一、实验要求

1. 学习掌握基于模糊身份的加密(FIBE)算法原理。

2. 代码实现基于模糊身份的加密(FIBE)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. 双线性映射

设 G1，GT 是素数阶 p 的群，设 g 是 G1 的生成元（即本原根，或称为阶），即 G1 为由 p 生成的循环加法群，0 为 G1 的单位元。我们说 G1 有一个允许的双线性映射即 e: G1×G1→GT，则 GT 为对称双线性群，即具有相同阶 p 的循环乘法群，1 为 GT 的单位元。该映射当且仅当以下两个条件成立：① 映射是双线性的；② 对于所有的 ab，我们有 e（g^a, g^b）=e（g, g）^{ab}。该映射具备非退化性则需满足 e（g, g）≠1，即无法映射到 GT 的幺元。

本算法实现中使用到对称双线性映射即 G1×G1→GT。由于双线性群现在的构造是基于椭圆曲线的，而椭圆曲线上的元素是由坐标(x, y)表示的，所以我们将 G1 的结果输出到 Java 的控制台，得到的是一个坐标。而 GT 是一个普通的 Zn 群，所以其元素的表示是一个数。

2. JPBC 库

PBC 库（pairing-based cryptography library）是斯坦福大学研究人员开发的一个免费可移植 C 语言库。它通过提供一个抽象的接口，使程序设计人员可以不必考虑具体的数学细节，甚至不必考虑椭圆曲线和数论的相关知识就可以实现基于配对的密码体制。JPBC 库（Java Pairing-Based Cryptography Library）是对 PBC 库的 Java 封装，常用于基于配对的密码学算法仿真程序编写中。

JPBC 库共提供四个循环群，其中 G1，G2，GT 均为阶为 p 的乘法循环群，而 Zp 为整数域上的加法循环群。乘法循环群上的点是 z 值为 0 的椭圆曲线上的点，而整数循环群上的点是数，二者均可抽象为 Element 数据类型并用于仿真中。G1，G2，GT 中元素的模幂运算、倍乘运算以及相互之间的加法运算，运算结果均为对应群上的元素，Zp 中元素的加减乘除运算以及乘方运算，运算结果为整数循环群上的元素。

需要注意的是，现在的密码学相关论文中，习惯将 G1，G2 设置为乘法循环群。但是基于椭圆曲线的双线性群构造中，G1，G2 是加法循环群。所以在 2005 年以前的论文中，双线性群一般写成加法群的形式。JPBC 库中将 G1，G2 表示成了乘法循环群，因此在加法循环群形式方案的仿真过程中，应特别注意将加法群改写为乘法群的写法再完成进一步仿真。由于加法群中的加法运算对应乘法群中的乘法运算，减法运算对应除法运算（即求逆元），乘法运算对应幂指数运算，而除法运算对应对数运算。故改写过程需要结合以上运算法则。

双线性群（即椭圆曲线）的初始化在 JPBC 中表现为对 Pairing 对象的初始化。JPBC 库支持 A、A1、D、E、F、G 六种椭圆曲线，对比如下。我们可以

通过代码动态产生和从文件中读取相关参数这两种方法完成上述初始化过程。

Type	Base field size (bits)	k	Dlog security (bits)
A	512	2	1024
D	n	6	$6n$
E	1024	1	1024
F	160	12	1920
G	n	10	$10n$

当确定椭圆曲线参数后重复调用 getG1()，newElement()和 newRandomElement()方法，可以得到使用 PairingFactory.getPairing(filename)函数导入特定参数的椭圆曲线后，每次调用 getG1()函数生成的循环群都是相同的，故可以通过保存椭圆曲线参数至 xxx.properties 文件并导入这一操作实现循环群的保存。对于群 G1，每次调用 G1.newElement()函数生成的生成元 g 都是相同的。然而调用 G1.newRandomElement()函数随机获取的群上元素则是不同的。

3. 拉格朗日差值算法

给定 n 个点$(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1},y_{n-1})$，可以决定一个 n- 1 次多项式：

$$P_{n-1}(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$$

分别将这 n 个点代入 P(x)中可以得到一组多项式，用矩阵乘法可得：

$$\begin{bmatrix} 1 & x_0 & \ldots & x_0^{n-1} \\ 1 & \ldots & \ldots & \ldots \\ 1 & x_{n-2} & \ldots & x_{n-2}^{n-1} \\ 1 & x_{n-1} & \ldots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \ldots \\ a_{n-2} \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ \ldots \\ y_{n-2} \\ y_{n-1} \end{bmatrix}$$

