Criptografía: Aspectos Teóricos y Prácticos
Laboratorio 4: Curvas elípticas sobre $\mathbb{F}_p$

En este laboratorio y el próximo implementarán algunas funciones relacionadas con curvas elípticas. Lo entregarán el **27 de junio** (o antes, si prefieren) y lo harán en `equipos de dos`.

1. **Constructor:** Definirán una función que recibe una 5-tupla $[a_1, a_2, a_3, a_4, a_6]$ de números módulo p y verificará si hay una curva elíptica con esos coeficientes.
2. **Sumar dos puntos:** Definirán una función que recibe dos puntos P, Q y una curva elíptica (definida como una 5-tupla), y calculará $P + Q$ en la curva.
3. **Calcular nP:** Definirán una funcion que recibe un entero n, un punto P, una curva E y la cadena `'binaria'` o `'ternaria'`. La función calculará nP usando el algoritmo indicado por la cadena.
4. **Curvas usadas en la práctica:** En las especificaciones

   `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`

   se identifican varias curvas. En particular, verificarán los datos que definen la curva `P-192`, salvo el valor de r, y que calcularán, $5P$ para $P = (G_x, G_y)$.
5. **Generación de curvas pseudo-aleatorias:** Implementarán la función en el apéndice 4 de las especificaciones. Para calcular $\#E(GF(p))$ se puede hacer algo tipo:

   ```
   sage: E = EllipticCurve(GF(10007),[1,2,3,4,5])
   sage: E.cardinality()
   10076
   ```

   Esta función usa un algoritmo de Schoof-Elkies-Atkin para calcular $\#E(GF(p^k))$ en general. En las especificaciones, dice que el grupo tiene que ser de orden r para un r "adecuado". Esto quiere decir: sabemos que $\#E(\mathbb{F}_p) \approx p$ y el cociente $\frac{p}{r}$ tiene que ser 1, 2 o 4. Para identificar una curva aleatoria, ademas se tiene que explicitar(mirar Sección 2 de las especificaciones):

   - el primo
   - el orden del punto (G_x, G_y)
   - una semilla s de 160 bits para SHA-1
   - la salida c de SHA-1
   - el coeficiente b (porque estamos considerando curvas de la forma

     $$E : y^2 = x^3 - 3x + b)$$

   - un punto de base (G_x, G_y).

   El último itém es el más sutil: un algoritmo para hacer esto sería calcular un x módulo p al azar y verificar usando reciprocidad cuadrática (algo que ustedes deberían implementar) si es un residuo cuadrático módulo p). Si es un residuo módulo p pueden calcular la raíz cuadrada con:

   ```
   sage: F = GF(nth_prime(10000))
   sage: x = F(11111)
   sage: x.sqrt()
   44233
   ```

Sage tiene implementada una versión del algoritmo Tonelli-Shanks. Si quieren implementar Tonelli-Shanks pueden, pero recomiendo que usen la implementación que ya está en Sage (hay algunos también en puro python si quieren – por ejemplo

http://eli.thegreenplace.net/2009/03/07/computing-modular-square-roots-in-python/

– pero no prometo que estén bien).

6. **Verificación de que una curva sea pseudo-aleatoria:** Implementarán la función en el apéndice 5 de las especificaciones.

7. **ElGamal:** Implementarán la variante de ElGamal de Menezes-Vanstone que vimos en el teórico. Esto se puede hacer en "high-level", usando cosas ya implementadas en Sage:

```
sage: E = EllipticCurve(GF(101), [1,1,0,0,1])
sage: P = E.random_point()
sage: 100*P
(69 : 11 : 1)
sage: P.xy()
(53, 61)
sage: P.order()
105
sage: E.cardinality()
105
```

8. **Curvas elípticas definidas sobre cuerpos binarios:** Implementarán la función en apéndice 3 de las especificaciones, esto también se puede hacer en "high-level":

```
sage: E = EllipticCurve(GF(2^163,'a'), [1,1,0,0,1])
# Elementos de GF(2^163) son polinomios en la variable a
sage: P = E.random_point()
sage: (xP,yP) = P.xy()
sage: tauP = E((xP^2,yP^2))
# apliqué Frobenius al punto P y después lo definí como un punto de E
sage: parent(tauP)
Abelian group of points on Elliptic Curve defined by

y^2 + x*y = x^3 + x^2 + 1

over Finite Field in a of size 2^163
```