

ECC (Criptografia amb corbes el.líptiques)

Sigui p un primer > 3 . Una **corba el.líptica** definida sobre el cos finit \mathbb{F}_p és una corba plana donada per una equació

$$y^2 = x^3 + ax + b$$

tal que a, b són enters mòdul p i $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, més un punt de l'infinit \mathbf{O} .

El conjunt de punts de la corba és

$$E(\mathbb{F}_p) = \{(x, y) \mid x, y \text{ enters mòdul } p \text{ que satisfan l'equació}\} \cup \{\mathbf{O}\}$$

Amb vistes a la implementació, el punt \mathbf{O} es pot representar mitjançant tres coordenades, de manera que la tercera sigui un 0.

En aquest conjunt es defineix una llei d'addició mitjançant la condició: tres punts sumen \mathbf{O} si i només si estan alineats. Concretament, la **suma de punts** s'expressa de la manera següent: si $P = (x_1, y_1)$ és un punt de la corba, posem $-P = (x_1, -y_1 \pmod{p})$. Aleshores,

- $P + \mathbf{O} = P$
- $P + (-P) = \mathbf{O}$
- si $Q = (x_2, y_2) \neq -P$, aleshores $P + Q = (x', \lambda(x_1 - x') - y_1 \pmod{p})$, on $x' = \lambda^2 - x_1 - x_2 \pmod{p}$ i

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} \pmod{p} & \text{si } Q \neq P, \\ (3x_1^2 + a)(2y_1)^{-1} \pmod{p} & \text{si } Q = P. \end{cases}$$

Aquesta operació dóna estructura de grup abelià al conjunt de punts de la corba el.líptica: la suma és associativa i commutativa, té element neutre (el \mathbf{O}) i cada element P té oposat $-P$ (l'oposat de \mathbf{O} és ell mateix).

En aquest grup, l'*exponenciació* kP es realitza mitjançant l'**algoritme de "quadrats" successius**: es considera l'expressió binària de k , $k = b_r 2^r + \dots + b_1 2 + b_0$, on $b_i \in \{0, 1\}$ i $b_r = 1$; llavors

```

R ← O, i ← r
mentre i ≥ 0 fer
    R ← R + R
    si bi = 1 fer R ← R + P
    i ← i - 1
sortida R
    
```

En les corbes el.líptiques que s'utilitzen en Criptografia, el grup de punts $E(\mathbb{F}_p)$ té cardinal $2^m n$, on n és un nombre *primer* i el cofactor $h = 2^m$ té exponent $0 \leq m \leq 16$. En aquestes condicions, sempre existeix algun punt G d'ordre n , és a dir, tal que $nG = \mathbf{O}$ però totes els múltiples anteriors $G, 2G, 3G, \dots, (n-1)G$ són $\neq \mathbf{O}$.

Fixats els paràmetres p, a, b, n, G d'un sistema criptogràfic, la **clau privada** de cada usuari serà un enter aleatori r , $1 < r < n - 1$, i la **clau pública** serà el punt $P = rG$.

Intercanvi de claus Diffie-Hellman. Si les claus privada i pública de dos usuaris són $(r_1, P_1 = r_1G)$ i $(r_2, P_2 = r_2G)$, aleshores el primer usuari pot calcular r_1P_2 i el segon usuari pot calcular r_2P_1 , de manera que tots dos estan calculant el mateix punt: $r_1r_2G = (x, y)$

Això es pot fer servir per a que tots dos obtinguin una clau secreta de 256 bits, per exemple per utilitzar amb l'AES:

$$K = \text{SHA256}(s||x)$$

on s és un nombre aleatori que es poden intercanviar de manera pública i $||$ indica la concatenació de cadenes binàries.

ECDSA (Elliptic Curve Digital Signature Algorithm). Amb els paràmetres p, a, b, n, G com abans, la signatura d'un missatge M per part de l'usuari que té clau privada r i clau pública P es fa de la manera següent:

- $kG = (x_1, y_1)$ amb $1 < k < n - 1$ aleatori
- $f_1 = x_1 \bmod n$
- $f_2 = k^{-1} (H(M) + f_1r) \bmod n$
- **Enviar la signatura** (f_1, f_2)

Si $f_1 = 0$ o $f_2 = 0$ s'ha de generar un nou valor de k i tornar al primer pas. Observem que si el primer p té longitud ℓ , aleshores la signatura té longitud $\leq 2\ell$.