该组多项式可以表示为：

$$L(x,n) = q(x) = \sum_{i=0}^{n-1} y_i \delta_i(x) = \sum_{i=0}^{n-1} y_i LB_i(x)$$

其中，拉格朗日因子：

$$LB_i(x) = \delta_i(x) = \prod_{j=0, j\neq i}^{n-1} \frac{x - x_j}{x_i - x_j}$$

4. Shamir 秘密共享

即将秘密 s 分割后共享给 n 个人，至少 k 个人组合后才可以恢复秘密 s。该分享方案具体内容为：

任意取 k-1 个随机数，构造如下 k-1 次随机多项式 q(x)：

$$q(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$$

并使得 $q(0) = a_0 = s$ ，取任意 n 个数 x_n 分别代入多项式，计算 $(x_1, q(x_1))$，$(x_2, q(x_2))$，...，$(x_n, q(x_n))$，将对应的 $q(x_i)$ 分享给第 i 个用户。（所有运算均在有限域中进行。）

k 个用户通过其秘密分片通过拉格朗日插值算法求解多项式系数可以恢复 q(x)，进而计算出 q(0)，也就是 $s = q(0) = a_0$。

5. FIBE 算法原理

① setup 初始化

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

（2）确定属性全集 U 为整数集合{1, 2, ... , |U|}，以及系统门限值 d。

（3）针对每个属性 i 选择随机数 $t_i \in Zp$ 作为主密钥组件，计算 $T_i = g^{ti}$ 作为对应的公钥组件。

（4）选取随机数 $y \in Zp$，并计算 $Y = e(g,g)^y$ 。

（5）最终，系统主密钥 $msk = < t_1, t_2, ..., t_{|U|}, y >$，公钥 $pk = < T_1, T_2, ..., T_{|U|}, Y >$。

② keygen 密钥生成

（1）随机选择一个 d-1 次多项式 q(x)，使得 q(0) = y（即拉格朗日差值法中的 a_0）。

（2）针对用户属性集合 S 中的每个属性 i，计算 q(i)，进一步计算 $D_i = g^{\frac{q(i)}{t_i}}$。

（3）用户私钥为 $sk = \{D_i\}$（$i \in S$）.

③ encrypt 加密

（1）选取随机数 $s \in Zp$，针对明文消息 $M \in GT$，计算 $E' = M \cdot Y^s = M \cdot e(g,g)^{ys}$。

（2）针对明文属性集合 W 中的每个属性 i，计算$E_i = T_i^{\,s}$。

（3）密文为 $ct = < E', \{E_i\}(i \in W) >$。

④ decrypt 解密

（1）如果用户属性集合 S 和明文属性 W 重合属性个数不小于 d，可按如下方法继续解密。

（2）从所有重合属性中选取 d 个构成属性集合 I。

（3）针对 I 中的每个属性 i，计算$P_i = e(E_i, D_i)^{\delta_i(0)} = e(g,g)^{sq(i)\delta_i(0)}$，其中$\delta_i(0)$是拉格朗日因子。

（4）$\prod_{i \in I} P_i = e(g,g)^{s \sum_{i \in I} q(i)\delta_i(0)} = e(g,g)^{sy}$。

（5）$\frac{E'}{\prod_{i \in I} P_i} = M$。

四、实验流程

本实验将复现基于模糊身份的加密(FIBE)算法，实现过程中有几点备注如下：

1. 实现过程中所有属性均用整数表示。在实际应用中是通过索引表将每一个整数和一个字符串属性对应起来。

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 pairingParametersFileName（实例中为 a.properties 文件）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。

5. 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 pk.properties、msk.properties、sk.properties、ct.properties 中。

6. 本实验中的明文消息在最初选用的是随机生成为 GT 群上的点，这样在解密后可以还原出对应的点。后来又尝试选用了单向函数 SHA-256 来对明文消息串进行转换。即需要加密的明文消息串从 input.txt 文件中读取，并通过 SHA-256 哈希为 GT 群上的点 Element 类，进而继续下一步加密。但这样解密出来的结果为对应的哈希点，而无法逆向解出具体的明文消息串。即只能证明具备解密权限，但无法得到解密消息串。

