



Project 6: 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: 李双平

学号: 202200180026

2025 年 8 月 15 日

目录

1 Google Password Checkup 验证	2
1.1 协议流程	2
1.2 代码解释	3
1.3 实验结果	5

1 Google Password Checkup 验证

1.1 协议流程

为实现论文中 Figure 2 (DDH-based PSI-Sum)，我们采用：一个素数阶子群 $\mathbb{G} = \langle g \rangle$ (阶为 q , 模数为 p , 满足 $p = kq + 1$)、随机预言机模型的哈希到群映射 $H : \{0, 1\}^* \rightarrow \mathbb{G}$, 以及可加同态的 Paillier 加密。两方输入如下： \mathcal{A} 持有带权集合 $P = \{(w_i, t_i)\}$, \mathcal{B} 持有集合 $Q = \{v_j\}$. 目标是让 \mathcal{A} 得到 $\sum_{x \in P \cap Q} t_x$, 但双方除该和以外不泄露其它信息。

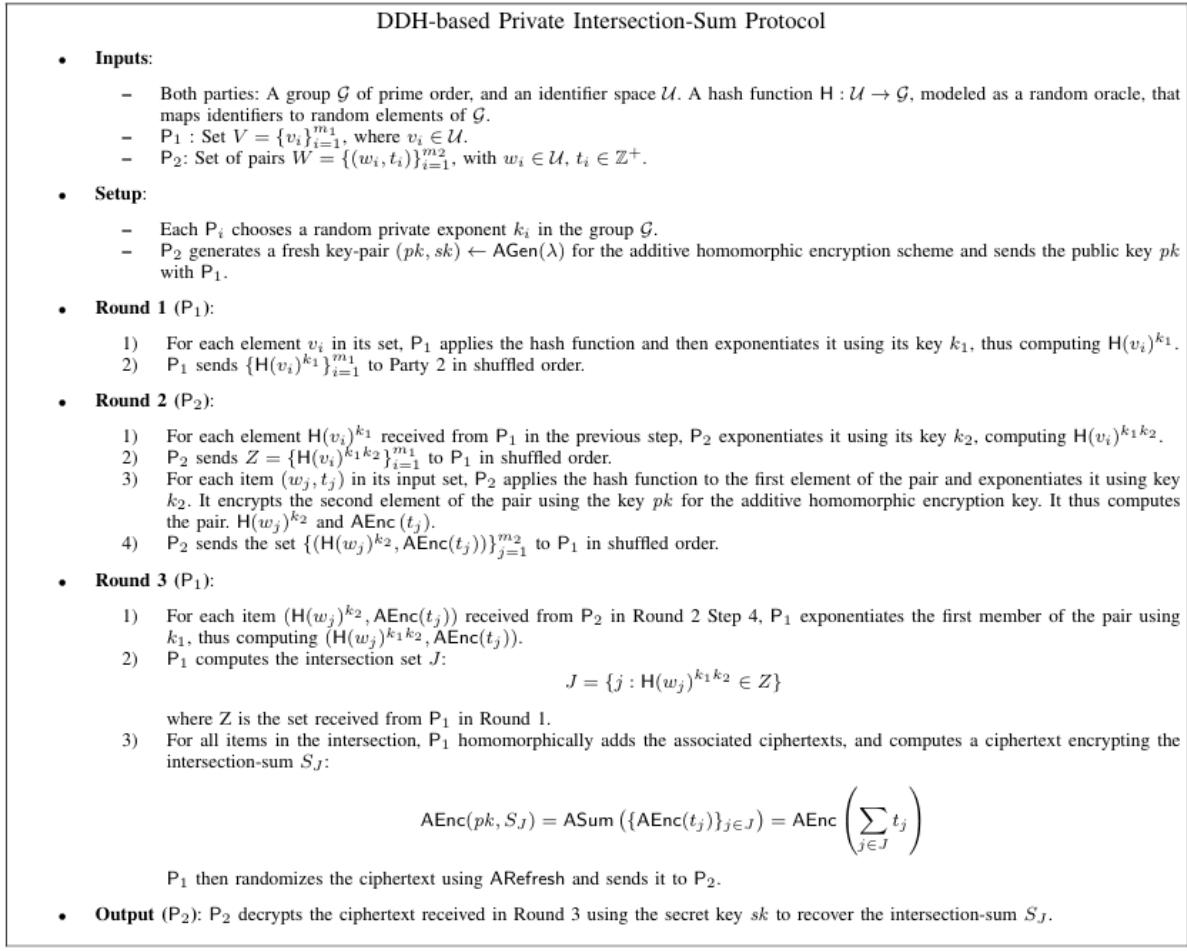


Figure 2: Π_{DDH} : Our deployed DDH-based Private Intersection-Sum protocol.

