任务三报告

蒋嵩林

对椭圆曲线计算原理的理解

基本原理

椭圆曲线相关密钥的安全性基于一个事实，即没有亚指数时间算法来解决椭圆曲线上的离散对数问题。

设 p 是一个素数，素域F_p表示{0, 1, 2, ... , p−1}共计 p 个元素。当 p 是大于 3 的素数时，Fp 上椭圆曲线方程在仿射坐标系下可以表示为：$y^2 = x^3 + a*x + b$，对应点的坐标为二维的(x, y)。

SM2 对应的 sm2p256v1 曲线便是符合上述定义的素域 256 位椭圆曲线，其对应的 p, a, b 值如下：

```
p=FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF
FFFFFFFF
a=FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF
FFFFFFFC
b=28E9FA9E 9D9F5E34 4D5A9E4B CF6509A7 F39789F5 15AB8F92 DDBCBD41
4D940E93
```

私钥

对于 sm2p256v1 曲线，私钥为一个 256 位的随机数，其值的范围在$[1, n-2]$，其中 n 的值为：

```
FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409
39D54123
```

实测当私钥取到 n-1 时使用 bc 签名会报错：

```
Private Key: FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54122
Public Key (Uncompressed): 0432C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C743C8C95C0B098863A642311C949
6DEAC2F56788239D5B8C0FD20CD1ADEC60F5F
Public Key: 0332C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
To sign: How are you?
Exception in thread "main" java.lang.ArithmeticException: BigInteger not invertible.
        at org.bouncycastle.util.BigIntegers.modOddInverse(Unknown Source)
        at org.bouncycastle.crypto.signers.SM2Signer.generateSignature(Unknown Source)
        at org.bouncycastle.jcajce.provider.asymmetric.ec.GMSignatureSpi.engineSign(Unknown Source)
        at java.base/java.security.Signature$Delegate.engineSign(Signature.java:1403)
        at java.base/java.security.Signature.sign(Signature.java:712)
        at org.sample.Test.sign(Test.java:44)
        at org.sample.Test.main(Test.java:80)
```

这个问题只会影响 SM2 签名算法，而不会影响到 SM2 加密算法。

（下段问题原因探究来自江莎老师，在此谢谢他的帮助）

国标定义的 SM2 签名的生成算法如下图所示，

A.1.2.2 仿射坐标表示

当p是大于3的素数时，F_p上椭圆曲线方程在仿射坐标系下可以简化为$y^2 = x^3 + ax + b$，其中$a,b \in F_p$，且使得$(4a^3 + 27b^2) \bmod p \neq 0$。椭圆曲线上的点集记为$E(F_p) = \{(x,y)|x,y \in F_p$且满足曲线方程$y^2 = x^3 + ax + b\} \bigcup \{O\}$，其中$O$是椭圆曲线的无穷远点。

$E(F_p)$上的点按照下面的加法运算规则，构成一个阿贝尔群：

a) $O + O = O$；

b) $\forall P = (x,y) \in E(F_p) \backslash \{O\}$，$P + O = O + P = P$；

---

c) $\forall P = (x,y) \in E(F_p) \backslash \{O\}$，$P$的逆元素$-P = (x,-y)$，$P + (-P) = O$；

d) 点$P_1 = (x_1,y_1) \in E(F_p) \backslash \{O\}$，$P_2 = (x_2,y_2) \in E(F_p) \backslash \{O\}$，$P_3 = (x_3,y_3) = P_1 + P_2 \neq O$，则

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \ , \\ y_3 = \lambda(x_1 - x_3) - y_1, \end{cases}$$

其中

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{若} x_1 \neq x_2, \\ \dfrac{3{x_1}^2 + a}{2y_1}, & \text{若} x_1 = x_2 \text{且} P_2 \neq -P_1 \text{。} \end{cases}$$

标准射影坐标系

使用射影坐标解决了上述问题，将点变换为三维的进行运算。在标准射影坐标中，点(x,y,z)对应仿射坐标中的点$\frac {x}{z},\frac {y}{z}$，当 z=0 时，则表示无穷远点。[1]