具体实现如下。

1. setup 初始化

```java
public static void setup(String pairingParametersFileName, int U, int d, String pkFileName, String mskFileName) {
    //输入为 < 相关初始化参数文件名，属性全集U（整数），系统门限d（整数），公钥集文件名，主密钥集文件名 >
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);  //从文件导入椭圆曲线参数，生成Pairing实例
    Element g = bp.getG1().newRandomElement().getImmutable();  //基于基本参数，从G1乘法循环群中随机选取Element整数生成元g（阶数）

    Properties mskProp = new Properties();  //新建Properties类以便生成对应主密钥msk封装文件
    Properties pkProp = new Properties();  //新建Properties类以便生成对应公钥pk封装文件
    //属性表示为1，2，3，...，U
    //对每个属性i，选取一个随机数ti作为该属性对应的主密钥，并计算相应公钥g^ti
    for (int i = 1; i <= U; i++){
        Element t = bp.getZr().newRandomElement().getImmutable();  //Element整数ti需要从加法循环群Zp中随机选取
        Element T = g.powZn(t).getImmutable();  //Element整数Ti=g^ti
        mskProp.setProperty("t"+i, Base64.getEncoder().withoutPadding().encodeToString(t.toBytes()));
        pkProp.setProperty("T"+i, Base64.getEncoder().withoutPadding().encodeToString(T.toBytes()));
        //将所得ti、Ti转换为字符串形式并进行Base64编码，并存入对应主密钥、公钥文件中
    }
    //另外选取一个随机数y，计算e(g,g)^y
    Element y = bp.getZr().newRandomElement().getImmutable();  //Element整数y需要从加法循环群Zp中随机选取
    Element egg_y = bp.pairing(g, g).powZn(y).getImmutable();  //基于椭圆曲线基本参数计算Y=egg_y=e(g,g)^y
    mskProp.setProperty("y", Base64.getEncoder().withoutPadding().encodeToString(y.toBytes()));
    pkProp.setProperty("egg_y", Base64.getEncoder().withoutPadding().encodeToString(egg_y.toBytes()));
    pkProp.setProperty("g", Base64.getEncoder().withoutPadding().encodeToString(g.toBytes()));
    //将所得y、egg_y、g转换为字符串形式并进行Base64编码，并存入对应主密钥、公钥、公钥文件中
    //注意区分数据类型。上面写的数据类型群元素，因此使用了Base64编码。
    //d在实际应用中定义为一个int类型，直接用Integer.toString方法转字符串
    pkProp.setProperty("d", Integer.toString(d));

    storePropToFile(mskProp, mskFileName);  //封装进对应文件
    storePropToFile(pkProp, pkFileName);  //封装进对应文件
    //输出：系统主密钥文件 msk = < t1, t2, ..., t|U|, y >，公钥文件 pk = < T1, T2, ..., T|U|, Y, g, d >
}
```

