一、实验要求

1. 学习掌握密文策略属性基加密(CP-ABE)算法原理。

2. 代码实现密文策略属性基加密(CP-ABE)算法。

3. 随机指定明文数据，能够输出访问策略 Access policy、属性集 Attribute Set、密文 Ciphertext、密钥 Private key 和解密结果 Decryption results。

二、实验设备

主机：Windows10

工具：IntelliJ Idea

三、实验原理

1. 访问控制机制

访问控制是指对用户合法使用资源的认证和控制。

2. 基本的属性基加密（如 FIBE 算法）

基本的属性基加密将密文和密钥都与一组属性关联起来，当密文与密钥之间至少有 d 个属性重合时，用户就可以解密密文。虽然这个策略对于生物识别的容错加密有一定的作用，但访问控制缺乏灵活性限制了它的应用。

3. 访问架构（策略）

访问结构（access structure）是安全系统研究的术语，系统的访问结构是指被授权的集合的结构。

4. 属性基加密系统的两种策略

KP-ABE（key policy attribute based encryption），即密钥策略属性基加密系统。密钥策略属性基加密系统，将策略嵌入到用户密钥中，属性嵌入到密文中，即密钥（私钥）对应于一个访问控制而密文对应于一个属性集合，当且仅当属性集合中的属性（主要为数据/文件属性）能够满足此访问结构才能解密得出最终明文。这种设计比较接近静态场景，此时密文用与其相关的属性加密存放在服务器上，当允许用户得到某些消息时，就分配一个特定的访问策略给用户，即控制“人可以访问哪类数据”。KP-ABE 的常见应用有付费视频网站、日志加密管理和审计、广播加密等。

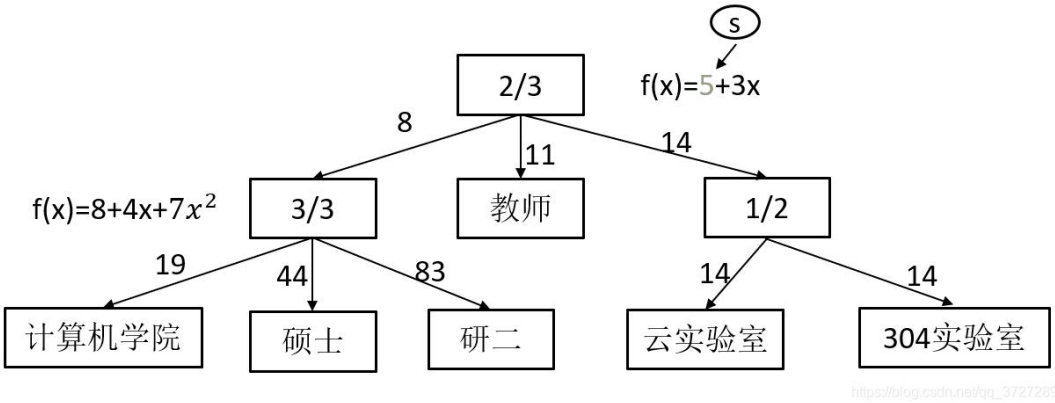
CP-ABE（ciphertext policy attribute based encryption），即密文策略属性基加密系统。密文策略属性基加密系统，是将策略嵌入到密文中，属性嵌入到用户密钥中，即密文对应于一个访问结构而密钥（私钥）对应于属性集合，当且仅当属性集合中的属性（主要为用户属性）能够满足此访问结构才能解密得出最终明文。这种设计比较接近于现实中的应用场景，可以假象每个用户根据自身条件或者属性从属性机构得到密钥，然后加密者来制定对消息的访问控制。这对这份数据做了一个粒度可以细化到属性级别的加密访问控制，即“有哪些属性的人可以访问数据”。CP-ABE 的常见应用有隐私数据共享等访问类应用，如云上的数据加密存储与细粒度共享等。

5. CP-ABE 算法原理

在 CP-ABE 方案中，访问树用于隐藏源数据的加密密钥。其叶子节点为数

据所有者设定的属性和属性值以及父节点传于此节点的秘密值，并对其加密处理，只有数据访问者拥有此属性方可解密出此节点的秘密值；非叶子节点为门限节点，数据访问者需满足此门限最低值方可解密此节点秘密值，例如门限为3/5，此节点有 5 个子节点，数据访问者需至少满足 3 个子节点才能解密出秘密值。

如下图所构造的访问树，能解密此访问树加密的源数据，数据访问者需满足的属性是：第一种：（“计算机学院” 且 “硕士” 且 “研二”）和 “教师”（此属性结合可能不存在，因为教师和研二不存在且关系），第二种：“教师” 和（“网络实验室” 或 “云实验室”），第三者：（“计算机学院” 且 “硕士” 且 “研二”）和（“网络实验室” 或 “云实验室”）；否则无法访问。



而构造访问树的过程基本原理为：在进行秘密值分发之前，加密者先随机选择秘密值 s（以 s=5）为例，然后，先从根节点构造多项式 $f(x)=ax^{(n-1)}+bx^{(n-2)}+\cdots+c$，其中 n 表示的为(t,n)门限的 t-1。如果为"AND"门，取值为其子树的数目-1，"OR"结构取值为 1 即可，常数项 c 设置为秘密值 s。从左到右依次编号，带入 1，2，3…来进行秘密值分发，第一层分发完成之后进入第二层进行秘密值分发，按照此规律完成加密过程。

