KP- ABE 的实现

一、实验目的

属性基加密（Attribute-Based Encryption, ABE）是一种先进的公钥密码学方案，它突破了传统加密技术中"一对一"或"一对多"的访问控制限制，实现了基于属性的"一对多"细粒度访问控制。

本次 ABE 方案复现实验的目的为：（1）实现基本的 KP-ABE 加解密算法；（2）理解双线性配对、访问树构造等核心算法原理；（3）熟悉使用密码库进行密码算法编程。

阅读给定的基于属性加密方案的论文，实现其各部分算法。

二、前置背景知识：

（1）访问树$\mathcal{T}$的定义与结构：

设$\mathcal{T}$为表示访问结构的树。树的每个非叶节点对应一个阈值门，由其子节点和阈值$\mathrm{k_x}$描述：若节点 x的子节点数为$\mathrm{num_x}$，其阈值满足$0 < \mathrm{k_x} \leq \mathrm{num_x}$

当$\mathrm{k_x} = 1$时，阈值门为 OR 门，表示所有叶子中只需满足一个即可。

当$\mathrm{k_x} = \mathrm{num_x}$时，阈值门为 AND 门，表示所有叶子都需要满足。

每个叶节点x关联一个属性，且阈值固定为$\mathrm{k_x} = 1$。

为便于操作访问树，定义以下函数：

父节点：parent(x)表示节点 x的父节点。

属性标记：att(x) 仅对叶节点有效，返回其关联的属性。

每个节点的子节点按 1 至 num 编号。index(x) 返回节点x的编号（索引值在密钥生成时唯一分配）。

访问树的满足条件：

设$\mathcal{T}$是以r为根的访问树，$\mathcal{T}_\mathrm{x}$ 表示以 x为根的子树（因此$\mathcal{T}$=$\mathcal{T}_\mathrm{r}$）。若属性集合γ 满足$\mathcal{T}_\mathrm{x}$，则记作$\mathcal{T}_\mathrm{x}(\gamma) = 1$.。其递归计算规则如下：

x为非叶节点时 ：

1、对所有子节点x′计算 $\mathcal{T}_\mathrm{x}'(\gamma)$。

2、当且仅当至少k_x个子节点返回 1 时，$\mathcal{T}_x(\gamma)=1$。

x为叶节点 时：

当且仅当 att(x) ∈ γ 时（即叶子节点关联的属性包含在属性集合中），$\mathcal{T}_x(\gamma)=1$。

（2）双线性映射与拉格朗日系数定义：

设 $\mathbb{G}_1$为素数阶 p 的双线性群， g是$\mathbb{G}_1$ 的生成元。双线性映射定义为 $e:\mathbb{G}_1\times\mathbb{G}_1\to\mathbb{G}_2$。安全参数 κ 决定群的规模。

定义拉格朗日系数为：

$$\Delta_{\{i,S\}}(x)=\prod_{\{j\in S,j\neq i\}}\frac{\{x-j\}}{\{i-j\}}$$

其中每个属性关联一个唯一的$\mathbb{Z}_p^*$ 元素。

三、实验内容

实现如下算法：

1、 $Setup$：用于生成后续算法需要的公私钥对，权威机构生成主密钥和公开参数。

定义属性域$\mathcal{U}=\{1,2,\dots,n\}$。对于每个属性$i\in\mathcal{U}$，从$\mathbb{Z}_p$中均匀随机地选择一个数$t_i$。最后，在$\mathbb{Z}_p$中均匀随机地选择$y$。

最终输出：公开参数PK：$T_1=g^{t_1},\dots,T_{|\mathcal{U}|}=g^{t_{|\mathcal{U}|}},Y=e(g,g)^y$；主密钥$MK$：$t_1,\dots,t_{|\mathcal{U}|},y$。

2、 $Key\ Generation(\mathcal{T},MK)$ ：用于生成解密密钥，根据用户属性生成个性化解密密钥。该算法输出一个使用户能解密满足$\mathcal{T}(\gamma)=1$ 的密文的密钥：

（1）多项式分配：从根节点 r 开始，为树中每个节点x分配多项式 q_x：

- 根节点：设$q_r(0)=y$，随机选择 $d_r=k_r-1$ 个点定义q_r
- 非根节点：设$q_x(0)=q_{parent(x)}(index(x))$，随机选择$d_x=k_x-1$个点定义 q_x