2. keygen 密钥生成

```java
1 usage
public static void keygen(String pairingParametersFileName, int[] userAttList, String pkFileName, String mskFileName, String skFileName) throws NoSuchA
    //输入为 < 相关初始化参数文件名，用户整数属性集，公钥文件名，主密钥文件名，私钥文件名 >
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);  //从文件导入椭圆曲线参数，生成Pairing实例

    Properties pkProp = loadPropFromFile(pkFileName);  //从对应公钥pk封装文件载入公钥Properties实例
    String gString = pkProp.getProperty("g");  //从对应公钥pk文件中获取生成元g
    Element g = bp.getG1().newElementFromBytes(Base64.getDecoder().decode(gString)).getImmutable();
    //将Base64编码后的字符串g解码转换成Element类整数
    String dString = pkProp.getProperty("d");  //从对应公钥pk文件中获取系统门限值d
    int d = Integer.parseInt(dString);
    //将字符串d解码转换成int类整数

    Properties mskProp = loadPropFromFile(mskFileName);  //从对应主密钥msk封装文件载入主密钥Properties实例
    String yString = mskProp.getProperty("y");  //从对应主密钥msk文件中获取随机数密钥y值
    Element y = bp.getZr().newElementFromBytes(Base64.getDecoder().decode(yString)).getImmutable();
    //将Base64编码后的字符串y解码转换成Element类整数

    //d-1次多项式表示为q(x)=coef[0] + coef[1]*x^1 + coef[2]*x^2 + coef[d-1]*x^(d-1)
    //多项式的系数的数据类型为Zr Element，从而使得后续相关计算全部在Zr群上进行
    //通过随机选取coef参数，来构造d-1次多项式q(x)。约束条件为q(0)=y。
    Element[] coef = new Element[d];  //多项式的系数coef个数为d个
    coef[0] = y;  //约束条件即赋值q(0)=a0=coef[0]=y
    for (int i = 1; i < d; i++){
        coef[i] = bp.getZr().newRandomElement().getImmutable();
        //在Zp群上进行随机选取多项式的系数coef
    }
```

```
        Properties skProp = new Properties();  //新建Properties类以便生成对应私钥sk封装文件
        //计算用户属性中每个属性对应的私钥Di=g^(q(i)/ti)，q(i)是多项式在该属性i位置的值，ti是属性对应的主密钥
        for (int att : userAttList) {
            String tString = mskProp.getProperty("t"+att);  //从对应主密钥msk文件中获取对应ti的字符串
            Element t = bp.getZr().newElementFromBytes(Base64.getDecoder().decode(tString)).getImmutable();
            //将Base64编码后的字符串t解码转换成Element类整数
            Element q = qx(bp.getZr().newElement(att), coef, bp.getZr()).getImmutable();
            //计算Element类整数值q(i)=q(att)=coef[0] + coef[1]*att^1 + coef[2]*att^2 + coef[d-1]*att^(d-1)
            //编写了函数qx来计算由coef为系数确定的多项式qx在点x处的值，注意多项式计算在群Zr上进行
            Element D = g.powZn(q.div(t)).getImmutable();
            //计算Element类整数值Di=g^(q(i)/ti)

            skProp.setProperty("D"+att, Base64.getEncoder().withoutPadding().encodeToString(D.toBytes()));
            //将所得Di转换为字符串形式并进行Base64编码，并存入对应私钥文件中
        }
        //将用户属性列表userAttList也添加在私钥中
        skProp.setProperty("userAttList", Arrays.toString(userAttList));
        storePropToFile(skProp, skFileName);  //封装进对应文件
        //输出：私钥文件 sk = < {Di}(i∈userAttList), userAttList >
    }
```

3. encrypt 加密

```
1 usage                                                                                                    11 2
public static void encrypt(String pairingParametersFileName, Element message, int[] messageAttList, String pkFileName, String ctFileName) {
    //输入为 < 相关初始化参数文件名，Element类明文信息（GT上一点），明文属性集，公钥文件名，密文文件名 >
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);  //从文件导入椭圆曲线参数，生成Pairing实例

    Properties pkProp = loadPropFromFile(pkFileName);  //从对应公钥pk封装文件载入公钥Properties实例
    String eggString = pkProp.getProperty("egg_y");  //从对应公钥pk文件中获取公钥egg_y=Y=e(g,g)^y
    Element egg_y = bp.getGT().newElementFromBytes(Base64.getDecoder().decode(eggString)).getImmutable();
    //将Base64编码后的字符串egg_y解码转换成Element类整数

    //计算密文组件 EP=Me(g,g)^(ys)
    Element s = bp.getZr().newRandomElement().getImmutable();
    //选取随机Element类整数s∈Zp
    Element EP = message.duplicate().mul(egg_y.powZn(s)).getImmutable();
    //计算Element类整数EP=M(Y^s)=Me(g,g)^(ys)（M点进行双线性映射得到整数）

    Properties ctProp = new Properties();  //新建Properties类以便生成对应密文ct封装文件
    //针对每个明文属性，计算密文组件 Ei=Ti^s
    for (int att : messageAttList) {
        String TString = pkProp.getProperty("T"+att);  //从对应公钥pk文件中获取对应Ti的字符串
        Element T = bp.getG1().newElementFromBytes(Base64.getDecoder().decode(TString)).getImmutable();
        //将Base64编码后的字符串Ti解码转换成Element类整数
        Element E = T.powZn(s).getImmutable();
        //针对每个明文属性，计算密文组件Element类整数 Ei=Ti^s

        ctProp.setProperty("E"+att, Base64.getEncoder().withoutPadding().encodeToString(E.toBytes()));
        //将所得密文组件Ei转换为字符串形式并进行Base64编码，并存入对应密文文件中
    }
```

```
        ctProp.setProperty("EP", Base64.getEncoder().withoutPadding().encodeToString(EP.toBytes()));
        //将Element类整数EP=M(Y^s)=Me(g,g)^(ys)也转换为字符串形式并进行Base64编码，并存入对应密文文件中
        //密文属性列表messageAttList也添加至密文文件中
        ctProp.setProperty("messageAttList", Arrays.toString(messageAttList));
        storePropToFile(ctProp, ctFileName);  //封装进对应文件
        //输出：密文文件 ct = < E', {Ei}(i∈messageAttList) , EP, messageAttList >
    }
```