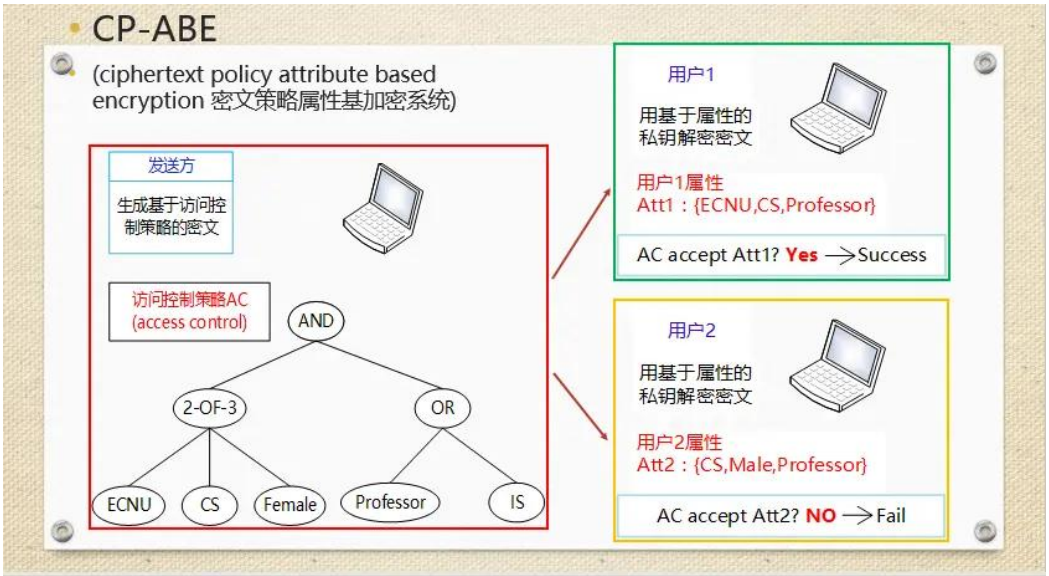
该例子中具体构造过程即：从根节点开始，其门限值为 2，孩子节点有 3 个，随机生成一个多项式，其最高次数为门限值少 1，故根节点的最高次数为 1，然后将常数项设置为秘密数（秘密数为需要秘密保存的数）；如此根节点随机的多项式为 f(x)=5+3x，秘密数为 5。此外，将根节点的孩子节点从左至右依次标记为 1,2,3,.....，将节点标记值代入 f(x)函数中，所得值（即生成新的秘密值）传给该标记的孩子节点秘密保存；故“3/3”节点（左边第一个节点）标记为 1，传给“3/3”节点的秘密值 f(1)=5+3*1=8，中间“教师”节点（中间节点）标记为 2，传给“教师”节点的秘密值 f(2)=5+3*2=11，“1/2”节点（右边节点）标记为 3，传给“1/2”节点的秘密值为 f(3)=5+3*3=14。

“3/3”节点和“1/2”节点在接收到父节点传来的值后，按照上述方式生成随机多项式，将常数项设置为父节点传来的值，此外也按照上述方式生成新的秘密值并将它传给子节点，数据如图所示（对于非叶子节点，都按照此方式进行）。对于叶子节点，在接受到父节点的秘密值后，用此叶子节点的属性对秘密值进行加密处理。

至于解密：数据访问者需满足访问树方可解密出访问树的秘密值。对于叶子节点，在数据访问者属性集中寻找出和此节点属性与属性值一致的属性，用找出的属性解密出此节点的秘密值，当然不能完全解密出，他是秘密值和加密时对此属性设置的加密值的乘积。解密出叶子节点后，开始解密其父节点（非叶子节点），在解密出叶子节点后，即可得到多对值；如在上述访问树的“3/3”节点，其孩子节点解密出三个值 19，44，83（推理过程忽略随机数），在生成这三个数时，f(1)=19，f(2)=44，f(3)=83，其中 f(x)=8+4x+7x^2（解密时并不知

道此多项式，只知道后面的三个点），因此在 f(x)上有三个点是（1，19），（2，44），（3，83）；因为此节点存储的秘密值是多项式的常数项，即 f(0)=秘密值，故我们需要根据这三个点得到 0 所对应的值是多少，根据拉格朗日公式就能求出 0 所对应的值，即解密出秘密值；对于非叶子节点均可按照上述方式解密出秘密值，在根节点处解密出整个树所隐藏的秘密值（是秘密值和加密时对此属性设置的加密值的乘积，所有节点解密出的秘密都存在一个随机数，解密出的值都是随机数和秘密值的乘积，只是讲述过程中为了方便忽略了随机数，最后根节点解密出的随机数在最后的解密过程中会抵消掉）。

