Trabajo Teoría

Curva Elíptica

3 de diciembre de 2019

Sofía Almeida Bruno
Pedro Manuel Flores Crespo
María Victoria Granados Pozo

Índice

1. Introducción

2. Curva Elíptica

Antes de entrar en cómo usar las curvas elípticas en criptografía debemos definirlas y estudiar su estructura de grupo.

Una curva elíptica será el conjunto de puntos que verifiquen la ecuación:

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

donde a y b son constantes y verifican $4a^3 + 27b \neq 0$[1]. Esta ecuación es la **ecuación de Weierstrass** para curvas elípticas. Normalmente, a, b, x, y toman valores en un cuerpo. Por ejemplo: los números reales $\mathbb{R}$, los números complejos $\mathbb{C}$, los números racionales $\mathbb{Q}$, un cuerpo finito $\mathbb{F}_p$ para un primo p (este es el que se utiliza en criptografía habitualmente), un cuerpo finito $\mathbb{F}_q$ donde $q = p^k$, $k \geq 0$, ...

De forma general, consideramos un cuerpo K y definimos la curva elíptica E sobre él de la siguiente forma:

$$E = \{(x, y) \in K \times K : y^2 = x^3 + ax + b\} \cup \{\infty\}.$$

El punto ∞ lo incluimos por definición. Lo podemos imaginar como un punto que se encuentra en la parte alta del eje y, pero también en la parte de abajo. Una línea que pase por este punto será una línea vertical, luego dos líneas verticales distintas coinciden en este punto.

Podemos usar Sage para visualizar algunos ejemplos de curvas elípticas. En el Código 1 observamos la forma genérica de definir en Sage curvas elípticas.

```
sage:  E = EllipticCurve(K, [a, b]); E
Elliptic Curve defined by y^2  = x^3 + x + 3 over a field K
```

Código 1: Curva elíptica en Sage

El Código 2 es el necesario para crear la curva con ecuación de WeiErstrass $y^2 = x^3 - x$.

```
sage: E =  EllipticCurve(RR, [-1, 0]);
      plot(E, (-4, 3), color=hue(0.6))
```

Código 2: Curva elíptica $y^2 = x^3 - x$

Podemos observar la salida obtenida en la Figura 1.

[1]Imponemos esta condición para que la curva no presente raíces múltiples, que pueden dar problemas en la teoría general que expondremos. Se pueden tratar ambos casos de forma individual aunque nosotros no incluiremos estas excepciones en nuestro trabajo

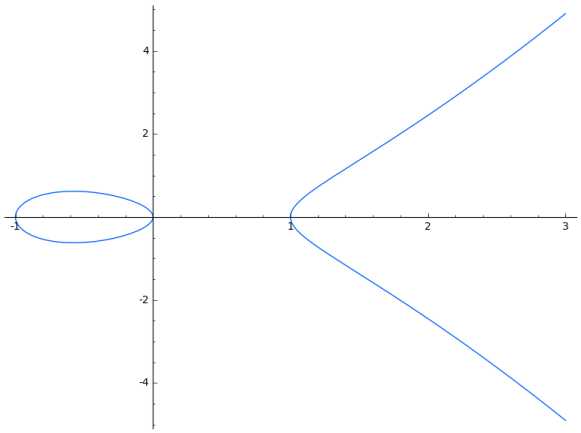

Figura 1: Salida Código 2

Programamos de forma similar el código para visualizar la gráfica de la curva dada por la ecuación $y^2 = x^3 + x$, véase el Código 3, cuya salida se encuentra en la Figura 2

```
sage: E =  EllipticCurve(RR, [1, 0]);
      plot(E, (-2, 4), color=hue(0.8))
```