4. decrypt 解密

```java
1 usage
public static Element decrypt(String pairingParametersFileName, String pkFileName, String ctFileName, String skFileName) {
    //输入为 < 相关初始化参数文件名，公钥文件名，密文文件名，私钥文件名 >
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);  //从文件导入椭圆曲线参数，生成Pairing实例

    Properties pkProp = loadPropFromFile(pkFileName);  //从对应公钥pk封装文件载入公钥Properties实例
    String dString = pkProp.getProperty("d");  //从对应公钥pk文件中获取公钥系统门限值d
    int d = Integer.parseInt(dString);
    //将字符串d解码转换成int类整数

    Properties ctProp = loadPropFromFile(ctFileName);  //从对应密文ct封装文件载入密文Properties实例
    String messageAttListString = ctProp.getProperty("messageAttList");  //从对应密文ct文件中获取明文属性列表
    //恢复明文消息的属性列表 int[]类型
    int[] messageAttList = Arrays.stream(messageAttListString.substring(1, messageAttListString.length()-1).split( regex: ",")).map(String::trim).mapToIn

    Properties skProp = loadPropFromFile(skFileName);  //从对应私钥sk封装文件载入私钥Properties实例
    String userAttListString = skProp.getProperty("userAttList");  //从对应私钥sk文件中获取用户属性列表
    //恢复用户属性列表 int[]类型
    int[] userAttList = Arrays.stream(userAttListString.substring(1, userAttListString.length()-1).split( regex: ",")).map(String::trim).mapToInt(Integer

    //判断两个列表重合个数是否小于d
    int[] intersectionAttList = intersection(messageAttList, userAttList);  //编写了求两个数组的交集的函数intersection
    System.out.println("重合属性列表：" + Arrays.toString(intersectionAttList));
    System.out.println("重合属性个数为：" + intersectionAttList.length);
    if (intersectionAttList.length < d) {
        System.out.println("不满足解密门限，无法解密！");
        return null;
    }
```

```java
    }
    //从两个列表中的重合项中取前d项，构成解密属性列表
    int[] decAttList = Arrays.copyOfRange(intersectionAttList,  from: 0, d);
    System.out.println("解密所用属性列表：" + Arrays.toString(decAttList));

    Element denominator = bp.getGT().newOneElement().getImmutable();
    //从GT中生成一个Element实例，作为Pi=e(Di,Ei)^delta(0)的连乘结果denominator
    //针对解密属性列表中的每个属性，计算Pie(Di,Ei)^delta(0)，并将结果连乘
    for (int att : decAttList){
        String EString = ctProp.getProperty("E"+att);  //从对应密文ct文件中获取密文组件Ei
        Element E = bp.getG1().newElementFromBytes(Base64.getDecoder().decode(EString)).getImmutable();
        //将Base64编码后的字符串Ei解码转换成Element类整数
        String DString = skProp.getProperty("D"+att);  //从对应私钥sk文件中获取用户每个属性对应的私钥Di
        Element D = bp.getG1().newElementFromBytes(Base64.getDecoder().decode(DString)).getImmutable();
        //将Base64编码后的字符串Di解码转换成Element类整数

        //计算属性对应的拉格朗日因子，作为指数。目标值x为0。
        Element delta = lagrange(att, decAttList,  x: 0, bp.getZr()).getImmutable();
        //编写了拉格朗日因子计算lagrange函数
        denominator = denominator.mul(bp.pairing(E,D).powZn(delta));
        //计算Pi=e(Di,Ei)^deltai(0)=e(g,g)^sq(i)deltai(0)，并将结果连乘
        //实际上得到denominator=e(g,g)^(s·(求和q(i)deltai(0)))===e(g,g)^sy
    }
```

```java
    String EPString = ctProp.getProperty("EP");  //从对应密文ct文件中获取EP=M(Y^s)=Me(g,g)^(ys)
    Element EP = bp.getGT().newElementFromBytes(Base64.getDecoder().decode(EPString)).getImmutable();
    //将Base64编码后的字符串Di解码转换成Element类整数
    //恢复M=EP/denominator=Me(g,g)^(ys)/e(g,g)^sy=M
    Element res = EP.div(denominator);
    return res;
    //输出返回为解密后的M（点）
}
```