另外需要说明的是“3/3”节点是一个 3 个孩子节点、门限值为 3 的节点，相当于且关系，数据访问者需解密出所有的孩子节点方可用拉格朗日公式解密出常数项（即秘密值）；“1/2”节点是一个 2 个孩子节点、门限值为 1 的节点，故为或关系，数据访问者只满足其中一个孩子节点即可解密出秘密值（秘密值就是孩子节点解密出的秘密值，因为根据随机多项式生成规则，多项式的最高次数为门限值少 1，门限值为 1，故最高次数为 0，即常数项）；若如“2/3”这样的节点，既非且关系，也非或关系，3 个孩子节点，门限值为 2，若需解密出此节点的秘密值，数据访问者需解密出三个孩子节点中的两个节点即可解密出孩子节点。



6. CP-ABE 具体算法流程

环境基础：G0 是一个素数阶 p 的乘法循环群，g 是群的一个生成元，e: G0×G0→G1 表示双线性映射，k 是一个能决定群大小的安全参数，同时定义拉格朗日系数：$\Delta_{i,S}(x)=\prod_{j\epsilon S,j\neq i}\frac{x-j}{i-j}$，其中 i∈Zp（整数域上的加法循环群），S 是属于 Zp 的成员集（属性集）。采用哈希函数 H:{0,1}*→G0，将其建模为随机预言机，将二进制字符串的任一属性映射到随机组元素。

① setup 初始化

（1）生成 pairing 相关公共参数<e, g, G0, G1, Zp>。

（2）选取两个随机数 α , β ∈Zp，计算g^{α} ，作为系统主密钥 MK 之一。

（3）并计算$h=g^{\beta}$，$f=g^{1/\beta}$，$e(g,g)^{\alpha}$ 作为公钥 PK。

（4）系统主密钥 MK = { α , β , g^{α} }，公钥 PK = { $h=g^{\beta}$,$f=g^{1/\beta}$,$e(g,g)^{\alpha}$,g, G0, G1, Zp }。

② keygen 密钥生成

（1）选取一个随机数 r∈Zp。

（2）为每个用户属性 j∈S，随机选取生成一个 $r_j \in Zp$。

（3）计算$D = g^{(\alpha+r)/\beta}$ ，$D_j = g^r H(j)^{r_j}$ ，$D'_j = g^{r_j}$ 。

（4）用户私钥 SK={ $D = g^{(\alpha+r)/\beta}$,S, $D_j = g^r H(j)^{r_j}$,$D'_j = g^{r_j}$,S }。

③ encrypt 加密（与访问控制树相关联）

（1）对于访问树 T，根节点记为 R，每个非叶子节点 x 都有一个门限值（节点阈值）k_x，而每个叶子节点都对应一个属性。

（2）从根节点 R 开始，自上而下为树中的每个节点 x（包括叶子）选择一个 $d_x=k_x-1$ 阶多项式 q(x)。

（3）选取一个随机数 s∈Zp，对于根节点 R 则设置为 $q_R(0)=s$（即拉格朗日差值法中的 a_0），然后 d_R 次多项式 q_R 的其它点完全随机的选取。

（4）其它内部节点 x 的多项式要满足 $q_x(0)=q_{Parent(x)}(index(x))$，其它 d_x 个点随机选取来完全定义 q_x。（index 为该节点的索引值，ID 为该节点的唯一性标识。）

（5）令 Y 为 T 中的叶子节点集合，每个叶子节点 y∈Y，att(y)为该叶子节点对应的属性值。

（6）对于明文 M，计算 $C^{\sim} = M \cdot e(g,g)^{\alpha s}$ 。

（7）计算$C = h^s$。

（8）对于每个叶子节点 y∈Y，计算 $C_y = g^{q_y(0)}$，$C'_y = H(att(y))^{q_y(0)}$。

（9）密文为 CT = {T, $C^{\sim} = M \cdot e(g,g)^{\alpha s}$, $C = h^s$,(任意 y∈Y) $C_y = g^{q_y(0)}$, $C'_y = H(att(y))^{q_y(0)}$}。

④ decrypt 解密（递归）

（1）如果访问树存在，则可以进行递归解密操作。

（2）如果 x 是叶子节点，令 i = att(x) ，并且如果 i∈S，那么计算解密节点：

$A = DecryptNode(CT, SK, x) = \frac{e(D_i, C_x)}{e(D'_i, C'_x)} = \frac{e(g^r H(i)^{r_i}, h^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} = e(g,g)^{rq_x(0)} = e(g,g)^{rs}$ 。

反之，如果 i∉ S，那么定义解密节点为空。

（3）如果 x 是非叶子节点，则递归进行如下操作计算：

(i) 对 x 的所有孩子节点 z，索引值为 i=index(z)，令 S_x 为一个由 x 的任意 k_x 个子节点组成的集合，这些节点要满足对这些孩子节点进行递归解密结果不为空，也就是说用户属性与访问控制树的所有叶子节点属性重合数量要满足每个内部节点的门限值。如果不存在这样的集合，则解密输出为空，即解密失败，否则进行下一步运算。

(ii) 计算 $F_z^{\Delta_{i,S'_x}(0)} = e(D_i, C_x)^{\Delta_{i,S'_x}(0)} = e(g,g)^{rq_x(i)\Delta_{i,S'_x}(0)}$ 【$i = index(z)$，S'_x = {index(z): z ∈ S_x}】。

