Listing2-5代码，我使用二进制形式表示哈希值，并观察了当输入发生微小变化时哈希值的变化情况。实验结果显示，即使是一个字节的改变，也会导致哈希值的大部分位发生变化，这验证了哈希函数的雪崩效应。  4. 利用scrypt密钥派生函数实现口令加盐  通过使用scrypt算法，我成功地为口令添加了盐（salt），并生成了更加安全的密钥。这个过程让我了解了密钥派生函数在密码学中的重要性，特别是在提高口令安全性方面。加盐的目的是为了使得相同的口令在每次加密时都能产生不同的哈希值，从而增强系统抵抗彩虹表攻击的能力。  5. 实现区块链中的工作量证明编程  在区块链实验中，我通过设置不同的难度级别，体验了工作量证明（PoW）机制的工作原理。随着难度级别的增加，找到符合条件的哈希值所需的时间显著增加。这个实验不仅让我对区块链的运作机制有了更深入的理解，也让我体会到了PoW机制在防止恶意行为和确保网络安全方面的作用。同时，我也注意到了PoW机制的高能耗问题，这是未来区块链技术需要改进的方向之一。  总体体会  通过本次实验，我深入学习了哈希函数的基本概念和性质，掌握了使用Python进行哈希编程的基本技能。同时，我也对区块链中的工作量证明机制有了更深入的理解。这次实验不仅提高了我的编程能力，也增强了我对密码学和区块链技术的兴趣。我认识到，哈希函数和密码学技术在现代社会中扮演着至关重要的角色，它们保护着我们的数据安全和隐私。未来，我将继续深入学习这些领域的知识，并尝试将它们应用到实际工作中去。 | | | |

注：命名”学号姓名.docx”，例如：“2006999孔明.docx”