# 实验 6

## ΠDDH 协议的实现

学院　网络空间安全学院

# 目录

# 1 实验目的

本实验尝试实现 https://eprint.iacr.org/2019/723.pdf 中 Figure 2 里展示的 ΠDDH 协议，达到 Google Password Checkup 验证的目的。

**DDH-based Private Intersection-Sum Protocol**

- **Inputs**:
  - Both parties: A group $\mathcal{G}$ of prime order, and an identifier space $\mathcal{U}$. A hash function $\mathsf{H} : \mathcal{U} \to \mathcal{G}$, modeled as a random oracle, that maps identifiers to random elements of $\mathcal{G}$.
  - $\mathsf{P}_1$ : Set $V = \{v_i\}_{i=1}^{m_1}$, where $v_i \in \mathcal{U}$.
  - $\mathsf{P}_2$: Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$, with $w_i \in \mathcal{U}$, $t_i \in \mathbb{Z}^+$.
- **Setup**:
  - Each $\mathsf{P}_i$ chooses a random private exponent $k_i$ in the group $\mathcal{G}$.
  - $\mathsf{P}_2$ generates a fresh key-pair $(pk, sk) \leftarrow \mathsf{AGen}(\lambda)$ for the additive homomorphic encryption scheme and sends the public key $pk$ with $\mathsf{P}_1$.
- **Round 1** ($\mathsf{P}_1$):
  1) For each element $v_i$ in its set, $\mathsf{P}_1$ applies the hash function and then exponentiates it using its key $k_1$, thus computing $\mathsf{H}(v_i)^{k_1}$.
  2) $\mathsf{P}_1$ sends $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ to Party 2 in shuffled order.
- **Round 2** ($\mathsf{P}_2$):
  1) For each element $\mathsf{H}(v_i)^{k_1}$ received from $\mathsf{P}_1$ in the previous step, $\mathsf{P}_2$ exponentiates it using its key $k_2$, computing $\mathsf{H}(v_i)^{k_1 k_2}$.
  2) $\mathsf{P}_2$ sends $Z = \{\mathsf{H}(v_i)^{k_1 k_2}\}_{i=1}^{m_1}$ to $\mathsf{P}_1$ in shuffled order.
  3) For each item $(w_j, t_j)$ in its input set, $\mathsf{P}_2$ applies the hash function to the first element of the pair and exponentiates it using key $k_2$. It encrypts the second element of the pair using the key $pk$ for the additive homomorphic encryption key. It thus computes the pair. $\mathsf{H}(w_j)^{k_2}$ and $\mathsf{AEnc}(t_j)$.
  4) $\mathsf{P}_2$ sends the set $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))\}_{j=1}^{m_2}$ to $\mathsf{P}_1$ in shuffled order.
- **Round 3** ($\mathsf{P}_1$):
  1) For each item $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))$ received from $\mathsf{P}_2$ in Round 2 Step 4, $\mathsf{P}_1$ exponentiates the first member of the pair using $k_1$, thus computing $(\mathsf{H}(w_j)^{k_1 k_2}, \mathsf{AEnc}(t_j))$.
  2) $\mathsf{P}_1$ computes the intersection set $J$:
  $$J = \{j : \mathsf{H}(w_j)^{k_1 k_2} \in Z\}$$
  where Z is the set received from $\mathsf{P}_1$ in Round 1.
  3) For all items in the intersection, $\mathsf{P}_1$ homomorphically adds the associated ciphertexts, and computes a ciphertext encrypting the intersection-sum $S_J$:
  $$\mathsf{AEnc}(pk, S_J) = \mathsf{ASum}\left(\{\mathsf{AEnc}(t_j)\}_{j \in J}\right) = \mathsf{AEnc}\left(\sum_{j \in J} t_j\right)$$
  $\mathsf{P}_1$ then randomizes the ciphertext using $\mathsf{ARefresh}$ and sends it to $\mathsf{P}_2$.
- **Output** ($\mathsf{P}_2$): $\mathsf{P}_2$ decrypts the ciphertext received in Round 3 using the secret key $sk$ to recover the intersection-sum $S_J$.

Figure 2: $\Pi_{\mathsf{DDH}}$: Our deployed DDH-based Private Intersection-Sum protocol.

