山东大学
网络空间安全学院

创新创业实践
Project6

学生姓名：滕怀源

学号：202200460054)

学院：网络空间安全学院

专业班级： 2022 级网安 2 班

完成时间：2025 年 7 月 30 日

目录

1 DDH-based Private Intersection-Sum 协议原理

该协议用于在两个参与方之间进行 **私密集合求交总和(Private Set Intersection-Sum, PSI-Sum)** 的计算。目标是在不泄露除交集以外的信息的前提下，让一方（P_2）得知交集中数据的加总值。该协议基于 DDH （Decisional Diffie-Hellman）假设和同态加密构建，协议流程如下：

输入

- $\mathcal{G}$ 是一个阶为素数的乘法群，$\mathcal{U}$ 是标识符空间；
- 哈希函数 $H : \mathcal{U} \to \mathcal{G}$ 被建模为随机预言机；
- P_1 持有集合 $V = \{v_i\}_{i=1}^m$；
- P_2 持有集合 $W = \{(w_j, t_j)\}_{j=1}^n$，其中 $w_j \in \mathcal{U}$，$t_j \in \mathbb{Z}^+$ 表示每个元素对应的数值。

协议流程

1. **Setup 阶段**：
   - P_1 与 P_2 分别生成随机私钥指数 $k_1, k_2 \in \mathbb{Z}_p^*$；
   - P_2 生成同态加密密钥对 (pk, sk)，将公钥 pk 发送给 P_1。
2. **Round 1 ($P_1 \to P_2$)**：
   - P_1 对每个元素 v_i 执行哈希并进行指数变换，发送 $\{H(v_i)^{k_1}\}_i$（乱序）给 P_2。
3. **Round 2 ($P_2 \to P_1$)**：
   - P_2 对 $\{H(v_i)^{k_1}\}$ 再执行一次指数操作，计算 $\{H(v_i)^{k_1 k_2}\}$ 形成集合 Z 并发送给 P_1；
   - 对于每个 (w_j, t_j)，P_2 计算 $H(w_j)^{k_2}$，并加密 t_j 得到 $Enc(t_j)$，将对 $(H(w_j)^{k_2}, Enc(t_j))$ 的乱序集合发送给 P_1。
4. **Round 3 (P_1 计算交集)**：
   - P_1 对接收到的 $H(w_j)^{k_2}$ 再执行 k_1 次幂，得到 $H(w_j)^{k_1 k_2}$；
   - 若 $H(w_j)^{k_1 k_2} \in Z$，则认为 $w_j \in V \cap W$；
   - P_1 对交集中所有 $Enc(t_j)$ 执行同态加法，得到加密的总和 $Enc(S_J)$；
   - P_1 可对密文进行随机化重加密后发送给 P_2。
5. **输出阶段（P_2 解密)**：
   - P_2 使用私钥 sk 解密 $Enc(S_J)$，得到交集中元素对应值的总和。

2 协议代码实现解析（Python 实现）

该实现在 Kaggle Notebook 中使用 Python 完成，结合大素数模拟乘法群、SHA-256 哈希函数与 Paillier 同态加密，具体如下，完整代码见附录。

基本设置

- 设定大素数 $\mathbb{P} = 2^{521} - 1$ 模拟群 $\mathcal{G}$；
- 使用 SHA-256 将字符串 ID 哈希成整数 $H(x) \in \mathbb{Z}_{\mathbb{P}}$；
- 指数操作使用 `pow(base, exponent, PRIME)` 实现群内幂；
- 同态加密使用 `phe.paillier` 库实现加法封装。

实现步骤对应

- **Round 0：** 使用 `paillier.generate_paillier_keypair()` 生成 (pk, sk)，用于同态加密；
- **Round 1：** P_1 计算每个 ID 的 $H(v_i)^{k_1}$，并发送乱序列表 `P1_masked`；
- **Round 2：**
  - P_2 对 Round1 中接收的数据执行 k_2 次幂，得到集合 Z；
  - 对每个 (w_j, t_j)，计算 $H(w_j)^{k_2}$，并加密 t_j 为 $Enc(t_j)$，形成对组；
- **Round 3：**
  - P_1 对每个 $H(w_j)^{k_2}$ 再执行 k_1 次幂，得到 $H(w_j)^{k_1 k_2}$；
  - 若命中 Z 集合，则累加对应 $Enc(t_j)$ 密文；
- **Round 4：** P_1 得到总和密文，P_2 解密还原明文总和。

最终输出

输出包括：

```
✅ 交集总金额：80
🔍 命中项（H^k1k2）前缀：['3438733556077085', '9033213999627467']
🧠 真实交集用户：{'user2', 'user3'}
```

可以看到成功验证了 Google Password Checkup 协议实现。

A Google Password Checkup

```python
import hashlib
import random
from phe import paillier

# ======== 参数设置 ========
P1_ids = ['user1', 'user2', 'user3', 'user4']
P2_data = [('user2', 50), ('user3', 30), ('user5', 90)]
PRIME = 2**521 - 1  # 模拟群 G（使用大素数）

# 哈希到群元素 G
def H(x: str) -> int:
    return int(hashlib.sha256(x.encode()).hexdigest(), 16) % PRIME

# ======== Round 0: 密钥生成 ========
# Paillier 密钥对（P2 生成公钥发给 P1）
pk, sk = paillier.generate_paillier_keypair()

# 双方各自选择随机指数 k1, k2
k1 = random.randint(2, PRIME-1)
k2 = random.randint(2, PRIME-1)

# ======== Round 1: P1 → P2: 发送 {H(vi)^k1} ========
P1_masked = [pow(H(vi), k1, PRIME) for vi in P1_ids]
random.shuffle(P1_masked)

# ======== Round 2: P2 → P1 ========
# Step 1: 对 Round1 的值再进行 ^k2 并发送 Z
Z = [pow(x, k2, PRIME) for x in P1_masked]
Z_set = set(Z)

# Step 2: 计算 {(H(wj)^k2, Enc(tj))}
masked_enc_list = []
for wj, tj in P2_data:
    h = H(wj)
    h_k2 = pow(h, k2, PRIME)
    enc_tj = pk.encrypt(tj)
    masked_enc_list.append((h_k2, enc_tj))
random.shuffle(masked_enc_list)

# ======== Round 3: P1 处理交集与加和 ========
intersection_sum = pk.encrypt(0)
hits = []

for h_k2, enc in masked_enc_list:
    h_k1k2 = pow(h_k2, k1, PRIME)
    if h_k1k2 in Z_set:
```

```
        intersection_sum += enc
        hits.append(h_k1k2)

# ======== Round 4: P1 → P2 解密 ========
final_sum = sk.decrypt(intersection_sum)

# ======== 输出结果 ========
print(f"  交集总金额: {final_sum}")
print(f"   命中项 (H^k1k2) 前缀: {[str(h)[:16] for h in hits]}")

expected = set([uid for uid, _ in P2_data]) & set(P1_ids)
print(f"   真实交集用户: {expected}")
```