

基于配对的密码学——基础知识及JPBC库

📅 2021-07-05 | 📁 Crypto_Knowledge | 👁 °C

📄 4,835 | ⌚ 21

0x00 前言

本科毕业设计过程中需要使用JPBC库实现Java语言下双线性配对运算的仿真，摸索过程中遇到一些问题及特性，记录如下。本文参考李发根等编著的《基于配对的密码学》一书，首先简要介绍基于配对密码学的相关性质，随后结合JPBC文档介绍该库中部分函数的特殊性质及用法。

参考链接：

- 《基于配对的密码学》

链接：<https://pan.baidu.com/s/1bocycprbAUtNzCopkF1v-A> 提取码：oe09

- JPBC jar包

链接：<https://pan.baidu.com/s/1MOZCaplESGF0gVk5dNeLGQ> 提取码：sa5m

- JPBC DOCS

链接：<http://gas.dia.unisa.it/projects/jpbc>

0x01 椭圆曲线密码体制

椭圆曲线密码体制（elliptic curve cryptosystem, ECC）是公钥密码体制的一个重要分支，其安全性基于椭圆曲线离散对数问题的困难性。该问题比大整数因子分解问题和有限域上的离散对数问题难得多。由于还没有找到求解椭圆曲线离散对数的亚指数算法，因此椭圆曲线密码体制可使用更短的密钥以保证相同的安全性。

有限域上的椭圆曲线

有限域上的椭圆曲线是指变量和系数均为有限域中元素的椭圆曲线。有限域 $GF(p)$ 上的椭圆曲线是指满足方程

$$y^2 \equiv x^3 + ax + b(mod p)$$

的所有点 (x, y) 及一个无穷远点 O 构成的集合，其中 a, b, x 和 y 均在有限域 $GF(p)$ 上取值， p 是素数。这里将该椭圆曲线记为 $E_p(a, b)$ ，曲线上只有有限个点，其个数 N 由Hasse定理确定。

Hasse定理 设 E 是有限域 $GF(p)$ 上的椭圆曲线， N 是 E 上点的个数，则满足

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

当 $4a^3 + 27b^2 \pmod{p} \neq 0$ 时，基于集合 $E_p(a, b)$ 可以定义一个Abel群，其加法规则与实数域上描述的代数方法一致。设 $P, Q \in E_p(a, b)$ ，则

$$P + O = P$$

- 如果 $P = (x, y)$ ，那么 $(x, y) + (x, -y) = O$ ，即点 $(x, -y)$ 是 P 的加法逆元，表示为 $-P$ 。
- 设 $P = (x_1, y_1)$ 和 $Q = (x_2, y_2)$ ， $P \neq -Q$ ，则 $S = P + Q = (x_3, y_3)$ 由以下规则确定：

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}$$

$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$

式中

$$\lambda \equiv (y_2 - y_1)/(x_2 - x_1) \pmod{p}, \text{ if } P \neq Q$$

$$\lambda \equiv (3x_1^2 + a)/2y_1 \pmod{p}, \text{ if } P = Q$$

- 倍点运算定义为重复加法，即 $3P = P + P + P$ 。

椭圆曲线上的ElGamal加密体制

- 定义1 椭圆曲线的阶

椭圆曲线 $E_p(a, b)$ 上点 P 的阶是指满足

$$nP = \sum_{i=1}^n P = O$$

的最小正整数，记为 $ord(P)$ ，其中 O 是无穷远点。

- 定义2 椭圆曲线上离散对数问题

设 G 是椭圆曲线 $E_p(a, b)$ 上的一个循环子群， P 是 G 的一个生成元， $Q \in G$ 。已知 P 和 Q ，求满足

$$mP = Q$$

的整数 m , $0 \leq m \leq \text{ord}(P) - 1$, 称为椭圆曲线上的离散对数问题 (elliptic curve discrete logarithm problem, ECDLP)。计算 mP 的过程称为点乘运算。

EC-ElGamal密码体制包含以下四个元操作:

- 编码与解码

将待发送的明文 m 编码为椭圆曲线上的点 $P_m = (x_m, y_m)$, 随后执行的加解密操作均针对点 P_m , 解密后的点 P_m 需执行**逆向解码**操作才可以获得明文。

