# JPBC-BBS04

Using JPBC(Java Pairing-Based Cryptography Library) to implement pairing-based cryptography algorithm like BBS04 group signature. And I proposed a modification of batch verification to the original BBS04.

If there exists formula displaying mistakes, check the pdf of README also.

## How to use JPBC

please refer to the official website of JPBC: [http://gas.dia.unisa.it/projects/jpbc/index.html](http://gas.dia.unisa.it/projects/jpbc/index.html)

Download the JPBC Library from the official website, and import the jar package you need to the project. `jpbc-api-2.0.0.jar` and `jpbc-plaf-2.0.0.jar` are mostly used.

Also copy the parameter file under the project directory, which is used to generate bilinear group.

Notice that Type A pairing is symmetric.

### Something that need to be noticed

See the following example that initializes a new element without calling getImmutable()

```
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Element;

public class JPBCDemo {
```

```java
    public static void main(String[] args) {
        Pairing bp =
PairingFactory.getPairing("a.properties");
        Field G1=bp.getG1();
        Field Zr=bp.getZr();

        Element g=G1.newRandomElement();
        Element a=Zr.newRandomElement();
        System.out.println(g);
        System.out.println(a);
        Element g_a=g.powZn(a);  //the value "g" will be
changed to g_a
        System.out.println(g);
        System.out.println(a);
        System.out.println(g_a);
    }
}
```

Although $a$ remains, the value of $g$ will be changed to $g^a$

***method to avoid:***

- call getImmutable() when defining g
  `Element g = pairing.getG1().newRandomElement().getImmutable();`
- or call duplicate() when using g
  `Element ga = g.duplicate().powZn(a);`

I would tend to use `getImmutable()`

# BBS04 group signature scheme

the original paper: [https://crypto.stanford.edu/~dabo/pubs/papers/groupsigs.pdf](https://crypto.stanford.edu/~dabo/pubs/papers/groupsigs.pdf)

# Process of scheme

## 1. System Initialization

Assume bilinear groups ( $G_1, G_2$ ), and $G_1, G_2$ are two multiplicative cyclic groups of prime order $p$. The number of group member is $n$.

## 2. Key Generation

- choose a generator $g_1$ from $G_1$, choose a generator $g_2$ from $G_2$
- select $h \in G_1 \setminus \{1_{G_1}\}$ (1 stands for the identity element of $G_1$, so $h$ is the element in $G_1$ except identity element 1), choose $\xi_1, \xi_2 \in {Z_p}^*$ ; select $u, v \in G_1$ so that $u^{\xi_1} = v^{\xi_2} = h$
- choose $\gamma \in {Z_p}^*$, compute $\omega = {g_2}^\gamma$
- ***group public key is*** $(g_1, g_2, h, u, v, \omega)$ , ***group manger private key is*** $(\xi_1, \xi_2)$

For each group member $i$, group manger select $x_i \in {Z_p}^*$, ensuring every member's $x_i$ is different from each other, and set $A_i = g_1^{\frac{1}{\gamma + x_i}}$ , thus ***the private key of each group member is*** $(A_i, x_i)$.

Therefore, the group manger know the private key of each group member. As a result, manger may forge member's signature theoretically. So the group manger needs to be trustable.

## 3. Signing

Given a group public key $(g_1, g_2, h, u, v, \omega)$, user private key $(A_i, x_i)$, message $M$.

For a certain member whose private key is $(A, x)$, compute the signature as follows:

1. randomly choose $\alpha, \beta \in Z_p$, compute $T_1 = u^\alpha, T_2 = v^\beta, T_3 = Ah^{\alpha+\beta}$

2. randomly choose$r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \in Z_p$, compute $R_1 = u^{r_\alpha}$, $R_2 = v^{r_\beta}$, $R_3 = e(T_3, g_2)^{r_x} \cdot e(h, \omega)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}$, $R_4 = {T_1}^{r_x} \cdot u^{-r_{\delta_1}}$, $R_5 = {T_2}^{r_x} \cdot v^{-r_{\delta_2}}$
3. compute the hash value $c = H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in Z_p$
4. compute $s_\alpha = r_\alpha + c\alpha$, $s_\beta = r_\beta + c\beta$, $s_x = r_x + cx$, $s_{\delta_1} = r_{\delta_1} + c\delta_1$, $s_{\delta_2} = r_{\delta_2} + c\delta_2$, where $\delta_1 = x\alpha, \delta_2 = x\beta$, and the signature is $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$

### 4. Verification

compute $\bar{R}_1 = u^{s_\alpha} \cdot T_1^{-c}$, $\bar{R}_2 = v^{s_\beta} \cdot T_2^{-c}$,
$\bar{R}_3 = e(T_3, g_2)^{s_x} \cdot e(h, \omega)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (\frac{e(T_3, \omega)}{e(g_1, g_2)})^c$,

$\bar{R}_4 = T_1^{s_x} \cdot u^{-s_{\delta_1}}$, $\bar{R}_5 = T_2^{s_x} \cdot v^{-s_{\delta_2}}$

verify whether $c = H(M, T_1, T_2, T_3, \bar{R}_1, \bar{R}_2, \bar{R}_3, \bar{R}_4, \bar{R}_5)$

if equals, then the signature is valid. Otherwise, invalid.

### 5. Open

Compute $A = \frac{T_3}{T_1^{\xi_1} \cdot T_2^{\xi_2}}$ so as to trace the member who signed the group signature

## Approach to map hash value to group element

Since bilinear mapping is initially used in Identity-Based Encryption system, we often need to hash a specific string or byte array into bilinear group.