图 1: 协议流程

# 2 实验内容

## 2.1 实验步骤

### 2.1.1 参数设置

预处理中，首先进行密码学基础输入参数的设置和初始化：

- 公共输入
- $P_1$ 的输入

  集合 $V = \{v_i\}_{i=1}^{m_1}$ $v_i \in \mathcal{U}$，表示 P 持有的标识符列表

- $P_2$ 的输入

  集合 $W = \{(w_i, t_i)\}_{i=1}^{m_2}$； $w_i \in \mathcal{U}$： 标识符 $t_i \in \mathbb{Z}^+$： 关联值

下面是关于群、哈希函数、私钥及同态加密方案的选取，具体如下：

| 群 $\mathcal{G}$ | group$_o rder$ |
|---|---|
| 哈希函数 $\mathsf{H}: \mathcal{U} \to \mathcal{G}$ | SHA-256 实现 hash$_t o_g roup$ |
| 私钥 $k_1, k_2$ | k1 = random.randint(1, group$_o rder - 1$) |
| $(pk, sk) \leftarrow \mathsf{AGen}(\lambda)$ | 2048 位 RSA 实现 AdditiveHomomorphicEncryption |

表 1: 参数设置

### 2.1.2 Round 1

Round 1 的协议公式为

$$P_1 \text{ 发送：} \{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$$

对每个 $v_i \in V$：先计算哈希 $\mathsf{H}(v_i)$（hash_to_group 方法）再进行指数运算 $\mathsf{H}(v_i)^{k_1} \mod p$（exponentiate 方法）最后发送结果列表给 $P_2$

```
def round1(self):
    return [self.group.exponentiate(v, self.k1) for v in self.V]#计算H(v_i)^k1
```

### 2.1.3 Round 2

Round 2 的协议公式为：

1. 计算 $Z = \{\mathsf{H}(v_i)^{k_1 k_2}\}$

2. 计算 $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j)\}$

即先进行双重盲化，对收到的每个 $\mathsf{H}(v_i)^{k_1}$，计算 $\mathsf{H}(v_i)^{k_1 k_2}$；再加密关联值，对每个 $(w_j, t_j) \in W$ 计算 $\mathsf{H}(w_j)^{k_2}$ 并加密 $t_j$ 得到 $\mathsf{AEnc}(t_j)$（RSA-OAEP 模拟）；最后发送 $Z$ 和加密对列表。

```
def round2(self, received_from_p1):
    Z = [self.group.exponentiate(item, self.k2) for item in received_from_p1]
    #计算H(v_i)^k1k2
    encrypted_pairs = [
        (self.group.exponentiate(w, self.k2), self.ahe.encrypt(t))
        #(H(w_j)^k2, Enc(t_j))
        for (w, t) in self.W
    ]
    return Z, encrypted_pairs
```

### 2.1.4 Round 3

Round 3 的协议公式为:

1. 计算交集 $J = \{j : \mathsf{H}(w_j)^{k_1 k_2} \in Z\}$
2. 同态求和 $\mathsf{AEnc}(\sum_{j \in J} t_j)$

即先计算交集，对每个 $\mathsf{H}(w_j)^{k_2}$，计算 $\mathsf{H}(w_j)^{k_1 k_2}$，检查是否存在于 $Z$ 中（通过集合快速查找）；再同态求和，初始化和为加密的 0（ahe.encrypt(0)）并对交集中的 $\mathsf{AEnc}(t_j)$ 累加（ahe.add）；最后返回加密的总和（encrypted_sum）。

```
def round3(self, Z, encrypted_pairs, ahe):
    Z_set = set(Z)
    sum_cipher = ahe.encrypt(0)  #初始化为加密的0

    for (w_k2, enc_t) in encrypted_pairs:
        w_k1k2 = self.group.exponentiate(w_k2, self.k1)#计算H(w_j)^k1k2
        if w_k1k2 in Z_set: #判断是否在交集中
            sum_cipher = ahe.add(sum_cipher, enc_t)#同态加

    return sum_cipher
```

## 2.2 实验结果

最终运行结果如下:

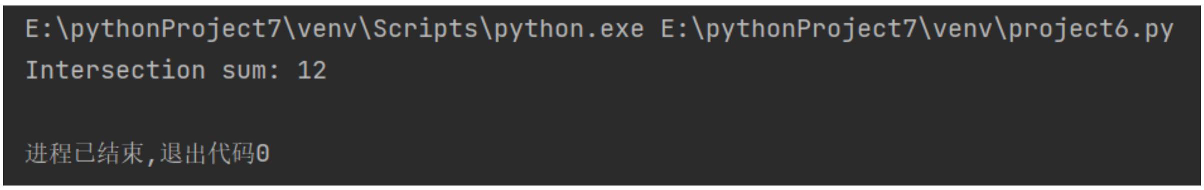


图 2: 结果