- 密钥生成

在椭圆曲线 $E_p(a, b)$ 上选取一个阶为大素数 n 的生成元 P 。随机选取整数 x 满足 $1 < x < n$, 计算 $Q = xP$, 将 x 作为私钥, Q 作为公钥。

- 加密

为加密 P_m , 随机选取一个整数 k 满足 $1 < k < n$, 计算

$$C_1 = kP, C_2 = P_m + kQ$$

则密文 $c = (C_1, C_2)$ 。

- 解密

为解密密文 $c = (C_1, C_2)$, 计算

$$C_2 - xC_1 = P_m + kQ - xkP = P_m + kxP - xkP = P_m$$

攻击者若妄图通过 $c = (C_1, C_2)$ 计算出 P_m , 则必须获得 k 。攻击者需要通过 P 和 kP 推算出 k 的值, 这一过程面临求解椭圆曲线上的离散对数问题。

0x02 双线性配对理论

抽象双线性配对

设 k 为安全参数, p 为 k 比特长的素数。令 G_1 为由 P 生成的循环加法群, 阶为 p , G_T 为具有相同阶 p 的循环乘法群, a, b 是 Z_p^* 中的元素。 0 表示 G_1 中的单位元, 1 表示 G_T 中的单位元。假设 G_1 和 G_T 这两个群中的离散对数问题都是困难问题。双线性配对是指满足下列性质的映射 $e: G_1 \times G_1 \rightarrow G_T$ 。

- 双线性性 (bilinearity): 对于任意的 $P, Q \in G_1$ 和 $a, b \in Z_p^*$, $e(aP, bQ) = e(P, Q)^{ab}$ 成立。

- 非退化性 (non-degeneracy) : 存在 $P, Q \in G_1$, 使得 $e(P, Q) \neq 1$ 。同时, 满足 $e(0, Q) = e(Q, 0) = 1$ 。
- 可计算性 (computability) : 对于所有 $P, Q \in G_1$, 存在有效的算法计算 $e(P, Q)$ 。

双线性映射可以通过有限域上的超奇异椭圆曲线或超奇异超椭圆曲线中的Weil配对或Tate配对推导出来。

非对称双线性配对

设计密码体制时, 有时会遇到非对称的配对。

令 G_1, G_2, G_T 为具有相同阶 p 的群, P 为 G_1 的生成元, Q 为 G_2 的生成元。非对称双线性配对指满足下列性质的一个映射: $e : G_1 \times G_2 \rightarrow G_T$ 。

- 双线性性: 对于任意 $(S, T) \in G_1 \times G_2$ 和 $a, b \in \mathbb{Z}_p^*$, $e(aS, bT) = e(S, T)^{ab}$ 成立。
- 非退化性: 存在 $(S, T) \in G_1 \times G_2$, 使得 $e(S, T) \neq 1$ 。
- 可计算性: 对于所有的 $(S, T) \in G_1 \times G_2$, 存在有效算法计算 $e(S, T)$ 。
- 存在一个有效的、可公开计算的同构映射 $\phi : G_2 \rightarrow G_1$, 满足 $\phi(Q) = P$ 。这个映射必须是不可逆的。

若令 $G_2 = G_1$ 且映射 ϕ 为恒等映射, 此时非对称配对就变成了对称配对。尽管对称配对比较简单且应用方便, 但只能从超奇异超椭圆曲线中的Weil配对或Tate配对推导出来。非对称配对比较复杂, 但不仅可以从超奇异超椭圆曲线中推导出来, 还可以从普通椭圆曲线中的Weil配对或Tate配对推导出来。

0x03 困难问题

计算Diffie-Hellman问题

给定一个阶为 p 的循环加法群 G_1 和一个生成元 P , G_1 中的计算Diffie-Hellman (computational Diffie-Hellman, CDH) 问题是给定 (P, aP, bP) , 计算 $abP \in G_1$ 。这里 $a, b \in \mathbb{Z}_p^*$ 是未知整数。

判定Diffie-Hellman问题

给定一个阶为 p 的循环加法群 G_1 和一个生成元 P , G_1 中的判定Diffie-Hellman (decisional Diffie-Hellman, DDH) 问题是给定 (P, aP, bP, cP) , 判断 $c \equiv ab \pmod{p}$ 是否成立。这里 $a, b, c \in \mathbb{Z}_p^*$ 是未知整数。若 (P, aP, bP, cP) 满足上述条件, 则称其为一个“Diffie-Hellman元组”, 可采用记号 $cP = DH_p(aP, bP)$ 来表示。