(iii) 连乘运算 $A = DecryptNode(CT, SK, x) = F_x = \prod_{i \in Index(x), z \in S_x} F_z^{\Delta_{i,S'_x}(0)} = e(g,g)^{r\sum_{i \in Index(x)} q_x(i)\Delta_{i,S'_x}(0)} = e(g,g)^{rq_x(0)} = e(g,g)^{rs}$。

（4）除法运算$\frac{C^{\sim}}{e(C,D)/A} = \frac{C^{\sim}}{e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}} = \frac{\mathrm{M}\cdot \mathrm{e(g,g)}^{\alpha s}}{\mathrm{e(g,g)}^{\alpha s}} = M$，得到明文消息 M。

四、实验流程

【4.1】备注

本实验将复现密文策略属性基加密(CP-ABE)算法，实现过程中有几点备注如下：

1. 实现过程中所有属性用中文字符串表示，通过 JPBC 库里的方法将明文

字符串编码转换为字节数组，再由该字节数组来实例化一个 GT 群上的点 Element 类，进而继续下一步计算。

2. 多项式求值和拉格朗日插值在群 Zp（在 JPBC 库中表示为 Zr）上进行，因此相应的 int 值在计算前要转换为 Zr Element。

3. 对于重复使用的值一定要记得使用 getImmutable()或者 duplicate()。尤其是在 for 循环中。

4. 使用从文件 a.properties 中读取相关参数的方法完成对双线性群（即椭圆曲线）的初始化即在 JPBC 中对 Pairing 对象初始化。生成的公钥 pk、主密钥 msk、用户私钥 sk、密文 ct 集合也都放入对应的文件中。

5. 本实验中的明文消息选用了 JPBC 库里的方法通过明文编码将明文字符串转换为字节数组，再由该字节数组来实例化一个 GT 群上的点 Element 类，进而继续下一步加密，这样也能够正确解密出明文消息串。

【4.2】具体实现（部分关键代码）

1. setup 初始化

```
27      public static void test1() {
28          //1.生成系统密钥  包含公钥-私钥
29          SystemKey systemKey = SystemKey.build();
30          //系统主私钥MK={beta,alpha,g^Alpha}
31          //系统公钥PK={h=g^Beta,f=g^(1/Beta),e(g,g)^Alpha,g,G0, G1,Zp}
```

```
    1 usage
39  public static MasterPrivateKey build(PairingParameter parameter){
40      MasterPrivateKey masterPrivateKey = new MasterPrivateKey(parameter);  //根据pairing实例创建系统主私钥
41      masterPrivateKey.setBeta(parameter.getZr().newRandomElement().getImmutable());  //从Zr群中随机生成Beta
42      masterPrivateKey.setAlpha(parameter.getZr().newRandomElement().getImmutable());  //从Zr群中随机生成Alpha
43      masterPrivateKey.setG_alpha((parameter.getGenerator().powZn(masterPrivateKey.getAlpha())).getImmutable());
44      //计算g^Alpha
45      return masterPrivateKey;
46  }
```

```
1 usage
public static PublicKey build(PairingParameter parameter, MasterPrivateKey msk) {
    PublicKey publicKey = new PublicKey(parameter);  //根据pairing实例创建系统公钥
    publicKey.setH(parameter.getGenerator().powZn(msk.getBeta()).getImmutable());  //根据系统密钥MSK对应参数，计算h=g^Beta
    publicKey.setF(parameter.getGenerator().powZn(msk.getBeta().invert()).getImmutable());  //根据系统密钥MSK对应参数，计算f=g^(1/Beta)
    publicKey.setEgg_a((parameter.getPairing().pairing(parameter.getGenerator(), parameter.getGenerator()).mulZn(msk.getAlpha())).getImmutal
    //根据系统密钥MSK对应参数，计算e(g,g)^Alpha
    return publicKey;
}
```

2. keygen 密钥生成

```
//3.生成用户私钥
UserPrivateKey userPrivateKey = cpAneEngine.keyGen(systemKey.getMasterPrivateKey(), attributes);
//用户私钥SK={D=g^[(Alpha+r)/Beta],Dj=(g^r)*[H(j)^rj],Dj'=g^rj,j∈S}
```

```
1 usage
public static UserPrivateKey build(MasterPrivateKey masterPrivateKey, List<Attribute> attributes){
    UserPrivateKey userPrivateKey = new UserPrivateKey();
    userPrivateKey.setPairingParameter(masterPrivateKey.getPairingParameter());  //根据系统主密钥，生成用户私钥
    userPrivateKey.setUserAttributes(attributes);  //获取用户属性集
    Element r = masterPrivateKey.getPairingParameter().getZr().newRandomElement().getImmutable();  //从Zr群中随机生成r
    Element alpha = masterPrivateKey.getAlpha();  //根据系统主密钥，获取Alpha
    Element beta = masterPrivateKey.getBeta();  //根据系统主密钥，获取Beta
    Element g = masterPrivateKey.getPairingParameter().getGenerator();  //根据系统主密钥，获取生成元g
    Element D = g.powZn((alpha.add(r)).div(beta)).getImmutable();  //计算D=g^[(Alpha+r)/Beta]
    userPrivateKey.setD(D);
    //遍历每个属性j∈S
    for (Attribute attribute : attributes){
        Element r_j = masterPrivateKey.getPairingParameter().getZr().newRandomElement().getImmutable();  //从Zr群中随机生成rj
        Element D_j = g.powZn(r).mul(masterPrivateKey.hash(attribute.getAttributeValue()).powZn(r_j)).getImmutable();
        //计算Dj=(g^r)*[H(j)^rj]
        Element D_j_pie = g.powZn(r_j).getImmutable();  //计算Dj'=g^rj
        userPrivateKey.putDjPie(attribute,D_j_pie);
        userPrivateKey.putDj(attribute,D_j);
    }
    return userPrivateKey;
}
```