A.1.2.3.1 标准射影坐标系

当p是大于3的素数时，F_p上椭圆曲线方程在标准射影坐标系下可以简化为$y^2z=x^3+axz^2+bz^3$，其中$a,b\in F_p$，且$4a^3+27b^2\neq 0 \bmod p$。椭圆曲线上的点集记为$E(F_p)=\{(x,y,z)|x,y,z\in F_p$且满足曲线方程$y^2z=x^3+axz^2+bz^3\}$。对于$(x_1,y_1,z_1)$和$(x_2,y_2,z_2)$，若存在某个$u\in F_p$且$u\neq 0$，使得：$x_1=ux_2$，$y_1=uy_2$，$z_1=uz_2$，则称这两个三元组等价，表示同一个点。

若$z\neq 0$，记$X=x/z$，$Y=y/z$，则可从标准射影坐标表示转化为仿射坐标表示：$Y^2=X^3+aX+b$；

若$z=0$，(0,1,0)对应的仿射坐标系下的点即无穷远点O。

标准射影坐标系下，$E(F_p)$上点的加法运算定义如下：

a) $O+O=O$；

b) $\forall P=(x,y,z)\in E(F_p)\backslash\{O\}$，$P+O=O+P=P$；

c) $\forall P=(x,y,z)\in E(F_p)\backslash\{O\}$，$P$的逆元素$-P=(ux,-uy,uz)$，$u\in F_p$且$u\neq 0$，$P+(-P)=O$；

d) 设点$P_1=(x_1,y_1,z_1)\in E(F_p)\backslash\{O\}$，$P_2=(x_2,y_2,z_2)\in E(F_p)\backslash\{O\}$，$P_3=P_1+P_2=(x_3,y_3,z_3)\neq O$，

若$P_1\neq P_2$，则：

$\lambda_1=x_1z_2$，$\lambda_2=x_2z_1$，$\lambda_3=\lambda_1-\lambda_2$，$\lambda_4=y_1z_2$，$\lambda_5=y_2z_1$，$\lambda_6=\lambda_4-\lambda_5$，

$\lambda_7=\lambda_1+\lambda_2$，$\lambda_8=z_1z_2$，$\lambda_9=\lambda_3^2$，$\lambda_{10}=\lambda_3\lambda_9$，$\lambda_{11}=\lambda_8\lambda_6^2-\lambda_7\lambda_9$，$x_3=\lambda_3\lambda_{11}$，

$y_3=\lambda_6(\lambda_9\lambda_1-\lambda_{11})-\lambda_4\lambda_{10}$，$z_3=\lambda_{10}\lambda_8$；

若$P_1=P_2$，则：

$\lambda_1=3x_1^2+az_1^2$，$\lambda_2=2y_1z_1$，$\lambda_3=y_1^2$，$\lambda_4=\lambda_3x_1z_1$，$\lambda_5=\lambda_2^2$，$\lambda_6=\lambda_1^2-8\lambda_4$，

$x_3=\lambda_2\lambda_6$，$y_3=\lambda_1(4\lambda_4-\lambda_6)-2\lambda_5\lambda_3$，$z_3=\lambda_2\lambda_5$。

Jacobian 加重射影坐标系

进一步，使用 Jacobian 加重射影坐标系，可以进一步降低运算复杂度：

表A.6　素域上椭圆曲线加法运算复杂度

运　算	坐　标　系		
	仿射坐标	标准射影坐标	Jacobian加重射影坐标
一般加法	1I+2M+1S	13M+2S	12M+4S
倍　点	1I+2M+2S	8M+5S	4M+6S

A.1.2.3.2 Jacobian加重射影坐标系

F_p上椭圆曲线方程在Jacobian加重射影坐标系下可以简化为$y^2 = x^3 + axz^4 + bz^6$。其中$a, b \in F_p$，且$4a^3 + 27b^2 \neq 0 \bmod p$。椭圆曲线上的点集记为$E(F_p) = \{(x,y,z) | x,y,z \in F_p$且满足曲线方程$y^2 = x^3 + axz^4 + bz^6\}$。对于$(x_1,y_1,z_1)$和$(x_2,y_2,z_2)$，若存在某个$u \in F_p$且$u \neq 0$，使得：$x_1 = u^2 x_2$，$y_1 = u^3 y_2$，$z_1 = uz_2$，则称这两个三元组等价，表示同一个点。