间隙Diffie-Hellman问题

给定一个阶为 p 的循环加法群 G_1 和一个生成元 P , G_1 中的间隙Diffie-Hellman (gap Diffie-Hellman, GDH) 问题是在DDH预言机的帮助下, 求解一个给定元组 (P, aP, bP) 的CDH问题。DDH预言机可以判断

(P, aP, bP, cP) 是否满足 $c \equiv ab \pmod{p}$ 。

q-强Diffie-Hellman问题

给定一个阶为 p 的循环加法群 G_1 和一个生成元 P ， G_1 中的 q -强Diffie-Hellman (q-strong Diffie-Hellman, q-SDH) 问题是给定 $(P, xP, x^2P, \dots, x^qP)$ ，计算

$$(c, P/(x + c)) \in \mathbb{Z}_p \times G_1$$

双线性Diffie-Hellman问题

给定两个阶都为 p 的循环加法群 G_1 和循环乘法群 G_T ，一个双线性映射 $e : G_1 \times G_1 \rightarrow G_T$ 和一个群 G_1 的生成元 P ，双线性Diffie-Hellman (bilinear Diffie-Hellman, BDH) 问题是给定 (P, aP, bP, cP) ，计算 $e(P, P)^{abc} \in G_T$ 。这里的 $a, b, c \in \mathbb{Z}_p^*$ 是未知整数。

判定双线性Diffie-Hellman问题

给定两个阶都为 p 的循环加法群 G_1 和循环乘法群 G_T ，一个双线性映射 $e : G_1 \times G_1 \rightarrow G_T$ 和一个群 G_1 的生成元 P ，判定双线性Diffie-Hellman (decisional bilinear Diffie-Hellman, DBDH) 问题是给定 (P, aP, bP, cP) 和 $z \in G_T$ ，判断

$$z = e(P, P)^{abc}$$

是否成立。这里的 $a, b, c \in \mathbb{Z}_p^*$ 是未知整数。

间隙双线性Diffie-Hellman问题

给定两个阶都为 p 的循环加法群 G_1 和循环乘法群 G_T ，一个双线性映射 $e : G_1 \times G_1 \rightarrow G_T$ 和一个群 G_1 的生成元 P ，间隙双线性Diffie-Hellman (gap bilinear Diffie-Hellman, GBDH) 问题是在DBDH预言机的帮助下，求解一个给定元组 (P, aP, bP, cP) 的BDH问题。DBDH预言机可以判断一个元组 (P, aP, bP, cP, z) 是否满足