（2）密钥构造：对每个叶节点x，生成解密分量：

$$D_x=g^{\frac{q_x(0)}{t_i}}\ where\ i=att(x)\ .$$

最终解密密钥D为所有D_x的集合。

3、 $Encryption(M,\gamma,\mathrm{PK})$：用于加密明文，发送者通过指定属性或策略加密明文。

在一组属性γ下加密消息$M \in \mathbb{G}_2$，选择一个随机值$s \in \mathbb{Z}_p$，并计算密文如下：

$$E = \left(\gamma, E' = MY^s, \{E_i = T_i^s\}_{i\in\gamma}\right)$$

4、 $Decryption(E,D)$:用于解密明文，只有属性满足条件的用户才能解密。将解密过程指定为一个递归算法。

递归算法 $DecryptNode(E,D,x)$ 定义如下：

将密文 $E = \left(\gamma, E', \{E_i\}_{i\in\gamma}\right)$，私钥 D 和树中的一个节点$x$作为输入. 输出一个 $\mathbb{G}_2$ 上的群元素或输出⊥。令 $i = att(x)$。 叶节点处理：

$$DecryptNode(E,D,x) = \begin{cases} e(D_x, E_i) = e\left(g^{\frac{q_x(0)}{t_i}}, g^{s\cdot t_i}\right) = e(g,g)^{s\cdot q_x(0)} if i \in \gamma \\ \perp \ otherwise \end{cases}$$

非叶节点处理：

（1）对子节点z递归计算$F_z = DecryptNode(E,D,z)$

（2）选取满足 $\mathrm{F_z} \neq \perp$的k_x个子节点构成集合S_x

（3）计算插值结果：

$$\begin{aligned} F_x &= \prod_{z\in S_x} F_z^{\Delta_{i,S'_x}(0)}, \quad where \quad i = index(z) \\ &= \prod_{z\in S_x} \left(e(g,g)^{s\cdot q_x(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z\in S_x} \left(e(g,g)^{s\cdot q_{parent}(z)(indca(z))}\right)^{\Delta_{i,S'_x}(0)} \quad (by\ construction) \\ &= \prod_{z\in S_x} e\,(g,g)^{s\cdot q_x(i)\cdot\Delta_{i,S'_x}(0)} \\ &= \quad e(g,g)^{s\cdot q_x(0)} \quad (using\ polynomial\ interpolation) \end{aligned}$$

最终解密：调用$DecryptNode(E,D,r)$ 得到$e(g,g)^{ys} = Y^s$，通过$E' = MY^s$恢复明文M。

更具体的算法实现部分可以参照论文原文第四章部分[1]。

四、程序设计工具

本次实验需要编程实现 ABE 密码学算法，可以选择采用 C++或者 python 算法实现。接下来介绍编程过程中需要使用的密码学库。

1、 PBC 库

PBC 库是一个用于双线性对(pairing)运算的免费 C 语言库[2]，它为基于配对的密码学方案提供了基础实现。该库构建在 GMP(GNU Multiple Precision Arithmetic Library)之上，专门为双线性对运算优化。在本次实验中可以用于编程实现 ABE 相应的算法。下面介绍一些该库的简要操作。

推荐大家用 type A 曲线，参数为：（rbit = 160, qbit = 512）。

（1）基本数据结构

```
element_t g;            // G1 群生成元
element_t h;            // G2 群元素
element_t gt;           // GT 群元素
element_t z;            // Zp 域元素
```

（2）初始化与参数设置

```
pairing_init_inp_str(pairing, param_str);   //初始化元素
element_init_G1(g, pairing);
element_init_G2(h, pairing);
element_init_GT(gt, pairing);
element_init_Zr(z, pairing);
```

（3）关键运算操作

```
pairing_apply(gt, g, h, pairing);    // e(g,h)→gt
//  群运算
element_add(a, b, c);           // a = b + c
element_mul(a, b, c);           // a = b * c
element_pow_zn(a, b, z);        // a = b^z
//  随机数生成
element_random(z);              //  随机 Zp 元素
```

2、 Charm 库

Charm 库是一个用于进行快速加密的 Python 库，它的底层使用 PBC 库进行配对运算。如果你是对 Python 熟悉的同学，可以考虑使用 Charm 库进行本次实验的编程实现。以下是 Charm 库的简要操作，详细操作请见官方文档[3]。

（0）说明

以下的 python 代码片段开头均有以下代码。

```
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2,GT,pair
group = PairingGroup('SS512')
```

（1）生成基本数据结构

```
g = group.gen1(G1)          # G1 群生成元
h = group.random(G2)        # G2 群元素
gt = group.random(GT)       # GT 群元素
z = group.random(ZR)        # Zp 域元素
```

（2）关键运算操作

```
# 配对运算
gt = group.pair_prod(a,b) # e(g,h)→gt
# 群运算
a = b + c          # a = b + c
a = b * c          # a = b * c
a = b ** z         # a = b^z
```

五、资料

[1] 论文名称：Goyal V, Pandey O, Sahai A, et al. Attribute-based encryption for fine grained access control of encrypted data[C]//Proceedings of the 13th ACM conference on Computer and communications security. Acm, 2006: 89-98.

[2] pbc 库资料 https://crypto.stanford.edu/pbc/manual.pdf

[3] charm 库资料 https://jhuisi.github.io/charm/