La verificació per part del receptor consisteix a fer el següent:

- $w_1 = H(M)f_2^{-1} \bmod n$
- $w_2 = f_1f_2^{-1} \bmod n$
- $w_1G + w_2P = (x_0, y_0)$
- **Acceptar** si $x_0 \bmod n = f_1$

En aquest algoritme hem indicat per $H(M)$ un hash del missatge M . En la implementació considerarem que el primer p té 256 bits i que la funció hash utilitzada és *SHA256*. En particular, podrem expressar la signatura amb 64 bytes.

Implementació: signatures. Definiu la classe `ecc` amb els següents mètodes:

```
public static BigInteger [] invers( BigInteger [] P, BigInteger[] ParametresCorba)
```

- entrada: P punt de la corba donat per 3 coordenades (x, y, z) , (si $z = 0$ és el punt de l'infinit), `ParametresCorba` = $\{a, b, p\}$, corresponents a la corba $y^2 = x^3 + ax + b \bmod p$;
 sortida: una llista $\{R_x, R_y, R_z\}$ que representa l'invers de P , $R = -P$ (si $R_z = 0$, és el punt de l'infinit).

```
public static BigInteger [] suma( BigInteger [] P, BigInteger [] Q, BigInteger[] ParametresCorba)
```

- entrada: P i Q punts de la corba donats per 3 coordenades (x, y, z) , (si $z = 0$ és el punt de l'infinit), `ParametresCorba` = $\{a, b, p\}$, corresponents a la corba $y^2 = x^3 + ax + b \bmod p$;
 sortida: una llista $\{R_x, R_y, R_z\}$ que representa el punt $R = P + Q$ (si $R_z = 0$, és el punt de l'infinit).

```
public static BigInteger [] multiple(BigInteger k, BigInteger [] P, BigInteger[] ParametresCorba)
```

entrada: **k** enter,
P punt de la corba donats per 3 coordenades (x, y, z) , (si $z = 0$ és el punt de l'infinit),
ParametresCorba = $\{a, b, p\}$, corresponents a la corba $y^2 = x^3 + ax + b \pmod p$;
sortida: una llista $\{R_x, R_y, R_z\}$ que representa el punt $R = P + \dots + P = k \cdot P$ (si $R_z = 0$, és el punt de l'infinit).

```
public static BigInteger[] clausECC(BigInteger[] parametresECC)
```

entrada: **parametresECC** = $\{n, G_x, G_y, a, b, p\}$, $G = (G_x, G_y)$ punt d'ordre n de la corba $y^2 = x^3 + ax + b \pmod p$ (evidentment, G no és el punt de l'infinit);
sortida: una llista $\{r, P_x, P_y\}$, r és la clau privada, i (P_x, P_y) punt (diferent del punt de l'infinit) que és la clau pública.

```
public static BigInteger ECCDHKT(byte[] bytesAleatoris, BigInteger clauPrivadaECC,  
                                BigInteger[] clauPublicaECC, BigInteger[] parametresECC)
```

entrada: **bytesAleatoris** és una llista de bytes aleatoris,
clauPrivadaECC és un enter,
clauPublicaECC = $\{P_x, P_y\}$ (diferent del punt de l'infinit)
parametresECC = $\{n, G_x, G_y, a, b, p\}$, $G = (G_x, G_y)$ punt d'ordre n de la corba $y^2 = x^3 + ax + b \pmod p$ (evidentment, G no és el punt de l'infinit);
sortida: una clau secreta de 256 bits pel AES: Amb **clauPrivadaECC** i **clauPublicaECC** es calcula una clau DH de components (x, y) . La clau secreta k es calcula $k = \text{sha256}(\text{bytesAleatoris} || x)$, x en bytes (sense complement a 2).

```
public static byte[] firmarECCDSA(byte[] M, BigInteger clauFirma, BigInteger[] parametresECC)
```

entrada: **M** és una cadena de bytes de longitud arbitrària que és el missatge per firmar,
clauFirma és un enter que és la clau privada del firmant
parametresECC = $\{n, G_x, G_y, a, b, p\}$, $G = (G_x, G_y)$ punt d'ordre n de la corba $y^2 = x^3 + ax + b \pmod p$ (evidentment, G no és el punt de l'infinit);
sortida: una cadena de bytes que sigui el missatge firmat; és la concatenació de la cadena **M** amb una cadena d'exactament 64 bytes que representen la firma.

```
public static boolean verificarECCDSA(byte[] MS, BigInteger[] clauVer, BigInteger[] parametresECC)
```

entrada: **MS** és un missatge (suposadament) firmat amb el sistema ECCDSA de paràmetres **parametresECC** amb la clau privada corresponent a la clau pública **clauVer**;
sortida: un booleà que indica si la firma és autèntica o falsa.