$$z = e(P, P)^{abc}$$

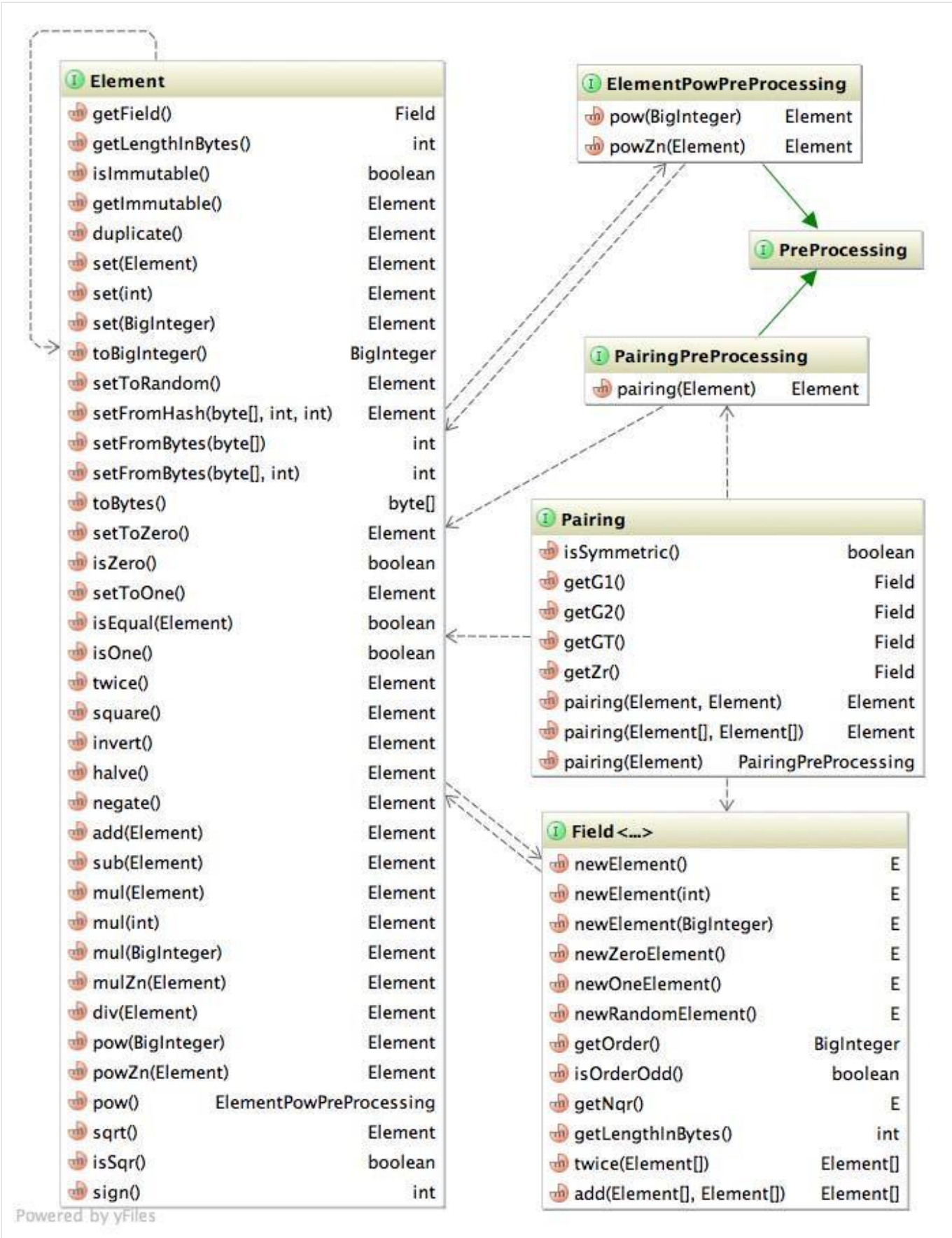
上述问题通常被视为困难问题，但其困难程度不尽相同。显然，判定问题不比计算问题更难，即如果能够求解CDH问题，那么DDH问题就容易解决；同样如果能够求解BDH问题，那么DBDH问题就容易解决。

0x04 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。

该库提供的各类API结构如下。



入门教程

除JPBC文档外，整理一些优秀视频及技术博客，链接如下：

[JPBC库（基于配对的密码学）入门和避坑指南](#)

[JPBC库应用之BLS签名](#)

[CSDN JPBC Library 专栏](#)

[刘巍然大佬的博客](#)

群上点的特性

JPBC库共提供四个循环群，其中 G_1, G_2, G_T 均为阶为 p 的乘法循环群，而 Z_p 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。生成测试元素并打印，结果如下：

```
1 // 生成测试元素
2 Element g1 = G1.newRandomElement();
3 System.out.println(g1);
4 Element g2 = G2.newRandomElement();
5 System.out.println(g2);
6 Element gt = Gt.newRandomElement();
7 System.out.println(gt);
8 Element zp = Zr.newRandomElement();
9 System.out.println(zp);
```

```
48297942592027602052600849461219507770426392327438,18155115627069074404435088323871811857974817934198,0
47980814087134329554358620569868040773264582072507,903229727577129282965806843436102411631888682997,0
{x=35154519223970026140973837744277574925455198295428,y=43888061774017202146605936583223472582025826380573}
580372014309974435829187757063595679845282293554
```

群运算

JPBC库支持的运算如下：

- G_1, G_2, G_T 中元素的模幂运算、倍乘运算以及互相之间的加法运算，运算结果均为对应群上的元素。
- Z_p 中元素的加减乘除运算及乘方运算，运算结果为整数循环群上的元素。