Código 3: Curva elíptica $y^2 = x^3 + x$

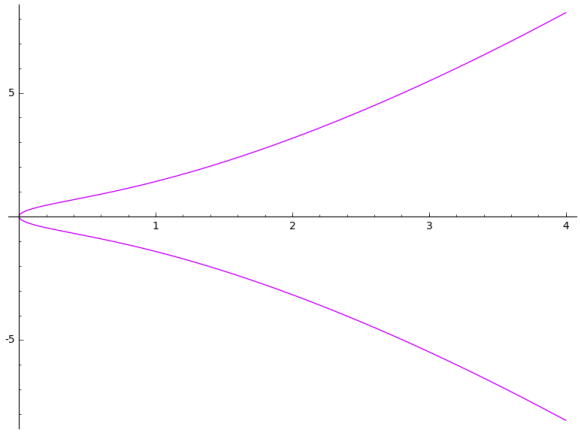

Figura 2: Salida Código 3

En la Figura 3 tenemos una tabla con la forma de las curvas elípticas para valores enteros de a entre -2 y 1 y b entre -1 y 2.

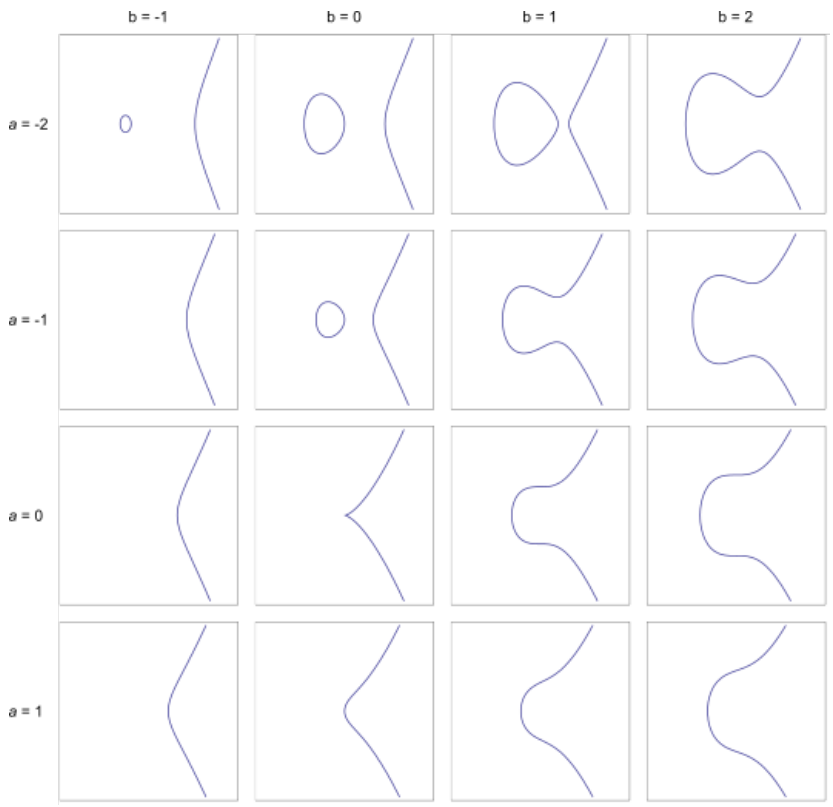

Figura 3: Ejemplos de curvas elípticas

2.1. Estructura de grupo

Dada una curva elíptica podemos definir un grupo abeliano formado por puntos de la misma.

1. Tomamos como elemento neutro el punto del infinito $\{\infty\}$.

2. El opuesto de un punto P viene dado por su simétrico por el eje X. Más concretamente, si $P = (x_P, y_P)$ entonces $-P = (x_P, -y_P)$.