若$z \neq 0$，记$X = x/z^2$，$Y = y/z^3$，则可从Jacobian加重射影坐标表示转化为仿射坐标表示：$Y^2 = X^3 + aX + b$；

若$z = 0$，(1,1,0)对应的仿射坐标系下的点即无穷远点O。

Jacobian加重射影坐标系下，$E(F_p)$上点的加法运算定义如下：

a) $O + O = O$；

b) $\forall P = (x,y,z) \in E(F_p) \backslash \{O\}$，$P + O = O + P = P$；

c) $\forall P = (x,y,z) \in E(F_p) \backslash \{O\}$，$P$的逆元素$-P = (u^2 x, -u^3 y, uz)$，$u \in F_p$且$u \neq 0$，$P + (-P) = O$；

d) 设点$P_1 = (x_1,y_1,z_1) \in E(F_p) \backslash \{O\}$，$P_2 = (x_2,y_2,z_2) \in E(F_p) \backslash \{O\}$，$P_3 = P_1 + P_2 = (x_3,y_3,z_3) \neq O$，

若$P_1 \neq P_2$，则：

$\lambda_1 = x_1 z_2^2$，$\lambda_2 = x_2 z_1^2$，$\lambda_3 = \lambda_1 - \lambda_2$，$\lambda_4 = y_1 z_2^3$，$\lambda_5 = y_2 z_1^3$，$\lambda_6 = \lambda_4 - \lambda_5$，$\lambda_7 = \lambda_1 + \lambda_2$，

$\lambda_8 = \lambda_4 + \lambda_5$，$x_3 = \lambda_6^2 - \lambda_7 \lambda_3^2$，$\lambda_9 = \lambda_7 \lambda_3^2 - 2x_3$，$y_3 = (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2$，$z_3 = z_1 z_2 \lambda_3$；

若$P_1 = P_2$，则：

$\lambda_1 = 3x_1^2 + az_1^4$，$\lambda_2 = 4x_1 y_1^2$，$\lambda_3 = 8y_1^4$，$x_3 = \lambda_1^2 - 2\lambda_2$，$y_3 = \lambda_1(\lambda_2 - x_3) - \lambda_3$，$z_3 = 2y_1 z_1$。

实测按照官方文档使用的 add-1986-cc2 jacobian 加重射影坐标计算方法实现出来的计算效果不如上面官方文档中提供的标准射影坐标系实现，因而最终决定参考 https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html 中最优的 2007 Bernstein–Lange 方法实现。

https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html#addition-add-2007-bl

https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html#doubling-dbl-2007-bl

私钥生成公钥原理

sm2p256v1 曲线规定的基点坐标为：

```
Gx=32C4AE2C 1F198119 5F990446 6A39C994 8FE30BBF F2660BE1 715A4589
334C74C7
Gy=BC3736A2 F4F6779C 59BDCEE3 6B692153 D0A9877C C62A4740 02DF32E5
2139F0A0
```

6 密钥对的生成与公钥的验证

6.1 密钥对的生成

输入：一个有效的F_q($q=p$且p为大于3的素数，或$q=2^m$)上椭圆曲线系统参数的集合。
输出：与椭圆曲线系统参数相关的一个密钥对(d,P)。
a) 用随机数发生器产生整数$d\in[1,n-2]$；
b) G为基点，计算点$P=(x_P,y_P)=[d]G$；(参见附录A.3.2。)
c) 密钥对是(d,P)，其中d为私钥，P为公钥。