测试相关运算，并打印对应结果如下：

```
1 // 相关运算
2 Element a = Zr.newRandomElement();
```

```

3  Element b = Zr.newRandomElement();
4  // 1. g1^a
5  Element g1_pow_a = g1.duplicate().powZn(a);
6  System.out.println("g1^a");
7  System.out.println(g1_pow_a);
8  // 2. a*g1
9  Element g1_mul_a = g1.duplicate().mulZn(a);
10 System.out.println("a*g1");
11 System.out.println(g1_mul_a);
12 // 3. g1+g2
13 Element g1_add_g2 = g1.duplicate().add(g2);
14 System.out.println("g1+g2");
15 System.out.println(g1_add_g2);
16 // 4. gt^b
17 Element gt_pow_b = gt.duplicate().powZn(b);
18 System.out.println("gt^b");
19 System.out.println(gt_pow_b);
20 // 5. b*gt
21 Element gt_mul_b = gt.duplicate().mulZn(b);
22 System.out.println("b*gt");
23 System.out.println(gt_mul_b);
24 // 6. gt+gt
25 Element gt_add_gt = gt.duplicate().add(gt);
26 System.out.println("gt+gt");
27 System.out.println(gt_add_gt);
28 // 7. a+b
29 Element a_add_b = a.duplicate().add(b);
30 System.out.println("a+b");
31 System.out.println(a_add_b);
32 // 8. a*b
33 Element a_mul_b = a.duplicate().mulZn(b);
34 System.out.println("a*b");
35 System.out.println(a_mul_b);

```

```

g1^a
48967826235925428995909048492806072603644339561060,16855691689017919128962064290782232595936642170904,0
a*g1
48967826235925428995909048492806072603644339561060,16855691689017919128962064290782232595936642170904,0
g1+g2
27480420987470931618067251517228632023016816969127,31004551452242001323153064667453898685322671217881,0
gt^b
{x=16532185977938182333065816116906202395162789508980,y=862891834448110227997224870725049515764431098546}
b*gt
{x=16532185977938182333065816116906202395162789508980,y=862891834448110227997224870725049515764431098546}
gt+gt
{x=3524476572945555421714753770222404522389887979372,y=3213985844846530986640563294531140690987710876522}
a+b
530624336822665196336684539174941942880374185354
a*b
300932798278767423489905482898389832495260036974

```


值得注意的是，现在的密码学相关论文中，习惯将 G_1, G_2 设置为乘法循环群。但是基于椭圆曲线的双线性群构造中， G_1, G_2 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 G_1, G_2 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。

初始化双线性群

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	$10n$

代码动态产生

动态产生的方法大概包括以下几个步骤：

- 指定椭圆曲线的种类
- 产生椭圆曲线参数
- 初始化Pairing对象

Type A曲线初始化过程中需要提供两个参数：`rBit` 代表 Z_p 中阶数 p 的比特长度，`qBit` 代表 G 中阶数的比特长度，生成代码如下：

```
1  TypeACurveGenerator pg = new TypeACurveGenerator(rBit, qBit);
2  PairingParameters typeAParams = pg.generate();
3  Pairing bp = PairingFactory.getPairing(typeAParams);
```

Type A1曲线需要提供两个参数：`numPrime` 是阶数 N 中包含质数因子的数量，`qBit` 是每个质数因子的比特长度。由于Type A1曲线涉及到的阶数较大，故参数产生的时间较长，代码如下：

```

1  TypeA1CurveGenerator pg = new TypeA1CurveGenerator(numPrime, qBit);
2  PairingParameters typeA1Params = pg.generate();
3  Pairing pairing = PairingFactory.getPairing(typeA1Params);

```

文件读取产生

当然我们可以选择事先生成参数并存放至文件中。在后续初始化过程中直接从文件中读取参数，就可以快速地完成双线性群的初始化过程。

可以利用Princeton大学封装的文件输出库将初始化后的椭圆曲线对象 `PairingParameters` 封装至 `x.properties` 文件中。后续使用过程中直接从对应配置文件中读取即可还原。代码如下：

```

1  // Type A曲线
2  TypeACurveGenerator pg = new TypeACurveGenerator(rBit, qBit);
3  // Type A1曲线
4  TypeA1CurveGenerator pg = new TypeA1CurveGenerator(numPrimes, qBit);
5  PairingParameters typeAParams = pg.generate();
6  //将参数写入文件a.properties中，使用Princeton大学封装的文件输出库
7  Out out = new Out("a.properties");
8  out.println(typeAParams);
9  //从文件a.properties中读取参数初始化双线性群
10 Pairing pairing = PairingFactory.getPairing("a.properties");