5. 相关函数

① 计算由 coef（即 a_n）为系数确定的多项式 q(x)在点 x 处的值

```java
//计算由coef为系数确定的多项式qx在点x处的值，注意多项式计算在群Zr上进行
1 usage
public static Element qx(Element x, Element[] coef, Field Zr){
    Element res = coef[0];
    for (int i = 1; i < coef.length; i++){
        Element exp = Zr.newElement(i).getImmutable();
        //x一定要使用duplicate复制使用，因为x在每一次循环中都要使用，如果不加duplicte，x的值会发生变化
        res = res.add(coef[i].mul(x.duplicate().powZn(exp)));
    }
    return res;
}
```

② 求两个数的交集

```java
//求两个数组的交集
1 usage
public static int[] intersection(int[] nums1, int[] nums2) {
    Arrays.sort(nums1);
    Arrays.sort(nums2);
    int length1 = nums1.length, length2 = nums2.length;
    int[] intersection = new int[length1 + length2];
    int index = 0, index1 = 0, index2 = 0;
    while (index1 < length1 && index2 < length2) {
        int num1 = nums1[index1], num2 = nums2[index2];
        if (num1 == num2) {
            // 保证加入元素的唯一性
            if (index == 0 || num1 != intersection[index - 1]) {
                intersection[index++] = num1;
            }
            index1++;
            index2++;
        } else if (num1 < num2) {
            index1++;
        } else {
            index2++;
        }
    }
    return Arrays.copyOfRange(intersection, from: 0, index);
}
```

③ 拉格朗日因子计算

```java
//拉格朗日因子计算 i是集合S中的某个元素，x是目标点的值
1 usage
public static Element lagrange(int i, int[] S, int x, Field Zr) {
    Element res = Zr.newOneElement().getImmutable();
    Element iElement = Zr.newElement(i).getImmutable();
    Element xElement = Zr.newElement(x).getImmutable();
    for (int j : S) {
        if (i != j) {
            //注意：在循环中重复使用的项一定要用duplicate复制出来使用
            //这儿xElement和iElement重复使用，但因为前面已经getImmutable所以可以不用duplicate
            Element numerator = xElement.sub(Zr.newElement(j));
            Element denominator = iElement.sub(Zr.newElement(j));
            res = res.mul(numerator.div(denominator));
        }
    }
    return res;
}
```

④ 保存和载入参数文件

```java
4 usages
public static void storePropToFile(Properties prop, String fileName){
    try(FileOutputStream out = new FileOutputStream(fileName)){
        prop.store(out, comments: null);
    }
    catch (IOException e) {
        e.printStackTrace();
        System.out.println(fileName + " save failed!");
        System.exit( status: -1);
    }
}

6 usages
public static Properties loadPropFromFile(String fileName) {
    Properties prop = new Properties();
    try (FileInputStream in = new FileInputStream(fileName)){
        prop.load(in);
    }
    catch (IOException e){
        e.printStackTrace();
        System.out.println(fileName + " load failed!");
        System.exit( status: -1);
    }
    return prop;
}
```

⑤ 从文件中读取字符串消息

```java
// 从文件中读取明文字符串消息
1 usage
private static String readPlaintextFromFile(String fileName) {
    StringBuilder plaintext = new StringBuilder();
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
        while ((line = reader.readLine()) != null) {
            plaintext.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return plaintext.toString();
}
```

⑥ 将明文字符串消息使用 SHA-256 哈希为 GT 群上的点 Element 类

```java
// 将明文字符串消息哈希为GT群上的点Element类
1 usage
private static Element hashToElement(String plaintext, String pairingParametersFileName) {
    try {
        MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] hashBytes = digest.digest(plaintext.getBytes(StandardCharsets.UTF_8));
        BigInteger hashInt = new BigInteger( signum: 1, hashBytes);
        return PairingFactory.getPairing(pairingParametersFileName).getGT().newElementFromHash(hashInt.toByteArray(), i: 0, hashInt.toByteArray().leng
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return null;
}
```