3. En general, dados dos puntos de la curva $P = (x_P, y_P)$ y $Q = (x_Q, y_Q)$ con $x_P \neq y_P$, su suma es el punto $-R$ que viene definida por:

   $$(x_R, -y_R) = (m^2 - x_P - x_Q, -y_P - m(x_R - x_P)) = (m^2 - x_P - x_Q, -y_Q - m(x_R - x_Q)),$$

   donde

   $$m = \frac{y_P - y_Q}{x_P - x_Q}.$$

   Puede causar asombro que definamos $P + Q = -R$. Esto se deba al sentido geométrico de la suma en la curva. Esta operación debe seguir la siguiente regla: "dados tres puntos P, Q y R alineados y no nulos su suma $P + Q + R$ debe ser el elemento neutro". Por lo

tanto, $P+Q$ es el opuesto al punto de corte de la recta que une ambos puntos. De ahí también notamos la presencia del signo menos en la segunda coordenada tal y como indicamos en el punto anterior sobre los opuestos. También notar que el sentido geométrico nos aporta intuitivamente la conmutatividad ya que la recta tangente es la misma en el caso $P+Q$ y $Q+P$ y la asociatividad ya que $(P+Q)+R = P+(Q+R) = (P+R)+Q$ (y las demás posibilidades).

No podemos usar estas fórmulas en algunos casos especiales, por ejemplo cuando $P = Q$. En este caso no existe una sola recta que pase por "ambos" puntos. Sin embargo, si nos aproximamos a Q mediante puntos Q' con $Q' \neq P$ en seguida vemos que la recta que obtendríamos es la tangente en P. Así, definimos su suma como el opuesto del punto de intersección de la recta tangente en P. Otra situación interesante es cuando $P \neq Q$ pero no hay más puntos de corte. Nos encontramos en un caso parecido al anterior ya que uno de los puntos es tangente a la curva. Si suponemos que P es donde se da la tangencia, por el caso anterior tememos que $P + P = -Q$ por lo que $P + Q = -P$. En estas situaciones de tangencia tenemos que:

$$m = \frac{3x_P^2 + a}{2y_P}.$$

En ambos casos debemos comprobar que la suma pertenece a la curva y que los tres puntos están alineados.

4. Finalmente, dado $n \in \mathbb{Z}$, el producto por escalares lo definimos como:

$$nP = \begin{cases} (P+ \overset{n}{\cdots} +P) & \text{si } n > 0 \\ 0 & \text{si } n = 0 \\ (-P \overset{n}{\cdots} -P) & \text{si } n < 0 \end{cases}.$$

2.2. Subrupos

Notamos que si, $n, m \in \mathbb{Z}$ y $P \in E$ se cumple que:

$$nP + mP = (n+m)P.$$

Así, los múltiplos de P son cerrados bajo la suma. Obtenemos de este modo que los múltiplos de un punto de la curva forman un subgrupo cíclico. Nos preguntamos ahora cuál es el orden de dichos subgrupos, es decir, debemos encontrar el mínimo $n \in \mathbb{N}$ que cumpla $nP = \infty$. Para su cálculo debemos tener en cuenta el Teorema de Lagrange que nos dice que si H es un subgrupo de G entonces el orden de H es un divisor del orden de G. El orden del grupo de la curva elíptica lo podemos calcular en tiempo polinomial gracias al algoritmo de Schoof. De este modo, como tenemos el orden del grupo formado por la curva elíptica que notamos como N, el orden del subgrupo con generador P queda determinado por:

$$\text{mín}\{n \in \mathbb{N} : \ nP = \infty \text{ y } n|N\}.$$

Normalmente, el proceso seguido en los algoritmos de criptografía de la curva elíptica es el opuesto, es decir, primero elegir el orden del subgrupo que queremos y posteriormente encontrar el generador de la siguiente. Si el orden del subgrupo que queremos es un primo p el proceso a seguir es calcular el cofactor $h = N/n$, tomar un punto aleatorio $P \in E$ y obtener hP. Si es 0, tomar un nuevo punto y si no P es el generador deseado. Como anotación, los órdenes de los subgrupos se suelen escoger de orden primo ya que Pohlig y Hellman probaron que un criptosistema es seguro si, y solo si, no es factible el cálculo del logaritmo discreto en grupos con orden un primo (REFERENCIAR). En la siguiente sección exponemos el problema del logaritmo discreto en el contexto de las curvas elípticas.