```

随机数内部机制

重点关注椭圆曲线循环群初始化过程中的相关事项。当确定椭圆曲线参数后重复调用 `getG1()`，`newElement()` 和 `newRandomElement()` 方法，验证生成结果是否相同。

```

1  public static void Group_Test(){
2      Pairing bp = PairingFactory.getPairing("a.properties");
3      Field G1 = bp.getG1();
4      Field G3 = bp.getG1();
5
6      Element g_1 = G1.newElement();
7      Element g_2 = G1.newElement();
8
9      Element g_3 = G1.newRandomElement();
10     Element g_4 = G1.newRandomElement();
11
12     if(G1.equals(G3)){
13         System.out.println("YES1!!!");
14     }
15     else
16         System.out.println("Nope!!!");
17     if(g_1.equals(g_2)){

```

```

18         System.out.println("YES2!!!");
19     }
20     else
21         System.out.println("Nope!!!");
22     if(g_3.equals(g_4)){
23         System.out.println("YES3!!!");
24     }
25     else
26         System.out.println("Nope!!!");
27 }

```

运行以上程序，结果如下：

```

"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
YES1!!!
YES2!!!
Nope!!!

```

可以看出，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 G_1 ，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。

工具函数

该部分总结利用JPBC库编写算法仿真程序过程中需要用到的工具函数。代码如下：

String&Byte

```

1  //16进制的byte[]数组转换为字符串
2  public static String hexBytesToString(byte[] bytes) {
3      char[] hexChars = new char[bytes.length * 2];
4      for (int j = 0; j < bytes.length; j++) {
5          int v = bytes[j] & 0xFF;
6          hexChars[j * 2] = HEX_ARRAY[v >>> 4];
7          hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
8      }
9      return new String(hexChars);
10 }
11
12 //16进制的字符串转换为byte[]数组

```

```

13 public static byte[] hexStringToBytes(String s) {
14     int len = s.length();
15     byte[] data = new byte[len / 2];
16     for (int i = 0; i < len; i += 2) {
17         data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)
18     }
19     return data;
20 }

```

获取特定元素

```

1 //G1中获取随机元素, 获取1, 获取0
2 public static Element getRandomFromG1() {
3     return pairing.getG1().newRandomElement().getImmutable();
4 }
5 public static Element getOneFromG1() {
6     return pairing.getG1().newOneElement().getImmutable();
7 }
8 public static Element getZeroFromG1() {
9     return pairing.getG1().newZeroElement().getImmutable();
10 }
11 //Zr中获取随机元素, 获取1, 获取0
12 public static Element getRandomFromZp() {
13     return pairing.getZr().newRandomElement().getImmutable();
14 }
15 public static Element getOneFromZp() {
16     return pairing.getZr().newOneElement().getImmutable();
17 }
18 public static Element getZeroFromZp() {
19     return pairing.getZr().newZeroElement().getImmutable();
20 }

```

哈希映射

◦ $H_0 : 0, 1^* \rightarrow Z_p$

```

1 public static Element hashFromStringToZp(String str) {
2     return pairing.getZr().newElement().setFromHash(str.getBytes(), 0, str.le
3 }
4 public static Element hashFromBytesToZp(byte[] bytes) {
5     return pairing.getZr().newElement().setFromHash(bytes, 0, bytes.length).g
6 }

```

◦ $H_1 : 0, 1^* \rightarrow G_1$

```

1 public static Element hashFromStringToG1(String str) {

```

```

2     return pairing.getG1().newElement().setFromHash(str.getBytes(), 0, str.le
3 }
4 public static Element hashFromBytesToG1(byte[] bytes) {
5     return pairing.getG1().newElement().setFromHash(bytes, 0, bytes.length).g
6 }

```

◦ $H_2 : G_1 \rightarrow Z_p$

```

1 public static Element hashFromG1ToZp( Element g1_element) {
2     // h(y) : G1 -> Zp
3     byte[] g1_bytes = g1_element.getImmutable().toCanonicalRepresentation();
4     try {
5         MessageDigest hasher = MessageDigest.getInstance("SHA-512");
6         zp_bytes = hasher.digest(g1_bytes);    //先把G1元素hash成512bits
7     } catch (Exception e) {
8         e.printStackTrace();
9     }
10    //再把hash后的bits映射到Zp
11    Element hash_result = pairing.getZr().newElementFromHash(zp_bytes, 0, zp
12    return hash_result;
13 }