图 1: 协议流程图

Setup. \mathcal{A} 选取掩码指数 $k_1 \leftarrow \mathbb{Z}_q^*$ 。 \mathcal{B} 选取 $k_2 \leftarrow \mathbb{Z}_q^*$, 生成 Paillier 密钥对 (pk, sk) 并发送 pk 给 \mathcal{A} 。

Round 1 ($\mathcal{A} \rightarrow \mathcal{B}$). 对每个 (w_i, t_i) , 计算标签 $L_i = H(w_i)^{k_1} \in \mathbb{G}$ 与密文 $C_i =_{\mathsf{pk}} (t_i)$; 打乱并发送 $\{(L_i, C_i)\}$ 。

Round 2 (\mathcal{B}). 对每个收到的 (L_i, C_i) , 令 $L_i^{(2)} = L_i^{k_2} = H(w_i)^{k_1 k_2}$, 据此将同标签的 C_i 归入同一桶；同时对本地元素 v_j 计算 $M_j = H(v_j)^{k_2}$, 打乱并发送集合 $\{M_j\}$ 给 \mathcal{A} 。

Round 3 (\mathcal{A}). 对每个 M_j 计算 $S_j = M_j^{k_1} = H(v_j)^{k_1 k_2}$, 打乱后将 $\{S_j\}$ 发送给 \mathcal{B} 。

Round 4 ($\mathcal{B} \rightarrow \mathcal{A}$). 用 S_j 与本地索引的 $L_i^{(2)}$ 进行匹配；对每个匹配到的桶，同态累加对应 C_i 得 $C^* =_{\mathsf{pk}} (\sum t_i)$ ；对 C^* 重新随机化并返回给 \mathcal{A} 。最后， \mathcal{A} 用 sk 解密得到 $\sum_{x \in P \cap Q} t_x$ 。

1.2 代码解释

1. Paillier 密钥生成 keygen:

```

1 def keygen(bits=512):
2     p, q = gen_prime(bits//2), gen_prime(bits//2)
3     while p == q: q = gen_prime(bits//2)
4     n, g, n2 = p*q, p*q + 1, (p*q)*(p*q)
5     lam = (p - 1) * (q - 1) // math.gcd(p - 1, q - 1)
6     mu = pow(L(pow(g, lam, n2), n), -1, n)
7     return PubKey(n, g), PriKey(lam, mu)

```

说明: 生成两个大素数 p, q 并构造 $n = pq$ 与 $g = n + 1$, 计算 λ ($p - 1$ 与 $q - 1$ 的最小公倍数) 和 μ ($L(g^\lambda \bmod n^2)$ 的逆元), 输出公钥 (n, g) 和私钥 (λ, μ) 。该密钥对用于 Paillier 同态加密。

2. Paillier 加解密与同态运算 enc/dec/add/mul:

```

1 def enc(pk, m):
2     n, g, n2 = pk.n, pk.g, pk.n2
3     m %= n
4     while True:
5         r = secrets.randbelow(n)
6         if math.gcd(r, n) == 1 and r != 0: break
7     return (pow(g, m, n2) * pow(r, n, n2)) % n2
8
9 def dec(pk, sk, c):
10    n, n2 = pk.n, pk.n2
11    return (L(pow(c, sk.lam, n2), n) * sk.mu) % n
12
13 def add(pk, c1, c2): return (c1 * c2) % pk.n2
14 def mul(pk, c, k): return pow(c, k, pk.n2)

```

说明: enc 使用随机 r 生成密文 $(g^m r^n) \bmod n^2$; dec 利用私钥 λ, μ 解密; add 实现密文相乘对应明文加法; mul 实现密文的幂运算对应明文的数乘。这些是协议中求和的核心运算。

3. DDH 群生成与哈希到群元 gen_group/hash_item:

```

1 def gen_group(q_bits=256):
2     q = gen_prime(q_bits)
3     for k in range(2, 10000):
4         p = k * q + 1
5         if is_prime(p):
6             for _ in range(50):
7                 h = secrets.randbelow(p - 3) + 2
8                 g = pow(h, k, p)

```

```

9         if pow(g, q, p) == 1 and g != 1:
10            return Group(p, q, g)
11        raise RuntimeError("生成群失败")
12
13    def hash_item(G, x: bytes):
14        e = int.from_bytes(hashlib.sha256(x).digest(), 'big') % G.q
15        if e == 0: e = 1
16        return pow(G.g, e, G.p)