6. 主函数

```
    public static void main(String[] args) throws Exception {
        int U = 20;
        int d = 5;
        System.out.println("系统解密门限为：" + d);

        int[] userAttList = {1, 5, 3, 6, 10, 11};
        int[] messageAttList = {1,  3,  5,  7, 9, 10, 11};

        String dir = "data/";
        String pairingParametersFileName = "a.properties";
        String pkFileName = dir + "pk.properties";
        String mskFileName = dir + "msk.properties";
        String skFileName = dir + "sk.properties";
        String ctFileName = dir + "ct.properties";

        String inputFileName = dir + "input.txt"; // 输入文件名


        // 读取明文字符串消息
        String plaintext = readPlaintextFromFile(inputFileName);
        System.out.println("明文消息: " + plaintext);

        // 将明文字符串消息哈希为GT群上的点Element类
        Element M = hashToElement(plaintext,pairingParametersFileName);

        // 打印哈希后的Element
        System.out.println("哈希后的明文消息: " + M);
```

```

        setup(pairingParametersFileName, U, d, pkFileName, mskFileName);

        keygen(pairingParametersFileName, userAttList, pkFileName, mskFileName, skFileName);

        System.out.println("加密密钥文件:" + pkFileName);

        //Element message = PairingFactory.getPairing(pairingParametersFileName).getGT().newRandomElement().getImmutable()

        //System.out.println("明文消息:" + message);
        //encrypt(pairingParametersFileName, message, messageAttList, pkFileName, ctFileName);
        encrypt(pairingParametersFileName, M, messageAttList, pkFileName, ctFileName);

        Element res = decrypt(pairingParametersFileName, pkFileName, ctFileName, skFileName);
        System.out.println("解密结果:" + res);
        if (M.isEqual(res)) {
            System.out.println("成功解密！");
        }

        //if (message.isEqual(res)) {
          //  System.out.println("成功解密！");
        //}


    }

}

```

7. 运行检验

① 测试数据：

属性集为{1, 2, ..., 20}，即 U=20。

系统解密门限值 d=5。

明文消息串为“hello”。

② 用户属性与明文属性重合个数大于等于系统解密门限值的情况

测试数据：

用户属性 = {1, 5, 3, 6, 10, 11}；明文属性 = {1, 3, 5, 7, 9, 10, 11}。

运行得到：

系统解密门限为：5

明文消息: hello

哈希后的明文消息:

{x=70106160125068299084316410300783153034619529792365851860698082290033804753970802774446276924480872810277124668903675225950371189515221642957593478262768 73,y=12832580375525523730793182477968770010395540439481961418334243270123742684094265190133684523986727749578310464228790203457717747160840204981120080343937 69}

加密密钥文件:data/pk.properties

重合属性列表：[1, 3, 5, 10, 11]

重合属性个数为：5

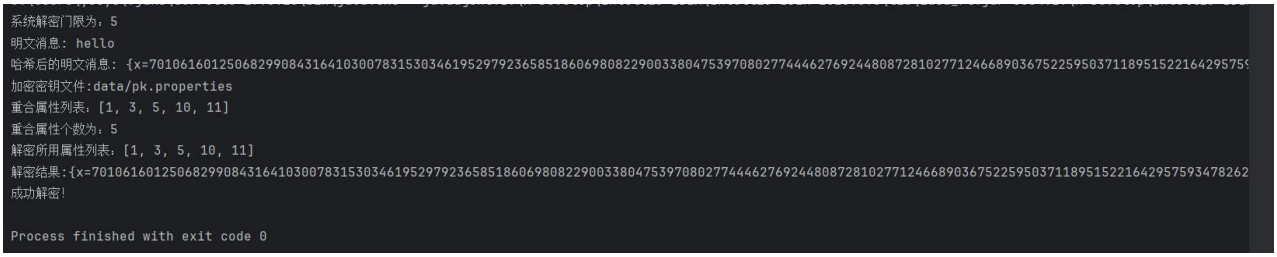
解密所用属性列表：[1, 3, 5, 10, 11]

解密结果:

{x=70106160125068299084316410300783153034619529792365851860698082290033804753970802774446276924480872810277124668903675225950371189515221642957593478262768 73,y=12832580375525523730793182477968770010395540439481961418334243270123742684094265190133684523986727749578310464228790203457717747160840204981120080343937 69}