由私钥生成公钥的过程，即为对基点进行私钥值的倍点计算[2]，得到的点的 x 坐标和 y 坐标即为公钥的一部分。

将生成的随机数转化为 16 进制形式并补零，即得到了常用的 64 位私钥。

点的字节串表示有多种形式，用一个字节PC加以标识。无穷远点O的字节串表示是单一的零字节$PC=00$。非无穷远点$P=(x_P,y_P)$有如下三种字节串表示形式：
a) 压缩表示形式，$PC=02$或03；
b) 未压缩表示形式，$PC=04$；
c) 混合表示形式，$PC=06$或07。
注：混合表示形式既包含压缩表示形式又包含未压缩表示形式。在实现中，它允许转换到压缩表示形式或者未压缩表示形式。
对于椭圆曲线上点的压缩表示形式和混合表示形式，本文本中定为可选形式。椭圆曲线上点的压缩表示形式参见附录A.5。

未压缩的 16 进制字符串公钥

以`04`开头，然后分别将 x 坐标和 y 坐标的值转换为 16 进制形式并补零，拼接即得到了 130 位未压缩的公钥。

压缩的 16 进制字符串公钥及其还原

当 y 坐标为偶数时，我们以`02`开头，当为奇数时以`03`开头，然后将 x 坐标的值转换为 16 进制形式并补零，拼接即得到了 66 位未压缩的公钥。

压缩密钥还原：

需要的的模素数平方根求解方法：

B.1.4 模素数平方根的求解

设p是奇素数，g是满足$0 \le g < p$的整数，g的平方根(mod p)是整数y，$0 \le y < p$，且$y^2 \equiv g \pmod{p}$。

若$g = 0$，则只有一个平方根，即$y = 0$；若$g \neq 0$，则g有0个或2个平方根(mod$p$)，若y是其中一个平方根，则另一个平方根就是$p - y$。

下面的算法可以确定g是否有平方根(mod$p$)，若有，就计算其中一个根。

输入：奇素数p，整数g，$0 < g < p$。

输出：若存在g的平方根，则输出一个平方根mod$p$，否则输出“不存在平方根”。

算法1：对$p \equiv 3 \pmod 4$，即存在正整数u，使得$p = 4u + 3$。

a) 计算$y = g^{u+1} \bmod p$(参见附录B.1.1)；

b) 计算$z = y^2 \bmod p$；

c) 若$z = g$，则输出y；否则输出“不存在平方根”。

两种方法实现标量乘法

这里密钥对的生成是自制的实现，其基于官方文档的纯数学计算，使用 Jacobian 加重射影坐标系计算，没有其他依赖项。

https://github.com/HollowMan6/My-Tencent-Rhino-Bird-Open-Source-Training-Program/tree/main/2022-Kona-JDK/Task3

根据官方文档[3]提供的三种方法，选取了前两种实现标量乘法。

二进制展开法（Double-and-add / Binary Expand）

A.3.2 椭圆曲线多倍点运算的实现

椭圆曲线多倍点运算的实现有多种方法，这里给出三种方法，以下都假设$1 \le k < N$。

算法一：二进制展开法

输入：点P，l比特的整数$k = \sum_{j=0}^{l-1} k_j 2^j$，$k_j \in \{0, 1\}$。

输出：Q=[k]P。

a) 置$Q = O$；

b) j从$l - 1$下降到0执行：

b.1) $Q = [2]Q$；

b.2) 若$k_j = 1$，则$Q = Q + P$；

c) 输出Q。

加减法

算法二：加减法

输入： 点P，l比特的整数$k=\sum_{j=0}^{l-1}k_j2^j$，$k_j\in\{0,1\}$。

输出： $Q=[k]P$。

a) 设$3k$的二进制表示是$h_rh_{r-1}\cdots h_1h_0$，其中最高位h_r为1；

b) 设k的二进制表示是$k_rk_{r-1}\cdots k_1k_0$，显然$r=l$或$l+1$；

c) 置$Q=P$；

d) 对i从$r-1$下降到1执行：

d.1) $Q=[2]Q$；

d.2) 若$h_i=1$，且$k_i=0$，则$Q=Q+P$；

d.3) 若$h_i=0$，且$k_i=1$，则$Q=Q-P$；

e) 输出Q。

注：减去点(x,y)，只要加上$(x,-y)$（对域F_p），或者$(x,x+y)$（对域F_{2^m}）。有多种不同的变种可以加速这一运算。

修改 SunEC 库实现 sm2p256v1 曲线步骤

https://github.com/HollowMan6/My-Tencent-Rhino-Bird-Open-Source-Training-Program/tree/main/2022-Kona-JDK/Task3-SunEC