During the process of signing and verification, hash function is indispensable. In the process of signing, hash the message with other variables, and then obtain the hash value $c$, which takes part in the subsequent computation, such as computing $s_\alpha$. If after hash $c$ is mapped to an interger or string, then $c$ can't proceed to be involved in computation any longer due to the error of operand type dismatch. The parameter type should be `Element`(JPBC), not `int`, so the hash value should be mapped to Z group, transforming into Element type.

```
int c_sign=M_sign.hashCode();  //M_sign is String type
byte[] c_sign_byte = Integer.toString(c_sign).getBytes();
Element c = (Zr.newElementFromHash(c_sign_byte, 0,
c_sign_byte.length)).getImmutable();
```

After hashing the message, transform int to byte array, then call `newElementFromHash()` method to generate a corresponding element of $Z_p$ group. Thus the hash value is mapped to group element.

`newElementFromHash()` can map the hash value into $G$ group as well, as long as the object calling the method is an element of $G$ group.

# A proposal of batch verification

## An improtant property of bilinear pairing

$$e(ab, c) = e(a, c)e(b, c)$$

prove:

let $a = {g_1}^\alpha, b = {g_1}^\beta, c = {g_2}^\gamma$ ($a$ and $b$ must from the same group, while $c$ not necessarily)

$e({g_1}^\alpha {g_1}^\beta, {g_2}^\gamma)$

$= e({g_1}^{\alpha+\beta}, {g_2}^\gamma)$

$= e(g_1, g_2)^{(\alpha+\beta)\gamma}$

$$= e(g_1, g_2)^{\alpha\gamma+\beta\gamma}$$

$$= e(g_1, g_2)^{\alpha\gamma} \cdot e(g_1, g_2)^{\beta\gamma}$$

$$= e({g_1}^{\alpha}, {g_2}^{\gamma})e({g_1}^{\beta}, {g_2}^{\gamma})$$

When computing two bilinear pairings with the same $c$, we can only compute one bilinear pairing through the property, thus decreasing the computation of bilinear pairing, which is far more time-consuming than multiplication.

## Simplify the formula of computing $\bar{R}_3$

During the verification, computing $\bar{R}_3$ is the most time-consuming, which have to do paring 5 times. With the help of the above property, it can be simplified to only 2 times.

$$e(T_3, g_2)^{s_x} \cdot e(h, \omega)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (\frac{e(T_3, \omega)}{e(g_1, g_2)})^c$$

$$= e({T_3}^{s_x}, g_2) \cdot e(h^{-s_\alpha - s_\beta}, \omega) \cdot e(h^{-s_{\delta_1} - s_{\delta_2}}, g_2) \cdot e({T_3}^c, \omega) \cdot e({g_1}^{-c}, g_2)$$

$$= e({T_3}^{s_x} \cdot h^{-s_{\delta_1} - s_{\delta_2}} \cdot {g_1}^{-c}, g_2) \cdot e(h^{-s_\alpha - s_\beta} \cdot {T_3}^c, \omega)$$

Then consider the case that the number of signatures is $n$, still apply the above property, we'll have :

$$\prod_{i=1}^{n} [e({T_{3}}_i^{s_{xi}} \cdot h^{-s_{\delta_1}{}_i - s_{\delta_2 i}} \cdot {g_1}^{-c_i}, g_2) \cdot e(h^{-s_{\alpha i} - s_{\beta_i}} \cdot {T_3}^{c_i}, \omega)]$$

$$= e(\prod_{i=1}^{n} {T_{3}}_i^{s_{xi}} \cdot h^{-s_{\delta_1}{}_i - s_{\delta_2 i}} \cdot {g_1}^{-c_i}, g_2) \cdot e(\prod_{i=1}^{n} h^{-s_{\alpha i} - s_{\beta_i}} \cdot {T_3}^{c_i}, \omega)$$

# Batch verification of BBS04 scheme

Since the BBS04 group signature scheme is based on bilinear pairing, and pairing operation is about 1500 times as time-consuming as multiplication. If we receive several signatures and verify them separately, it will takes even more time.

What if we verify a batch of signatures simultaneously?

In this way, we can decrease the number of times to do paring from 5n to 2, however big n is. Thus saving a lot of time for verification.

**A modification of batch verification as follow :**

Group public key is $(g_1, g_2, h, u, v, \omega)$

Assume now we have $n$ signatures to verify, and the signatures are separately $\sigma_i = (T_{1i}, T_{2i}, T_{3i}, c_i, R_{1i}, R_{2i}, R_{3i}, R_{4i}, R_{5i}, s_{\alpha i}, s_{\beta_i}, s_{x i}, s_{\delta_1 i}, s_{\delta_2 i})$ $(1 \leq i \leq n)$

for each $i \in [1, n]$

- firstly verify whether the four equations are true

$$u^{s_{\alpha i}} \cdot {T_1}_i^{-c_i} = R_{1i}$$

$$v^{s_{\beta_i}} \cdot {T_2}_i^{-c_i} = R_{2i}$$

$${T_1}_i^{s_{xi}} \cdot u^{-s_{\delta_1 i}} = R_{4i}$$

$${T_2}_i^{s_{xi}} \cdot v^{-s_{\delta_2 i}} = R_{5i}$$

- then verify the equation

$$e(\prod_{i=1}^{n} {T_3}_i^{s_{xi}} \cdot h^{-s_{\delta_1 i} - s_{\delta_2 i}} \cdot {g_1}^{-c_i}, g_2) \cdot e(\prod_{i=1}^{n} h^{-s_{\alpha i} - s_{\beta_i}} \cdot {T_3}^{c_i}, \omega) = \prod_{i=1}^{n} R_{3i}$$

If and only if the above equations are true, these n signatures can successfully pass the verification together.