3. encrypt 加密

```
//6.加密
CipherText cipherText = cpAneEngine.encrypt(systemKey.getPublicKey(), plainText, accessTree);
//密文CT={ T, C~ = M * e(g,g)^(Alpha*s), C = h^s, Cy = g^qy(0) = g^qi(0), Cy' = [H(att(y))]^qy(0) }
System.out.println("cipherText : " + cipherText);
```

```java
public CipherText encrypt(PublicKey pk, PlainText plainText, AccessTree accessTree) {
    AccessTreeNode root = accessTree.getRoot();

    //根节点的秘密数
    Element s = getRandomElementInZr(pk);  //从Zr加法循环群中依据公钥参数随机生成一个s
    root.setSecretNumber(s);  //令根节点的秘密数为qr(0)=s

    CipherText cipherText = new CipherText();  //新建密文类

    //1.设置密文第一部分

    Element c_ware = (plainText.getMessageValue().mul(pk.getEgg_a().powZn(s).getImmutable())).getImmutable();
    //C~ = M * e(g,g)^(Alpha*s) 【从公钥里获取e(g,g)^Alpha】
    cipherText.setC_wave(c_ware);

    //2.设置密文第二部分
    Element c = pk.getH().powZn(s).getImmutable();
    //C = h^s 【从公钥里获取 h=g^beta，也就是说 C = g^(Beta*s)】
    cipherText.setC(c);

    //3.递归设置子节点的秘密值qx(y)，Cy = g^qy(0) = g^qi(0)，Cy' = [H(att(y))]^qy(0)
    compute(root, pk, cipherText);

    //设置访问树T
    cipherText.setAccessTree(accessTree);
    return cipherText;
}
```

4. decrypt 解密

```java
//7.解密
String decryptStr = cpAneEngine.decryptToStr(systemKey.getPublicKey(), userPrivateKey, cipherText);
//递归解密
//解密最终计算
//输入1：系统公钥PK={h=g^Beta,f=g^(1/Beta),e(g,g)^Alpha,g,G0，G1,Zp}
//输入2：用户私钥SK={D=g^[(Alpha+r)/Beta],Dj=(g^r)*[H(j)^rj],Dj'=g^rj,j∈S}
//输入3：密文CT={ T，C~ = M * e(g,g)^(Alpha*s)，C = h^s，Cy = g^qy(0) = g^qi(0)，Cy' = [H(att(y))]^qy(0) }
//拉格朗日差值算法或直接计算，递归恢复每个解密节点的秘密值，获得 A = e(g,g)^(r*qx(0)) = e(g,g)^(r*s)
//计算 M = C~/[e(C,D)/A] = M * e(g,g)^(Alpha*s) / e(g,g)^(Alpha*s)
//输出：M
System.out.println("decryptStr : " + decryptStr);
```

```java
//解密最终计算
//输入1：系统公钥PK={h=g^Beta,f=g^(1/Beta),e(g,g)^Alpha,g,G0，G1,Zp}
//输入2：用户私钥SK={D=g^[(Alpha+r)/Beta],Dj=(g^r)*[H(j)^rj],Dj'=g^rj,j∈S}
//输入3：密文CT={ T，C~ = M * e(g,g)^(Alpha*s)，C = h^s，Cy = g^qy(0) = g^qi(0)，Cy' = [H(att(y))]^qy(0) }
//拉格朗日差值算法或直接计算，递归恢复每个解密节点的秘密值，获得 A = e(g,g)^(r*qx(0)) = e(g,g)^(r*s)
//输出：M (Element类)
1 usage
public Element decrypt(PublicKey publicKey, UserPrivateKey userPrivateKey, CipherText cipherText) {
    //拉格朗日差值算法或直接计算，递归恢复每个解密节点的秘密值，获得 A = e(g,g)^(r*qx(0)) = e(g,g)^(r*s)
    Element decryptNode = decryptNode(publicKey, userPrivateKey, cipherText, cipherText.getAccessTree().getRoot(), userPrivateKey.getUserAtt
    if (decryptNode != null) {
        Element D = userPrivateKey.getD();  //从用户私钥中获取D=g^[(Alpha+r)/Beta]
        Element C = cipherText.getC();  //从密文中获取C = h^s
        Element c_wave = cipherText.getC_wave();  //从密文中获取C~ = M * e(g,g)^(Alpha*s)
        Pairing pairing = publicKey.getPairingParameter().getPairing();  //从公钥中获取Pairing实例各参数
        return c_wave.div(pairing.pairing(C, D).div(decryptNode));  //计算 M = C~/[e(C,D)/A] = M * e(g,g)^(Alpha*s) / e(g,g)^(Alpha*s)
    }
    return null;
}
```