另外基于 SunEC 已有的标量乘法实现，添加了 sm2p256v1 曲线支持。

https://github.com/HollowMan6/jdk/tree/sm2

因为`sm2p256v1`也是一条素数域曲线，且其`p-a`同样为 3，就像`secp256r1`一样，所以我们可以模仿 SunEC 库中现有的`secp256r1`实现来帮助我们实现我们的实现。

https://github.com/HollowMan6/My-Tencent-Rhino-Bird-Open-Source-Training-Program/tree/main/2022-Kona-JDK/Task3-SunEC

我们首先将曲线参数填充到 CurveDB，https://github.com/HollowMan6/jdk/blob/c3e924641bb3a838f6abc496dd380ceb619df163/src/java.base/share/classes/sun/security/util/CurveDB.java#L258-L265 。

然后添加 OID 和名称。

最重要的部分是`FieldGen`用于自动生成优化的有限域实现。https://github.com/HollowMan6/jdk/blob/c3e924641bb3a838f6abc496dd380ceb619df163/make/jdk/src/classes/build/tools/intpoly/FieldGen.java

我们需要生成两个字段，“整数多项式(`Integer Polynomial`)”（对应于`p`）和“阶(`Order`)”（对应于`n`）。

因为：

FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFF

=

FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF $2^{256} - 1$

\-

00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 2^{224}

\-

00000000 00000000 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF $2^{96} - 1$

\+

00000000 00000000 00000000 00000000 00000000 00000000 FFFFFFFF FFFFFFFF

$2^{64} - 1$

= $2^{256} - 2^{224} - 2^{96} + 2^{64} - 2^0$

所以整数多项式应该像这样填充。我们可以从`secp256r1`复制其他参数，因为它们都是 256 位素数域曲线。

各种维度的性能比较和原因分析

由于 Bouncy Castle 原生支持 SM2，所以这里直接将它添加进来进行测评：

```
Benchmark
Mode  Cnt    Score    Error  Units
BenchmarkKeyGeneration.sm2p256v1_bc              thrpt   25
16378.547 ±    267.594  ops/s
BenchmarkKeyGeneration.sm2p256v1_sunec           thrpt   25
2648.607 ±     29.897  ops/s
BenchmarkKeyGeneration.sm2p256v1_homemade_compressed_AddMinus
thrpt   25    815.874 ±   8.999  ops/s
BenchmarkKeyGeneration.sm2p256v1_homemade_compressed_BinaryExpans
ion  thrpt   25    730.941 ±   6.298  ops/s
BenchmarkPublicKeys.sm2p256v1_compressed_Addminus
thrpt   25    819.361 ±   1.839  ops/s
BenchmarkPublicKeys.sm2p256v1_uncompressed_Addminus
thrpt   25    799.982 ±   7.037  ops/s
BenchmarkPublicKeys.sm2p256v1_jacob_compressed_Addminus
thrpt   25    829.413 ±   7.376  ops/s
BenchmarkPublicKeys.sm2p256v1_jacob_uncompressed_Addminus         thrpt   25
828.280 ±  14.280  ops/s
```

可以看到，使用 Jacobian 加重射影坐标系相较于标准摄影坐标系而言性能上有着提高，随后，二进制展开法的性能最差，加减法的性能要优于二进制展开法，SunEC 的性能的实现较前两个算法实现有很大程度的提高，Bouncy Castle 的实现性能要远远超过其他所有的算法。

由于二进制展开法（Double-and-add/ Binary Expand）对于不同的私钥，由于对每位为 1 和为 0 时产生了计算时间不相等的分支，所以存在执行时间不同，根据执行时间/功率分析就可以间接推算私钥。同理，对于加减法，当私钥的三倍和私钥本身的位值相同或不同时，也会产生不同的执行时间。另外，对于官方文档提供的另一种滑动窗法，也不能实现对所有私钥的恒定公钥生成时间，所以当执行时间或功耗暴露时，官方文档中提供的所有方法都是存在风险的。

