



网络安全创新创业实践第六次实验

Google Password Checkup 验证

学院: 网络空间安全学院

专业: 密码科学与技术

姓名: 金鑫悦

学号: 202200460042

目录

1 实验任务	2
2 实验环境	2
3 实验原理	2
3.1 协议详细介绍	2
3.1.1 输入和设置	2
3.1.2 协议的三个阶段	2
3.1.3 协议的安全性分析	3
4 代码实现	3
5 结果分析	7
6 总结与感悟	8

1 实验任务

google password checkup，参考论文 <https://eprint.iacr.org/2019/723.pdf> 的 section 3.1，也即 Figure 2 中展示的协议，用 python 实现该协议

2 实验环境

硬件环境：

12th Intel (R) Core (TM) i7-12700H

软件环境：

windows11、python3.12 (64-bit)、pycharm2024.1.4

3 实验原理

3.1 协议详细介绍

3.1.1 输入和设置

- 输入：

- 参与方 P1 和 P2 分别持有不同的数据集。
- P1 持有一组标识符 $V = \{v_1, v_2, \dots, v_{m_1}\}$ 。
- P2 持有一组标识符 $W = \{(w_1, t_1), (w_2, t_2), \dots, (w_{m_2}, t_{m_2})\}$ ，其中 w_i 是标识符， t_i 是与每个标识符 w_i 相关联的整数值。

- 设置：

- 每方选择一个随机的私钥，并生成相应的加密密钥对。
- P2 生成一对加密密钥 (pk, sk) ，其中 pk 是公钥， sk 是私钥，用于加法同态加密方案。

3.1.2 协议的三个阶段

1. 第一轮：P1 发送哈希结果

- P1 对每个标识符 v_i 计算哈希值：

P1 对其数据集中的每个元素 v_i 进行哈希，并用私钥 k_1 对其进行指数运算，得到 $H(v_i)^{k_1}$

- 将这些哈希结果 $H(v_1)^{k_1}, H(v_2)^{k_1}, \dots, H(v_{m_1})^{k_1}$ 发送给 P2，顺序打乱。

2. 第二轮：P2 计算并返回加密数据

- P2 对 P1 发送的哈希值进行计算：

P2 用其私钥 k_2 对每个哈希值 $H(v_i)^{k_1}$ 进行指数运算，得到 $H(v_i)^{k_1 k_2}$

- P2 计算每个标识符 w_j 的哈希值 $H(w_j)^{k_2}$ ，并对与 w_j 关联的整数 t_j 使用加法同态加密，得到 $AEnc(t_j)$ 。

- P2 发送加密后的结果 $\{(H(w_1)^{k_2}, AEnc(t_1)), (H(w_2)^{k_2}, AEnc(t_2)), \dots, (H(w_{m_2})^{k_2}, AEnc(t_{m_2}))\}$ 给 P1，顺序打乱。

3. 第三轮：P1 计算交集并返回加密和

- P1 计算交集并加密求和结果：

P1 对每个哈希值 $H(w_j)^{k_2}$ 进行指数运算，得到 $H(w_j)^{k_1 k_2}$

$$J = \{j : H(w_j)^{k_1 k_2} \in Z\}$$

- 对交集中的所有元素，P1 同态加密地将关联的整数值进行加总，得到加密的交集和 $AEnc(pk, S_J)$ 。
- P1 随后使用 $ARefresh$ 操作随机化该密文，并将结果发送给 P2。

4. 输出阶段：P2 解密结果

P2 使用自己的私钥 sk 对 P1 返回的随机化密文进行解密，得到交集的整数和 S_J 。

3.1.3 协议的安全性分析

- 协议的安全性基于加法同态加密和 DDH 假设。协议证明保证了在半诚实模型下，每方只能知道交集的大小和交集的求和结果，无法获得其他信息。
- 使用混合算法和 DDH 难题的假设，可以保证协议的安全性。每方的视图可以通过模拟器进行模拟，确保各方无法通过协议泄露任何多余的信息。

4 代码实现

1. crypto_utils.py，含哈希、群运算、Paillier 同态加密等函数，详细介绍见注释

```
1 import hashlib
2 import random
3 from phe import paillier
4
5 # 设定群 G：使用素数阶 p 的乘法群 Z_p^* 来模拟 DDH 安全群
6 PRIME_P = 2 ** 127 - 1 # 取一个大的素数，便于模运算
7
8 # 哈希函数 H：将输入字符串映射为群 G 中的元素
9 def H(x):
10     h = hashlib.sha256() # 使用 SHA-256 哈希函数
11     h.update(x.encode('utf-8')) # 编码字符串
12     return int(h.hexdigest(), 16) % PRIME_P # 将结果映射到模 p 的整数上
13
14 # 生成随机指数 k，用于 exponentiation
15 def gen_exponent():
16     return random.randint(2, PRIME_P - 2)
17
```

```
18 # 群上指数运算：计算 base^exponent mod p
19 def modexp(base, exponent):
20     return pow(base, exponent, PRIME_P)
21
22 # 生成 Paillier 同态加密的公私钥对
23 def paillier_keygen():
24     public_key, private_key = paillier.generate_paillier_keypair()
25     return public_key, private_key
26
27 # 同态加密：用公钥加密一个整数值 val
28 def AEnc(pk, val):
29     return pk.encrypt(val)
30
31 # 同态解密：用私钥解密密文 ciphertext
32 def ADec(sk, ciphertext):
33     return sk.decrypt(ciphertext)
34
35 # 同态求和：将密文列表相加，利用 Paillier 的加法同态性
36 def ASum(ciphertexts):
37     total = ciphertexts[0]
38     for ct in ciphertexts[1:]:
39         total += ct
40     return total
41
42 # 同态密文刷新 (re-randomize)：用于打乱密文，防止关联性攻击
43 def ARefresh(ciphertext):
44     return ciphertext.public_key.encrypt(ciphertext.ciphertext(False))
```