```

◦ $H_{ch} : G_T \rightarrow Z_p$

```

1 public static Element transformFromGtToZp(Element pairing_result){
2     BigInteger pairing_params = pairing_result.toBigInteger();
3     return pairing.getZr().newElement().set(pairing_params);
4 }

```

Element I/O

- 定义 `writeElement(Element elem, String filename, Pairing pairing)` 函数，实现将 `Element` 对象写入文件。该函数的三个参数分别为待写入的 `Element` 对象，写入文件路径以及对象所在椭圆曲线。返回结果为 `void` 类型。

```

1 public static void writeElement(Element elem, String filename, Pairing pairi
2     DataOutputStream dOut = new DataOutputStream(new FileOutputStream("Param
3     dOut.writeBoolean(elem == null);
4     if (elem == null) {
5         return;
6     }
7     dOut.writeInt(pairing.getFieldIndex(elem.getField()));
8     byte[] bytes = elem.toBytes();
9     dOut.writeInt(bytes.length);
10    dOut.write(bytes);
11    // this is a workaround because it.unisa.dia.gas.plaf.jpbc.field.curve.C

```

```

12     dOut.writeBoolean(elem instanceof CurveElement && elem.isZero());
13     if (elem instanceof CurveElement && elem.isZero()) {
14         throw new IOException("Infinite element detected. They should not ha
15     }
16 }

```

- 定义 `readElement(String filename, Pairing pairing)` 函数，实现从文件中读取 `Element` 对象。该函数的两个参数为读取文件路径和对象所在椭圆曲线，返回结果为 `Element` 类型。

```

1  public static Element readElement(String filename, Pairing pairing) throws I
2      DataInputStream dIn = new DataInputStream(new FileInputStream("Parameter
3      if (dIn.readBoolean()) {
4          return null;
5      }
6      int fieldIndex = dIn.readInt(); // TODO: check if this is in a sensible
7      int length = dIn.readInt(); // TODO: check if this is in a sensible rang
8      byte[] bytes = new byte[length];
9      dIn.readFully(bytes); // throws an exception if there is a premature EOF
10     Element e = pairing.getFieldAt(fieldIndex).newElementFromBytes(bytes);
11     boolean instOfCurveElementAndInf = dIn.readBoolean();
12     if (instOfCurveElementAndInf) {
13         //e.setToZero(); // according to the code this simply sets the infFl
14         throw new IOException("The point is infinite. This shouldn't happen.
15     }
16     return e;
17 }

```



函数运行效率

结合文章《jPBC: java Pairing Based Cryptography》，比较jPBC和PBC之间的运算效率。用于比较效率的计算机配置为Intel® Core™2 Quad CPU Q6600, 2.40GHz, 3 GB 内存, Ubuntu 10.04系统。JDK版本是Oracle jdk1.6.0 20。结果如下。

Operation	jPBC	jPBC with preprocessing	PBC
Pairing#pairing(\cdot, \cdot)	14.654	7.234	2.688
Element#pow(\cdot) in \mathbb{G}_1	18.592	2.841	4.122
Element#pow(\cdot) in \mathbb{G}_2	18.796	2.813	4.008
Element#pow(\cdot) in \mathbb{G}_T	2.112	0.365	0.529
Element#pow(\cdot) in \mathbb{Z}_r	0.068	0.021	0.087

可以看出，由于Java语言特性的限制，JPBC库在处理乘法循环群上运算及配对运算方面效率远低于PBC库，但在处理整数循环群上运算方面效率高于PBC库。显然，可以通过预处理的方法提高JPBC库对应函数的运行效率。

请作者吃个小鱼饼干吧
打赏

 JPBC  Pairing Based Cryptography

[◀ 剑指offer刷题记录（一）](#) [汇编语言学习笔记（一） ▶](#)

昵称	邮箱	网址(http://)
<div>一起来进我的妙妙屋吧!</div> <div></div> <div> </div> <div>提交</div>		

来发评论吧~

Powered By [Valine](#)
v1.4.18

© 2021  Blank

 访客数量: 由 [Hexo](#) 强力驱动 | 主题 – [NexT.Gemini v5.1.4](#) 博客全站共170.3k字