5. 主函数

```java
public static AccessTree getAccessTree(PublicKey publicKey) {
    AccessTreeBuildModel[] accessTreeBuildModels = new AccessTreeBuildModel[7];  //访问控制树大小为7
    //根节点ID必须为1
    accessTreeBuildModels[0] = AccessTreeBuildModel.innerAccessTreeBuildModel( id: 1,  threshold: 2,  index: 1,  parentId: -1);  //根节点I
    accessTreeBuildModels[1] = AccessTreeBuildModel.leafAccessTreeBuildModel( id: 2,  index: 1,  attributeName: "学生",  parentId: 1);  //
    accessTreeBuildModels[2] = AccessTreeBuildModel.leafAccessTreeBuildModel( id: 3,  index: 2,  attributeName: "老师",  parentId: 1);  //
    accessTreeBuildModels[3] = AccessTreeBuildModel.leafAccessTreeBuildModel( id: 4,  index: 3,  attributeName: "硕士",  parentId: 1);  //
    accessTreeBuildModels[4] = AccessTreeBuildModel.innerAccessTreeBuildModel( id: 5,  threshold: 1,  index: 4,  parentId: 1);     //ID为5
    accessTreeBuildModels[5] = AccessTreeBuildModel.leafAccessTreeBuildModel( id: 6,  index: 1,  attributeName: "二班",  parentId: 5);  //
    accessTreeBuildModels[6] = AccessTreeBuildModel.leafAccessTreeBuildModel( id: 7,  index: 2,  attributeName: "护士",  parentId: 5);  //
    return AccessTree.build(publicKey, accessTreeBuildModels);
}
```

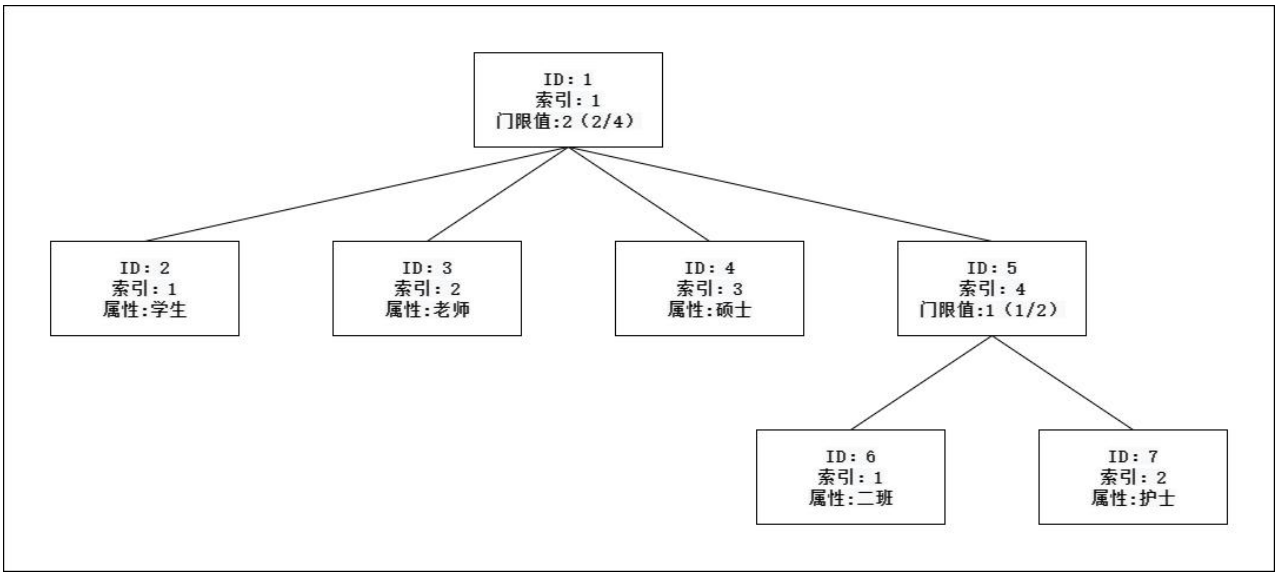
```java
1 usage
public static void test1() {
    //1.生成系统密钥  包含公钥-私钥
    SystemKey systemKey = SystemKey.build();
    //系统主私钥MK={beta,alpha,g^Alpha}
    //系统公钥PK={h=g^Beta,f=g^(1/Beta),e(g,g)^Alpha,g,G0, G1,Zp}

    //2.设置用户属性
    List<Attribute> attributes = Arrays.asList(
            // new Attribute("学生", systemKey.getPublicKey()),
             //new Attribute("老师", systemKey.getPublicKey()),
             new Attribute( attributeName: "硕士", systemKey.getPublicKey())//,  //新建用户属性"硕士"及对应的Element属性元素
             //new Attribute("护士", systemKey.getPublicKey())
            // new Attribute("二班", systemKey.getPublicKey())
    );

    //新建主算法引擎
    CpAneEngine cpAneEngine = new CpAneEngine();

    //3.生成用户私钥
    UserPrivateKey userPrivateKey = cpAneEngine.keyGen(systemKey.getMasterPrivateKey(), attributes);
    //用户私钥SK={D=g^[(Alpha+r)/Beta],Dj=(g^r)*[H(j)^rj],Dj'=g^rj,j∈S}
```

```java
    //4.明文
    String plainTextStr = "你好, CP - ABE, 我是3201603102";
    PlainText plainText = new PlainText(plainTextStr, systemKey.getPublicKey());  //将明文字符串转换为字节数组，并使用G1群的Element元素来表示
    System.out.println("plainTextStr : " + plainTextStr);

    //5.构建访问控制树
    AccessTree accessTree = getAccessTree(systemKey.getPublicKey());

    //6.加密
    CipherText cipherText = cpAneEngine.encrypt(systemKey.getPublicKey(), plainText, accessTree);
    //密文CT={ T, C~ = M * e(g,g)^(Alpha*s), C = h^s, Cy = g^qy(0) = g^qi(0), Cy' = [H(att(y))]^qy(0) }
    System.out.println("cipherText : " + cipherText);

    //7.解密
    String decryptStr = cpAneEngine.decryptToStr(systemKey.getPublicKey(), userPrivateKey, cipherText);
    //递归解密
    //解密最终计算
    //输入1：系统公钥PK={h=g^Beta,f=g^(1/Beta),e(g,g)^Alpha,g,G0, G1,Zp}
    //输入2：用户私钥SK={D=g^[(Alpha+r)/Beta],Dj=(g^r)*[H(j)^rj],Dj'=g^rj,j∈S}
    //输入3：密文CT={ T, C~ = M * e(g,g)^(Alpha*s), C = h^s, Cy = g^qy(0) = g^qi(0), Cy' = [H(att(y))]^qy(0) }
    //拉格朗日差值算法或直接计算，递归恢复每个解密节点的秘密值，获得 A = e(g,g)^(r*qx(0)) = e(g,g)^(r*s)
    //计算 M = C~/[e(C,D)/A] = M * e(g,g)^(Alpha*s) / e(g,g)^(Alpha*s)
    //输出：M
    System.out.println("decryptStr : " + decryptStr);
}
```