算法三：滑动窗法

输入： 点P，l比特的整数$k=\sum_{j=0}^{l-1}k_j2^j$，$k_j\in\{0,1\}$。

输出： $Q=[k]P$。

设窗口长度$r>1$。

预计算

a) $P_1=P$，$P_2=[2]P$；

b) i从1到$2^{r-1}-1$计算$P_{2i+1}=P_{2i-1}+P_2$；

c) 置$j=l-1$，$Q=O$。

主循环

d) 当$j\geq 0$执行：

d.1) 若$k_j=0$，则$Q=[2]Q$，$j=j-1$；

d.2) 否则

d.2.1) 令t是使$j-t+1\leq r$且$k_t=1$的最小整数；

d.2.2) $h_j=\sum_{i=0}^{j-t}k_{t+i}2^i$；

d.2.3) $Q=[2^{j-t+1}]Q+P_{h_j}$；

d.2.4) 置$j=t-1$；

e) 输出Q。

进一步的，维基百科中[4]提供的 w-ary 非相邻形式 (wNAF) 方法和蒙哥马利梯，尽管是恒定时间的，但是由于依赖于私钥中每一位的值来选择分支执行相关操作[5]，所以均可利用 Flush-Reload[6]进行侧信道攻击获取私钥值[7]，因而维基百科中提供的所有方法也是存在风险的。

SunEC 的相关实现基于论文[9]"Complete addition formulas for prime order elliptic curves"[10]，查看其源代码[11]，可知其也不存在依赖于私钥的选择分支，因而也是侧信道攻击安全的。结合之前的修改 SunEC 库实现 sm2p256v1 曲线经验，可知其使用了高效且恒定时间的模运算库[12]，因而大大提高了运行时运算性能同时也是十分安全的。

Bouncy Castle 的相关实现源代码[8]可以看到，Fixed Point Comb Multiplier 不存在依赖于私钥的选择分支，因而初步分析是侧信道攻击安全的。BC 其通过标量乘法预运算，在密钥生成时直接查询预计算结果，从而相对于没有使用预计算的 SunEC 性能有了极大的提高。这里存在一个较难实现的点是表格存取也要是定时的问题。如果不是定时的，则仍然存在被攻击的

可能性（BC 的预计算表格存取实现是否为定时的尚不清楚）。（谢谢范学雷老师对原理的解读）

在代码实现中，需要关注的安全问题

生成私钥时

必须使用 SecureRandom 生成私钥。Random 一般以当前时间作为种子，而对于密码学而言，由于密钥的生成时间是可知的，因而这是极其不安全的。而 SecureRandom 通过读取操作系统生成的熵作为种子，如类 Unix 操作系统上从 /dev/random 读取，因而是相对安全的。

生成公钥时[13]

1. 为恒定时间，且不会根据私钥的内容进行分支，从而确保私钥免受缓存和定时侧通道攻击的影响（最重点）。

2. 最好使用全 Java 实现，减少内存错误或泄露的威胁。

参考

1. https://blog.csdn.net/qq_50680426/article/details/120940244

2. https://www.zhihu.com/question/22399196/answer/96016340

3.https://www.oscca.gov.cn/sca/xxgk/2010-12/17/1002386/files/b791a9f908bb4803875ab6aeeb7b4e03.pdf

4. https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication

5. https://www.rfc-editor.org/rfc/rfc7748.html#page-10

6. https://zhuanlan.zhihu.com/p/32848483

7. https://eprint.iacr.org/2014/140

8. https://github.com/bcgit/bc-java/blob/master/core/src/main/java/org/bouncycastle/math/ec/FixedPointCombMultiplier.java

9. https://bugs.openjdk.org/browse/JDK-8208698

10. https://eprint.iacr.org/2015/1060.pdf

11. https://github.com/openjdk/jdk/blob/master/src/jdk.crypto.ec/share/classes/sun/security/ec/ECOperations.java#L231

12. https://bugs.openjdk.org/browse/JDK-8181594

13. https://openjdk.org/jeps/8204574