2. party1.py, 客户端 P1 协议逻辑 (Round 1 & 3), 详细介绍见注释

```
1 from crypto_utils import H, gen_exponent, modexp, ASum
2
3 class Party1:
4     def __init__(self, v_list):
5         # Party1 拥有的标识符集合 v_list, 例如 ['u1', 'u2', 'u3']
6         self.v_list = v_list
7         self.k1 = gen_exponent() # P1 随机选取私钥指数 k1
8         self.intersection = [] # 用于记录交集元素在 P2 中的索引
9
10    # 协议第一轮：对每个 v 进行 H(v)^k1 运算后发送给 P2
11    def round1(self):
12        return [modexp(H(v), self.k1) for v in self.v_list]
13
```

```

14     # 协议第三轮：接收 P2 返回的  $(H(w)^{k2}, AEnc(t))$  对，判断是否属于交集
15     def round3(self, data_from_p2, pk):
16         intersection = [] # 记录交集在 P2 数据中的索引
17         encrypted_vals = [] # 存储交集中对应的加密值 AEnc(t)
18
19         for (val, ct) in data_from_p2:
20             val1 = modexp(val, self.k1) # 还原为  $H(w)^{k1 \cdot k2}$ 
21             if val1 in self.Z: # 若出现在 P1 Round1 中的哈希集合中，则为交集
22                 j = self.Z.index(val1)
23                 intersection.append(j)
24                 encrypted_vals.append(ct)
25
26             self.intersection = intersection # 保存交集索引
27             ct_sum = ASum(encrypted_vals) # 对交集项的 AEnc(t) 同态求和
28             return ct_sum
29
30     # 设置来自 P2 的 Z 集合，即  $H(w)^{k2}$  的集合 (Round2 中发来)
31     def set_Z(self, Z):
32         self.Z = Z

```

3. party2.py, 服务端 P2 协议逻辑 (Round 2 解密), 详细介绍见注释

```

1  from crypto_utils import H, gen_exponent, modexp, paillier_keygen, AEnc
2
3  class Party2:
4      def __init__(self, w_t_list):
5          # Party2 拥有的标识符-值对  $(w, t)$ , 例如  $[('u2', 10), ('u3', 20)]$ 
6          self.w_t_list = w_t_list
7          self.k2 = gen_exponent() # P2 随机生成指数 k2
8          self.pk, self.sk = paillier_keygen() # 生成 Paillier
9              加密的公钥和私钥
10
11     # 获得 Paillier 公钥, 发送给 Party1 用于同态加密
12     def get_public_key(self):
13         return self.pk
14
15     # 协议第二轮:
16     # 接收来自 P1 的  $H(v)^{k1}$  值, 对其做幂运算 ( $\wedge k2$ ) , 得到  $H(v)^{k1 \cdot k2}$ 
17         作为 Z
18     # 同时将本地的  $H(w)^{k2}$  和  $AEnc(t)$  对组成列表返回
19     def round2(self, data_from_p1):
20         # 对每个  $H(v)^{k1}$  再做一次  $\wedge k2$ , 得到  $H(v)^{k1 \cdot k2}$ 
21         self.Z = [modexp(x, self.k2) for x in data_from_p1]

```

```
20
21     results = []
22     for (w, t) in self.w_t_list:
23         val = modexp(H(w), self.k2) # 计算 H(w)^k2
24         ct = AEnc(self.pk, t) # 加密 value t, 得到 AEnc(t)
25         results.append((val, ct)) # 返回 (H(w)^k2, AEnc(t)) 对
26
27     return self.Z, results
28
29     # 解密 P1 返回的 Paillier 密文之和 (即交集中所有 t_j 的总和)
30     def decrypt_sum(self, ct_sum):
31         return self.sk.decrypt(ct_sum)
```

4. protocol.py。封装完整三轮协议执行流程

```
1 from party1 import Party1
2 from party2 import Party2
3
4 def run_protocol(v_list, w_t_list):
5     P1 = Party1(v_list)
6     P2 = Party2(w_t_list)
7
8     X = P1.round1()
9     pk = P2.get_public_key()
10    Z, data = P2.round2(X)
11
12    P1.set_Z(Z)
13    ct_sum = P1.round3(data, pk)
14
15    final_sum = P2.decrypt_sum(ct_sum)
16    return final_sum, P1.intersection
```

