Project6 Google Password Checkup 验证

学院 网络空间安全

专业 网络空间安全

学号 202200460149

班级姓名 网安 22.1 班张弛

2025 年 8 月 8 日

目录

1 实验任务

Project 6: Google Password Checkup 验证

来自刘巍然老师的报告 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，尝试实现该协议，（编程语言不限）。

2 协议实现

根据链接里的 pdf，协议的流程如下图：

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 $\mathcal{G}$.
  - P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$, where $v_i \in \mathcal{U}$.
  - 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 P_i chooses a random private exponent k_i in the group $\mathcal{G}$.
  - 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 P_1.
- **Round 1** (P_1):
  1) For each element v_i in its set, P_1 applies the hash function and then exponentiates it using its key k_1, thus computing $\mathsf{H}(v_i)^{k_1}$.
  2) P_1 sends $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ to Party 2 in shuffled order.
- **Round 2** (P_2):
  1) For each element $\mathsf{H}(v_i)^{k_1}$ received from P_1 in the previous step, P_2 exponentiates it using its key k_2, computing $\mathsf{H}(v_i)^{k_1 k_2}$.
  2) P_2 sends $Z = \{\mathsf{H}(v_i)^{k_1 k_2}\}_{i=1}^{m_1}$ to P_1 in shuffled order.
  3) For each item (w_j, t_j) in its input set, 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) P_2 sends the set $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))\}_{j=1}^{m_2}$ to P_1 in shuffled order.
- **Round 3** (P_1):
  1) For each item $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))$ received from P_2 in Round 2 Step 4, 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) 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 P_1 in Round 1.
  3) For all items in the intersection, P_1 homomorphically adds the associated ciphertexts, and computes a ciphertext encrypting the intersection-sum S_J:
  $$\mathsf{AEnc}(pk, S_J) = \mathsf{ASum}\left(\{\mathsf{AEnc}(t_j)\}_{j \in J}\right) = \mathsf{AEnc}\left(\sum_{j \in J} t_j\right)$$
  P_1 then randomizes the ciphertext using $\mathsf{ARefresh}$ and sends it to P_2.
- **Output** (P_2): P_2 decrypts the ciphertext received in Round 3 using the secret key sk to recover the intersection-sum S_J.

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

图 1 协议流程

具体而言，我们可以：

输入

- **双方输入**:
  - 一个素数阶群 $\mathcal{G}$ 和一个标识符空间 $\mathcal{U}$。
  - 哈希函数 $\mathsf{H} : \mathcal{U} \to \mathcal{G}$（建模为随机预言机），将标识符映射到群 $\mathcal{G}$ 的随机元素。
- P_1 的输入: 集合 $V = \{v_i\}_{i=1}^{m_1}$，其中 $v_i \in \mathcal{U}$。
- P_2 的输入: 集合 $W = \{(w_i, t_i)\}_{i=1}^{m_2}$，其中 $w_i \in \mathcal{U}$，$t_i \in \mathbb{Z}^+$。

初始化

- P_1 为群 $\mathcal{G}$ 中的每个元素选择一个随机私钥指数 k_i。

- P_2 为加法同态加密方案生成新密钥对 $(pk, sk) \leftarrow \mathsf{AGen}(\lambda)$，并将公钥 pk 发送给 P_1。

第一轮 (P_1)

1. 对集合中的每个元素 v_i，P_1 先应用哈希函数，再用私钥 k_1 进行指数运算，计算 $\mathsf{H}(v_i)^{k_1}$。
2. P_1 将 $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ 以乱序方式发送给 P_2。

第二轮 (P_2)

1. 对从 P_1 收到的每个元素 $\mathsf{H}(v_i)^{k_1}$，P_2 用私钥 k_2 进行指数运算，计算 $\mathsf{H}(v_i)^{k_1k_2}$。
2. P_2 将 $Z = \{\mathsf{H}(v_i)^{k_1k_2}\}_{i=1}^{m_1}$ 以乱序方式发送给 P_1。
3. 对输入集合中的每个项 (w_j, t_j)，P_2 对第一项 w_j 应用哈希函数并用私钥 k_2 进行指数运算，得到 $\mathsf{H}(w_j)^{k_2}$；对第二项 t_j 用公钥 pk 进行加法同态加密，得到 $\mathsf{AEnc}(t_j)$。
4. P_2 将集合 $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))\}_{j=1}^{m_2}$ 以乱序方式发送给 P_1。

第三轮 (P_1)

1. 对从 P_2 收到的每对 $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))$，$\mathsf{P}_1$ 用私钥 k_1 对第一项进行指数运算，得到 $(\mathsf{H}(w_j)^{k_1k_2}, \mathsf{AEnc}(t_j))$。
2. P_1 计算交集集合 J：
$$J = \{j : \mathsf{H}(w_j)^{k_1k_2} \in Z\}$$
其中 Z 是从 P_2 收到的集合。
3. 对交集中的所有项，P_1 同态地累加关联的密文，计算加密的交集和 S_J：
$$\mathsf{AEnc}(pk, S_J) = \mathsf{ASum}\left((\mathsf{AEnc}(t_j))_{j\in J}\right) = \mathsf{AEnc}\left(\sum_{j\in J} t_j\right)$$
然后使用 $\mathsf{ARefresh}$ 随机化密文并发送给 P_2。