Expuestas todas las herramientas necesarias, los parámetros de dominio para los algoritmos son (p, a, b, G, n, h) donde:

- p: primo
- a, b: Coeficientes de la curva elíptica, hay que elegirlos con cuidado para que el algoritmo sea seguro
- G: Generador del grupo
- n: Orden del grupo
- h: Cofactor del subgrupo

3. El problema del logaritmo discreto

La seguridad de la criptografía de la curva elíptica reside en la dificultad para calcular logaritmos discretos. Este problema es análogo el de otros criptosistemas como el Digital Signature Algorithm (DSA) o el intercambio de llaves mediante el mecanismo de Diffie-Hellman.

En nuestro caso, el problema consiste en lo siguiente: "si conocemos P y Q dos puntos de la curva elíptica, ¿podemos encontrar un k tal que $Q = kP$? ". Vemos que no estamos calculando un logaritmo como tal pero se sigue usando ese término por analogía a otros sistemas como se ha mencionado anteriormente.

Actualmente, no existe ninguna demostración matemática que efectivamente pruebe la dificultad de dicho cálculo. Solo sabemos que es difícil pero no podemos estar seguros. Con el siguiente ejemplo ilustramos su complejidad. Para ello usamos un algoritmo "Baby-step, giant-step" que se basa en el hecho de que cualquier entero x puede expresar como $x = am+b$ con a, m y b también enteros. Escribimos entonces:

$$\begin{aligned} Q &= xP \\ &= (am + b)P \\ &= amP + bP. \end{aligned}$$

Así, podemos expresar $Q - amP = bP$. El algoritmo consiste en en calcular algunos valores de bP y otros de $Q - amP$ hasta que encontremos una correspondencia. El algoritmo también

indica que debemos escoger $m = \sqrt{n}$ y los números a y b se mueven entre 0 y m por lo que mientras bP tiene incrementos pequeños ("baby") amP los tiene grandes ("huge"). Este método nos aporta una complejidad tanto en tiempo como en memoria de $O(\sqrt{n})$, es decir, exponencial pero es mejor que uno de fuerza bruta. De todos modos sigue siendo intratable ya que para un n tal que $\sqrt{n} = 7{,}922816251426434 \times 10^{28}$ el almacenamiento de datos que podemos llegar a necesitar es de $2,5 \times 10^{30}$ bytes de memoria (la capacidad de almacenamiento mundial es de aproximadamente 10^{21} bytes).

4. Algoritmos

4.1. ECDSA

ECDSA es un algoritmo de firma digital, Elliptic Curve Digital Signature Algorithm, es una variante del algoritmo DSA (Digital Signature Algortihm) aplicado a curvas elípticas. Trabaja con el hash del mensaje en lugar de con el propio mensaje. La elección de la función hash es importante, de esto dependerá la seguridad del sistema criptográfico. El hash del mensaje tendrá una longitud de n bit.

Imaginemos que Alice quiere firmar un mensaje con su llave privada (d_A), y la otra persona, Bob, quiere validar la firma con la llave pública de Alice (H_A). Alice es la única que puede producir las firmas válidas, sin embargo todo el mundo que tenga su llave pública puede verificarlas.

Alice y Bob están usando los parámetros del dominio. El hash truncado lo denotaremos por z.

Firma del mensaje

Algoritmo de firma del mensaje de Alice, a partir de k y z se genera la firma con la clave privada de Alice:

1. Tomamos un entero k de forma aleatorio en el conjunto $\{1, ..., n-1\}$.
2. Calcular $P = kG$.
3. Calcular el número $r = x_P \mod n$.
4. Si r es 0 entonces se toma otro k y se intenta de nuevo.
5. Se calcula $s = k^{-1}(z + rd_A) \mod n$ con k^{-1} el inverso multiplicativo de k módulo n.
6. Si s es 0 entonces se elige otro k y se vuelve al principio.

Al final obtendremos la firma que será la pareja (r, s)
Si el subgrupo tiene orden no primo, entonces el algoritmo ECDSA no se puede usar.