5. test_protocol.py, 示例测试入口

```
1 from protocol import run_protocol
2
3 # Example data
4 v_list = ['u1', 'u2', 'u3']
5 w_t_list = [('u2', 10), ('u3', 20), ('u4', 40)]
6
7 sum_result, intersection = run_protocol(v_list, w_t_list)
8 print("Intersection indexes:", intersection)
9 print("Intersection sum:", sum_result)
```

5 结果分析

```
Intersection indexes: [1, 2]
Intersection sum: 30
```

图 1: 运行结果 1

1. 输入数据

P1 的标识符集合 v_list :

$$v_list = \{u1', u2', u3'\}$$

P1 拥有标识符集合 $\{u1, u2, u3\}$ 。

P2 的标识符-值对集合 w_t_list :

$$w_t_list = \{(u2', 10), (u3', 20), (u4', 40)\}$$

P2 拥有标识符与整数值对的集合 $\{(u2, 10), (u3, 20), (u4, 40)\}$ 。这意味着 P2 持有标识符 ‘u2’ 和 ‘u3’ 的关联值分别为 10 和 20，而 ‘u4’ 对应的值为 40。

2. 协议流程回顾

- 第一轮: P1 向 P2 发送哈希值

P1 使用哈希函数 $H(v)$ 将自己的标识符 $v_1 = u1', v_2 = u2', v_3 = u3'$ 转换为群元素，然后对每个元素执行指数运算。P1 发送以下数据给 P2:

$$H(u1)^{k1}, H(u2)^{k1}, H(u3)^{k1}$$

- 第二轮: P2 计算并返回加密值

P2 对 P1 发送的哈希值进行操作:

$$H(w_j)^{k2}, \quad j \in \{2, 3, 4\}$$

P2 对每个整数值 $t_1 = 10, t_2 = 20, t_3 = 40$ 进行 Paillier 加密，得到密文 $AEnc(t_j)$ 。

P2 返回的数据为:

$$\{(H(u2)^{k2}, AEnc(10)), (H(u3)^{k2}, AEnc(20)), (H(u4)^{k2}, AEnc(40))\}$$

- 第三轮: P1 计算交集并返回加密和

P1 接收到 P2 返回的数据后，对每个标识符 $H(w_j)^{k2}$ 使用自己的私钥 k_1 进行指数运算，得到 $H(w_j)^{k1k2}$ ，然后检查这些值是否出现在 P1 的数据集中。P1 计算交集:

$$J = \{j : H(w_j)^{k1k2} \in Z\}$$

其中 Z 是 P1 在第一轮发送的数据中接收到的哈希值集合。

交集是 $\{u2, u3\}$ ，对应的索引为 $[1, 2]$ （索引从 0 开始）。

P1 同态加密地对交集的值进行求和:

$$10 + 20 = 30$$

3. 最终输出

交集索引: 交集是 $\{u2, u3\}$ ，在 P2 返回的数据中的索引为 [1, 2]。因此，交集索引为：

Intersection indexes: [1, 2]

交集和: 交集项对应的整数值为 10 和 20，它们的和为 30。因此，交集和为：

Intersection sum: 30

4. 结果分析

交集索引: 交集包含了 P1 和 P2 数据集合中都存在的元素：‘u2’ 和 ‘u3’。它们在 P2 返回数据中的位置分别为 1 和 2，因此交集索引为 ‘[1, 2]’。

交集和: 交集项对应的整数值是 10 和 20，它们的和为 30。这个结果通过同态加密求和得到，确保了数据隐私的同时完成了求和计算。

6 总结与感悟

实验过程中的问题:

- 协议的实现挑战:** 在实现基于 Google Password Checkup 的协议时，由于该协议涉及私密数据的计算，如何确保协议中每一轮的隐私保护和有效性是一个挑战。尤其是在第 3.1 节中，协议的安全性和正确性要求高度精准的加密处理和算法实现，导致了实验中频繁的调试和优化。
- 通信复杂性与效率优化:** 协议的设计需要进行多轮数据交换，如何在保证隐私的同时优化通信效率，减少数据交换的开销，是实验中的一个难点。尤其是当协议中涉及到大量的数据处理时，如何通过算法优化（例如使用更加高效的加密算法和数据传输策略）来提升效率，成为实验中的核心问题之一。

实验亮点:

- 私密计算协议实现:** 本实验通过实际实现 Google Password Checkup 协议，成功模拟了密码检查机制中私密计算的过程。通过对协议的每一阶段进行细致的代码实现，我更好地理解私密计算如何在真实环境中保护用户数据，同时完成所需的统计计算。
- 协议中加密技术的应用:** 协议的实现过程中，密钥交换、数据加密、加密后的数据交换等技术的应用成为实验的亮点。通过实现具体的加密算法，我对加密协议的使用和细节有了更深刻的理解，并验证了如何在隐私保护的前提下进行计算和数据交换。

实验收获: 通过本次实验，我对用于隐私保护的 Google Password Checkup 协议有了更深入的理解。实验过程中，通过实现和分析协议，我明白了如何使用加密算法进行私密计算，并保证计算结果的正确性和安全性。