Google Password Checkup协议实现说明

1. 实现概述

本项目实现了Google Password Checkup协议，基于论文《Privacy-Preserving Password Checking with Secret Sharing》(https://eprint.iacr.org/2019/723.pdf) 中section 3.1和Figure 2描述的协议。

实现采用模块化设计，将不同功能分离到不同的模块中，提高代码的可维护性和可扩展性。

2. 模块说明

2.1 密码学工具模块 (crypto_utils.py)

该模块提供了协议所需的基本密码学功能：

- **大素数生成**: 使用`generate_large_prime`函数生成指定比特长度的大素数，作为模运算的基础
- **哈希函数**: 实现了将任意输入映射到有限域$\mathbb{Z}_p$的哈希函数
- **秘密分割与合并**: 实现了将秘密值分割为两个份额以及将份额合并还原秘密的功能
- **随机数生成**: 提供了在有限域内生成随机数的功能

哈希函数实现说明：def hash_function(data, prime):
# 使用SHA-256哈希算法
sha256_hash = hashlib.sha256(data.encode('utf-8')).digest()
# 转换为整数并取模，映射到[0, prime-1]范围
return int.from_bytes(sha256_hash, byteorder='big') % prime

2.2 客户端模块 (client.py)

客户端模块实现了用户侧的功能：

- 初始化客户端，保存大素数
- 处理用户密码，计算哈希值并分割为两个份额
- 根据辅助服务器返回的r2值计算r1

核心代码：def process_password(self, password):
# 计算密码的哈希值
password_hash = hash_function(password, self.prime)
# 将哈希值分割为两个份额
t1, t2 = split_secret(password_hash, self.prime)
return t1, t2

2.3 服务器模块 (server.py)

服务器模块实现了主服务器和辅助服务器的功能：

- **主服务器(Server)**:

- 维护泄露密码集合
- 为每个泄露密码计算哈希值并分割为秘密份额
- 生成挑战值并验证客户端响应

- **辅助服务器(HelperServer)**:
  - 存储主服务器发送的秘密份额
  - 处理客户端查询，生成r2和响应值

主服务器验证响应的核心代码：def verify_response(self, response, challenge, r1):
k = challenge
for password in self.leaked_passwords:
s1, _ = self.shares[password]
if (k * s1 + r1) % self.prime == response:
return True
return False

2.4 协议核心模块 (protocol.py)

协议核心模块协调各参与方之间的交互，实现完整的协议流程：
def run_protocol(leaked_passwords, client_password, prime=None):
# 初始化参与方
# 执行协议步骤
# 返回检查结果

3. 数据流程

1. **初始化阶段**:
   - 服务器接收并存储泄露密码列表
   - 服务器为每个泄露密码计算哈希值并分割为两个份额
   - 服务器将第二个份额发送给辅助服务器

2. **查询阶段**:

```
客户端 -> 密码处理 -> 生成(t1, t2)
                    |
服务器 -> 生成挑战值k <-|
                        |
客户端 -> 发送t2 -> 辅助服务器 -> 生成(r2, response) -> 返回给客户端
                                                        |
客户端 -> 计算r1 <-|                                    |
    |              |                                     |
    -> 发送response -> 服务器 -> 验证(k*s1 + r1) == response -> 返回结果
```

4. 安全性考虑

1. **素数选择**: 实现中使用了2048位的大素数，提供了足够的安全性
2. **哈希函数**: 使用SHA-256作为基础哈希函数，具有抗碰撞性

3. **随机数生成**: 使用密码学安全的随机数生成器
4. **隐私保护**: 协议设计确保任何参与方都无法单独获取完整的密码哈希信息

5. 可能的改进

1. 增加缓存机制，提高频繁查询的效率
2. 实现批量查询功能，支持一次检查多个密码
3. 增加通信加密层，保护传输中的数据安全
4. 优化大素数运算性能，减少计算开销