6. 运行检验

① 测试数据

访问控制树如图：



明文消息串为“你好，CP - ABE，我是 3201603102”。

② 用户属性满足访问控制树属性要求的情况

测试数据：用户属性 ＝{“硕士”，“护士”}

运行得到：

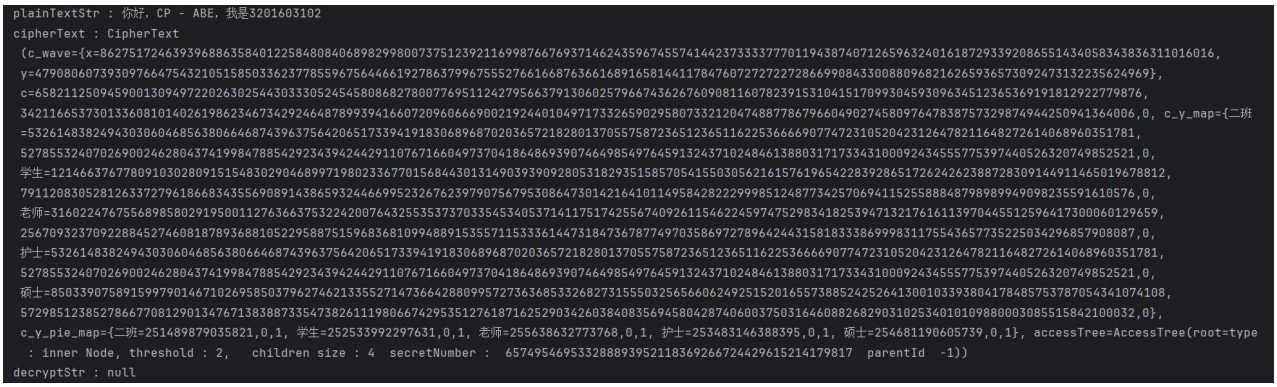
```
plainTextStr : 你好，CP - ABE，我是3201603102
cipherText : CipherText
 (c_wave={x=472187164645007362462249762400582930249182044043021278473978158747709558700584034104095510904420003793290254798125542794132916312145438776647077127892 50,
 y=796728669714384124877562447374778697496430647151896337131406978016994026348109149379966820152551283898203164449113014403139520975154121592847966730639055 1},
 c=368425048841109422963952522437389614290429092743335138933601031957522674570249068400936349968546911213019982787545926942434249591432423844842142499849141 8,
 308386900789910851016550787370088817208450092358251005876376852320262118210221278447211806293899127336276013202492192333927972486436337456450136298106612 2,0, c_y_map={硕士
 =335376113152945453506999037155381790771144195533821003003473974016822632490139979940616993099818124188380365959646169665040399941282506557147254937345 2014,
 215184506113458564200833619392740448333444954850173176569711800875283836510840053802850219663315267518776499316297135955514069591563498204602050894607464,0,
 学生=434881693000841700166994750744129013465038708140398873895255086814122927615391998422312097810764525213434372141775430427687873601428344555719677977459 0728,
 143148733321710038134062355054360439455661720290538014737748145634861384085137246579588128351923304226941545386831715280026774777562853364068396351842420,0,
 老师=732204040897291826185893095104484720259857290729373120185165938728295914938859295549893823849159831575379448385563363672856705254758882051381440000748 4831,
 438063457356665671510209305377771781307821671005262166400776604236020817483040188312549935351344583363241430157452031019138254115626958462587271205786244 32,0,
 护士=420599848261779490482953856920716822375166055891942659709063066303253185631253405481684182230710879591063239637661941101794862608415866703311003514469838,
 821331114102187589588831622079911372632093915148233379831754363256779612437740417275241339394321138524775251589279179791883561356339593849424046093861471 0,0,
 二班=420599848261779490482953856920716822375166055891942659709063066303253185631253405481684182230710879591063239637661941101794862608415866703311003514469838,
 821331114102187589588831622079911372632093915148233379831754363256779612437740417275241339394321138524775251589279179791883561356339593849424046093861471 0,0},
 c_y_pie_map={硕士=254681190605739,0,1, 学生=252533992297631,0,1, 老师=255638632773768,0,1, 护士=253483146388395,0,1, 二班=251489879035821,0,1}, accessTree=AccessTree(root=type
  : inner Node, threshold : 2,  children size : 4  secretNumber :  37034422551196942360423917645641581731313058716 8  parentId  -1))
decryptStr : 你好，CP - ABE，我是3201603102
```