输出 (P_2)

P_2 用私钥 sk 解密密文，得到交集和 S_J。

基于以上内容，我们定义两个类作为主体：

```
class Party1:
    def __init__(self, V, curve):
        self.V = V
        self.curve = curve
        self.k1 = random.randint(1, curve.field.n - 1)
        self.Z_tuples = None
        self.intersection_set = set()

    def round1_send(self):
        points = []
        for v in self.V:
            p = H_to_G(v, self.curve)
            p1 = self.k1 * p
            points.append(p1)
        random.shuffle(points)
        return points

    def round3_receive(self, encrypted_pairs):
        Z_set = set(self.Z_tuples)
        sum_enc = None
        for (point_k2, enc_t) in encrypted_pairs:
            val = self.k1 * point_k2
            val_tuple = point_to_tuple(val)
            if val_tuple in Z_set:
                self.intersection_set.add(val_tuple)
                if sum_enc is None:
                    sum_enc = enc_t
                else:
                    sum_enc = ECCElGamal.add_ciphertexts(sum_enc, enc_t)
        return sum_enc


class Party2:
    def __init__(self, W, curve):
        self.W = W
        self.curve = curve
        self.k2 = random.randint(1, curve.field.n - 1)
        self.elgamal = ECCElGamal(curve)

```

```python
    def round2_receive_and_send(self, points_k1):
        Z = []
        for p in points_k1:
            Z.append(self.k2 * p)
        random.shuffle(Z)

        pairs = []
        for (w, t) in self.W:
            p = H_to_G(w, self.curve)
            p_k2 = self.k2 * p
            enc_t = self.elgamal.encrypt(t)
            pairs.append((p_k2, enc_t))
        random.shuffle(pairs)
        return Z, pairs

    def round3_decrypt(self, sum_enc):
        return self.elgamal.decrypt(sum_enc)
```

由于这个协议是基于 DDH 的，所以我们定义相关的工具函数如下：

```python
# ------------------------------------
# 工具函数
# ------------------------------------

def H_to_G(msg, curve):
    """
    哈希字符串到曲线点
    """
    h = sha256(msg.encode()).digest()
    scalar = int.from_bytes(h, 'big') % curve.field.n
    point = scalar * curve.g
    return point

def point_to_tuple(point):
    return (point.x, point.y)

def tuple_to_point(tpl, curve):
    return curve.Point(tpl[0], tpl[1])

def encode_int_to_point(m, curve):
    """
    简单映射: m * G
    注意: m必须小于曲线阶
    """
```

```
    return m * curve.g

def decode_point_to_int(point, curve):
    """
    简单暴力搜索解码，只能小权重用
    """
    # 小权重时可用暴力搜索解码
    for i in range(10000):
        if i * curve.g == point:
            return i
    raise ValueError("无法解码该点为整数")

# ------------------------------------
# ECC-ElGamal 加密/解密
# ------------------------------------

class ECCElGamal:
    def __init__(self, curve):
        self.curve = curve
        self.priv = random.randint(1, curve.field.n - 1)
        self.pub = self.priv * curve.g

    def encrypt(self, m):
        """
        m是整数，先编码成点，再加密
        返回 (C1, C2)，两个椭圆曲线点
        """
        M = encode_int_to_point(m, self.curve)
        k = random.randint(1, self.curve.field.n - 1)
        C1 = k * self.curve.g
        C2 = M + k * self.pub
        return (C1, C2)

    def decrypt(self, C):
        """
        C = (C1, C2)
        解密得到点M，再解码成整数
        """
        C1, C2 = C
        S = self.priv * C1
        M = C2 - S
        m = decode_point_to_int(M, self.curve)
        return m
```

```
    @staticmethod
    def add_ciphertexts(C1, C2):
        """
        椭圆曲线上的点加法，实现同态加密相加
        """
        return (C1[0] + C2[0], C1[1] + C2[1])
```

在 main 函数中，我们定义相关参数：

```
curve = registry.get_curve('secp256r1')

V = ["alice", "bob", "carol", "dave"]
W = [("bob", 3), ("carol", 5), ("eve", 2), ("frank", 1)]

P1 = Party1(V, curve)
P2 = Party2(W, curve)

round1_msg = P1.round1_send()
Z_points, pairs = P2.round2_receive_and_send(round1_msg)
P1.Z_tuples = set(point_to_tuple(p) for p in Z_points)
sum_enc = P1.round3_receive(pairs)
intersection_sum = P2.round3_decrypt(sum_enc)

print(f"Intersection sum = {intersection_sum}")  # 预期：8
```

按照期望应当输出 8，我们查看结果：

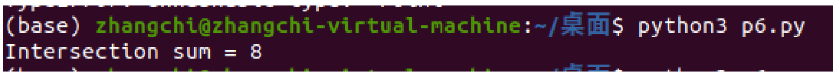

图 2 结果

可以看到结果符合预期。