80822900338047539708027744462769244808728102771246689036752259503711
89515221642957593478262 76873,y=12832580375525523730793182477968770010
39554043948196141833424327012374268409426519013368452398672774957831
04642287902034577177471608402049811200803439376 9}

成功解密！



③ 用户属性与明文属性重合个数小于系统解密门限值的情况

测试数据：

用户属性 ={1, 5, 3, 6, 10, 11}；明文属性 ={1, 3, 5, 7, 9}。

运行得到：

系统解密门限为：5

明文消息: hello

哈希后的明文消息:

{x=70106160125068299084316410300783153034619529792365851860698
80822900338047539708027744462769244808728102771246689036752259503711
89515221642957593478262 76873,y=12832580375525523730793182477968770010
39554043948196141833424327012374268409426519013368452398672774957831
04642287902034577177471608402049811200803439376 9}

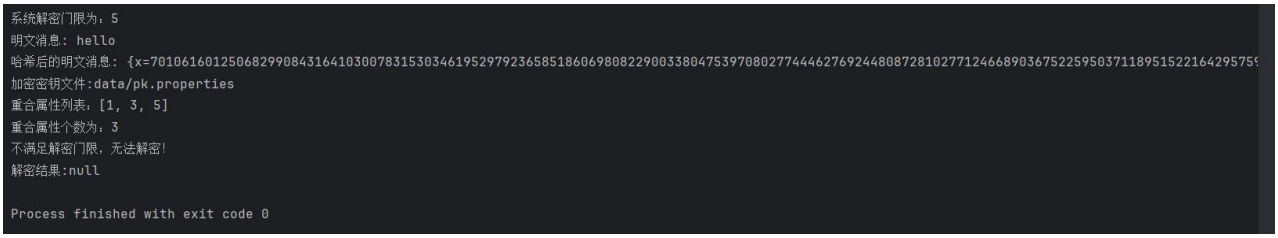
加密密钥文件:data/pk.properties

重合属性列表：[1, 3, 5]

重合属性个数为：3

不满足解密门限，无法解密！

解密结果:null



五、实验结论

在 FIBE 中，将身份看作是一组描述性属性。它允许具有身份 w 的用户去解密用身份 w’加密的密文，当|w∩w’|≥d，即在一定的度量下，他们的身份重叠大于等于门限值 d，也就是他们之间的属性重合大于等于门限值时，所以 FIBE 具有一定的容错特性，可以使用生物识别特征作为属性输入。

同时，它还具有抗共谋的特性，对于不同用户组合他们的属性，从而解密他们各自不能解密的密文，这在 FIBE 中是不允许的，主要是使用生成随机多项式来实现抗共谋攻击。FIBE 的提出产生了两个新的新应用，第一个就是可以使用生物识别身份的身份基加密系统。例如{身高，体重，血型}等生物特征作为身份。第二个就是属性基加密系统。例如{本科生，研究生，教师}等属性作为身份。

FIBE 就是最基本的属性基加密系统，将系统中的属性映射到中，密文和用户的密钥都与属性相关。该机制仅支持基于属性的门限策略，即当用户属性集合与密文属性集合相交的属性数量达到系统规定的门限值以上才允许该用户对

密文进行解密。例如，图书馆中某论文的属性集合为{计算机，安全，研究生，英文}，且该论文属性加密门限值为 3，则属性基为{计算机，安全，研究生}可以访问该论文，而属性集合为{计算机，安全，本科生}的用户就无法访问它。

该机制中，公钥与系统属性数目线性相关，幂运算和双线性对数目较多，系统越大，计算复杂。

本次实验 Github 地址：https://github.com/YTR1020/FIBE/tree/main

参考文献

[1]Sahai, A., Waters, B. (2005). Fuzzy Identity-Based Encryption. In: Cramer, R. (eds) Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science, vol 3494. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11426639_27（pdf 地址：https://eprint.iacr.org/2004/086.pdf）

[2]十陆.（2022）.模糊身份基加密 (Fuzzy Identity based Encryption)算法及 JPBC 实现. CSDN. 检索于 2024 年 4 月 2 日, https://blog.csdn.net/weixin_44960315/article/details/122394033.

[3]Intuzgm.（2022）.属性基加密——模糊身份基加密（FIBE）. CSDN. 检索于 2024 年 4 月 2 日, https://blog.csdn.net/m0_52322997/article/details/124997902.