能够成功解密得到明文消息串。

③ 明文属性不满足访问控制树属性要求的情况

测试数据：用户属性 ＝{“硕士”}

运行得到：



无法正确解密得到明文消息。

五、实验结论

在基于密钥策略属性的加密 （KP-ABE） 系统中，密文由发送方使用一组描述性属性进行标记，而用户的私钥由受信任的属性颁发机构捕获策略（也称为访问结构），该策略指定密钥可以解密哪种类型的密文。KP-ABE 计划适用于有关于谁可以阅读特定文件的规则的结构化组织。KP-ABE 的典型应用包括安全取证分析和目标广播。例如，在安全取证分析系统中，可以使用用户名、用户操作的日期和时间以及用户操作修改或访问的数据类型等属性对审核日志条目进行注释。而负责某些调查的取证分析师将获得与特定访问结构相关的私钥。私钥只会打开其属性满足与私钥关联的访问策略的审核日志记录。

在密文策略基于属性的加密 （CP-ABE） 系统中，当发送方加密消息时，他们会根据密文中属性的访问结构指定特定的访问策略，说明哪种接收方能够解密密文。用户拥有属性集，并从属性权限处获取相应的秘密属性密钥。如果此类用户的属性满足与密文关联的访问策略，则该用户可以解密密文。因此，CP-ABE 机制在概念上更接近于传统的基于角色的访问控制方法。

换句话说，KP-ABE （Key Policy） 的加密者，用一堆 attributes（属性）去加密文件，然后有个中间人会订立 policy（访问策略），再分发具备 policy 的 private keys（密钥）给不同人。例如加密者用 “pay_user” 这个 attribute 去加密，然后 policy 可以是 {pay_user， free_user， tom...} 等等，任由中间人决定。加密者只知道符合 attributes 的人可以解密，但不能预知解密者的全部 policy + attributes 是什么。KP-ABE 密文中會有 attributes（属性），所以如果解密者见到密文，而手中没有符合该 attributes 的 private key 的话，那么就无法解密获得明文。这种加密法，通常在付费服务等情况中使用。

而 CP-ABE 则不同。加密者定义了整个 access structure （policy + attributes），中间人的作用只是基于这个 policy，分发不同 private keys 给不同人。情况有点像建立一棵 private key tree，不同 matching 會有不同 private keys。换句话说，加密者基本上可以预知到，有什么种类的人，而且当中有谁可以解密文件。CP-ABE 密文 （Cipher Text） 中會有整个 access structure，所以如果解密者见到密文，而手中没有符合该 structure + attributes 的 private key 的话，那么就无法解密获得明文。这种加密法，通常在文件分享等情况中使用。

本次实验记录的 Github 地址：https://github.com/YTR1020/CP-ABE/tree/main

参考文献：

[1]https://blog.csdn.net/m0_50299484/article/details/117934277.

[2]https://blog.csdn.net/qq_37272891/article/details/107008073?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-0&spm=1001.2101.3001.4242.

[3]https://www.jianshu.com/p/8d8cf34a9aa0.

[4]https://blog.csdn.net/jingzi123456789/article/details/104783728.

[5]https://cat.chrizchow.com/2019/11/kp-abe-vs-cp-abe.html.

[6]Changji Wang and Jianfa Luo,“An Efficient Key-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length”,Mathematical Problems in Engineering,vol. 2013, Article ID 810969,7 pages, 2013.https://doi.org/10.1155/2013/810969.

[7]https://eprint.iacr.org/2006/309.pdf.

[8]https://hal.archives-ouvertes.fr/hal-01788815/document.

[9]https://asecuritysite.com/public/cpabe.pdf.

[10]https://github.com/lengnu/CP-ABE.