Verificación de la firma

Ahora entra en juego Bob que para validar la firma, a partir del mensaje firmado y de z con la clave pública de Alice H_A.

1. Calcular $u_1 = s^{-1}\ z \bmod n$
2. Calcular $u_2 = s^{-1}\ r \bmod n$
3. Calcular el punto $P = u_1 G + u_2 H_A$

La firma solo es válida si $r = x_P \bmod n$

Exactitud del algoritmo

Veamos que esta definición de P del punto 3 con la definición anterior de $P = kG$
Partimos de $P = u_1 G + u_2 H_A$, sustituimos el valor de la clave pública $H_A = d_A G$

$$\begin{aligned} P &= u_1 G + u_2 H_A \\ &= u_1 G + u_2 d_A G \\ &= (u_1 + u_2 d_A) G \end{aligned} \tag{1}$$

Sustituimos los valores de u_1 y u_2 que hemos declarado en la verificación de la firma:

$$\begin{aligned} P &= (u_1 + u_2 d_A) G \\ &= (s^{-1} z + s^{-1} r d_A) G \\ &= s^{-1}(z + r d_A) G \end{aligned} \tag{2}$$

Como el subgrupo cíclico de orden n en el que estamos trabajando está generado por G por tanto no hace falta poner "mod n" si múltiplicamos por G.

Ahora nos fijamos en la definición de $s = k^{-1}(z + rd_A) \bmod n$. Multiplicamos por k a ambos lados y dividimos por s y obtenemos: $k = s^{-1}(z + rd_A) \bmod n$

$$\begin{aligned} P &= s^{-1}(z + r d_A) G \\ &= kG \end{aligned} \tag{3}$$

Importancia de *k*

Es importante el valor que escojamos de k la clave secreta. Si además siempre firmamos con el mismo valor de k y con los mismos valores aleatorios, para los atacantes serían más fácil encontrar la clave privada.

Para descubrir la clave privada d_S necesitamos calcular k a partir de los hashes z_1 y z_2 y de las correspondientes firmas (r_1, s_1) y (r_2, s_2)

1. r_1 es igual a r_2, puesto que $r = x_P \bmod n$ y $P = kG$ es el mismo para las dos firmas.
2. Observamos que $(s_1 - s_2) \bmod n = k^{-1}\ (z_1 - z_2) \bmod n$ viene directamente de la definición de s
3. $k(s_1 - s_2) mod n = (z_1 - z_2) mod n$
4. $k = (z_1 - z_2)(s_1 - s_2)^{-1} \bmod n$

Ahora despejamos d_S de la definición de s y obtenemos que:

$$\begin{aligned} s &= k^{-1}(z + rd_S) mod n \\ d_S &= r^{-1}(sk - z) mod n \end{aligned} \tag{4}$$

Así hemos conseguido la clave privada con los valores que conocemos.

4.2. ECDH

4.3. Comparación con RSA

La principal diferencia entre la criptografía de la curva elíptica y la de RSA, es la cantidad de almacenamiento que se necesita para guardar una clave al mismo nivel de seguridad. Por ejemplo, si para guardar una clave de RSA se necesitan 1024 bits, para almacenar la de ECC se requiere de 160, las diferencias son aún mayores cuando aumenta el nivel de seguridad, para RSA son necesarios 15360 bits mientras que para ECC son 521 bits.

La relación entre los tamaños no es lineal, es decir, si multiplicamos por k el tamaño de RSA, no tenemos k veces el tamaño de ECC.

Por otro lado, la generación de la clave y la firma es considerablemente más rápido con ECC que con RSA.

Los algoritmos más rápidos para el computo de logaritmos discreots, además de la curva elíptica, son Pollard's rho y baby-step giant-step. Destaca uno en particula que es el algoritmo "criba general del cuerpo de números" el más rápido conocido a día de hoy para la factorización de enteros, que además se puede usar para el logaritmo discreto.

5. Conclusiones

I. Glosario