```

说明: `gen_group` 找到素数 $p = kq + 1$ 并生成生成元 g 形成 DDH 群 (p, q, g) ; `hash_item` 将任意字节串映射到群元素, 通过 SHA-256 取模 q 后指数运算, 保证输入不可逆映射到群上。

4. 客户端 A 逻辑 A 类:

```

1  class A:
2      def __init__(self, G, data):
3          self.G, self.data = G, data[:]
4          self.k1 = secrets.randrange(G.q - 1) + 1
5          self.pk = None
6
7      def recv_pk(self, pk): self.pk = pk
8
9      def r1(self):
10         out = []
11         for (w, t) in self.data:
12             lbl = pow(hash_item(self.G, w), self.k1, self.G.p)
13             c = enc(self.pk, t)
14             out.append((lbl, c))
15         secrets.SystemRandom().shuffle(out)
16         return out
17
18     def r3(self, M_list):
19         S = [pow(M, self.k1, self.G.p) for M in M_list]
20         secrets.SystemRandom().shuffle(S)
21         return S
22
23     def recv_res(self, agg, sk):
24         return dec(self.pk, sk, agg)

```

说明: A 拥有自己的数据 (w, t) , 用密钥 k_1 对标签做幂运算实现盲化, 并将 t 用 Paillier 加密。第 1 轮发送盲化标签与密文给 B; 第 3 轮对 B 发来的标签做 k_1 次方后返回; 最后解密求和结果。

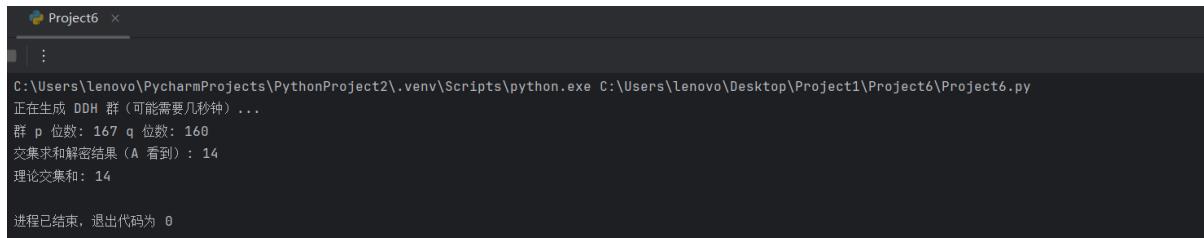
5. 服务端 B 逻辑 B 类:

```
1  class B:
2      def __init__(self, G, items):
3          self.G, self.items = G, items[:]
4          self.k2 = secrets.randrange(G.q - 1) + 1
5          self.pk, self.sk = None, None
6          self.map = {}
7
8      def gen_pk(self, bits=512):
9          self.pk, self.sk = keygen(bits)
10         return self.pk
11
12     def r2(self, lbl_enc):
13         self.map = {}
14         for (lbl, c) in lbl_enc:
15             Lk = pow(lbl, self.k2, self.G.p)
16             self.map.setdefault(Lk, []).append(c)
17         M_list = [pow(hash_item(self.G, v), self.k2, self.G.p) for v in
18                   self.items]
19         secrets.SystemRandom().shuffle(M_list)
20         return M_list
21
22     def r4(self, S_list):
23         agg = None
24         for S in S_list:
25             if S in self.map:
26                 for c in self.map[S]:
27                     agg = c if agg is None else add(self.pk, agg, c)
28         if agg is None: agg = enc(self.pk, 0)
29         return re_rand(self.pk, agg)
```

说明: B 生成 Paillier 密钥对, 将收到的标签用 k_2 盲化并存入映射表; 对自己的每个元素也计算盲化标签返回给 A; 收到 A 返回的标签后, 找到交集密文并同态求和, 最后重随机化密文发回 A。

1.3 实验结果

实验结果如下:



The screenshot shows the PyCharm terminal window titled "Project6". The output of the script "Project6.py" is displayed, which performs a DDH attack. The terminal shows the following text:

```
C:\Users\lenovo\PycharmProjects\PythonProject2\.venv\Scripts\python.exe C:\Users\lenovo\Desktop\Project1\Project6\Project6.py
正在生成 DDH 群（可能需要几秒钟）...
群 p 位数: 167 q 位数: 160
交集求和解密结果 (A 看到) : 14
理论交集和: 14

进程已结束，退出代码为 0
```

图 2: 实验结果