ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

1 DESAROLLO MATEMÁTICO

En este capítulo haremos el estudio matemático de la teoría de curvas elípticas. En el apartado 1.1 se desarrolla la teoría básica, los puntos de torsión son tratados en el apartado 1.2 y por último se particulariza la teoría de curvas elípticas sobre cuerpos finitos en el apartado 1.3.

Las referencias utilizadas para el desarrollo matemático han sido sido [5], [1] y [4].

1.1 TEORÍA BÁSICA DE CURVAS ELÍPTICAS

En este apartado se tratarán las ecuaciones de Weierstrass, las operaciones de adicción y duplicación, los puntos proyectivos, los endomorfismos de curvas elípticas y la estructura de los puntos de torsión.

Las principales referencias utilizadas en este capítulo han sido [5] y [1].

1.1.1 *Definición de curva elíptica*

En esta apartado veremos la definición general de curva elíptica aunque posteriormente simplificaremos la ecuación que la define.

Definición 1.1

Una *curva elíptica* E se define por una una ecuación de la forma

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

donde $a_1, a_2, a_3, a_4, a_6 \in K$ y $\Delta \neq 0$, siendo Δ el *discriminante* de E y definiéndose como:

$$\left.\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + 4a_2 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{aligned}\right\} \tag{2}$$

Si L es una extensión del cuerpo K, entonces el conjunto de puntos *L-racionales* de E es $E(L) = \{\infty\} \cup \{(x, y) \in L \times L : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\}$.

Nota 1.2 (comentarios de la definición 1.1)**.**

- La ecuación (1) se conoce como la *ecuación de Weierstrass*.
- Diremos que E *está definida sobre* K y lo notaremos E/K. A K lo llamaremos *cuerpo base*.
- La condición $\Delta \neq 0$ asegura que la curva elíptica no tenga puntos *singulares*, esto es, puntos que anulen las derivadas parciales de la función polínomica

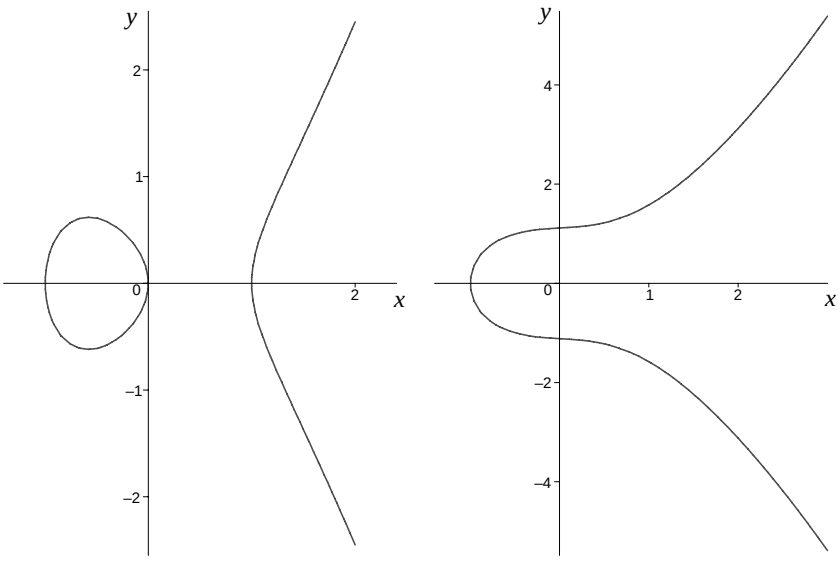
  $$f(x, y) = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6$$

  asociada a la curva elíptica. Esto asegura que no haya puntos en los que la curva tenga dos o más rectas tangentes.
- El punto ∞ lo llararemos *punto del infinito*. Es el único punto en la recta del infinito que satisface la forma proyectiva de la ecuación de Weierstrass (véase apartado 1.1.5).

Ejemplo 1.3 (curvas elípticas sobre $\mathbb{R}$)**.** Consideramos las curvas elípticas:

$$E_1 : y^2 = x^3 - x$$
$$E_2 : y^2 = x^3 + x$$

definidas sobre el cuerpo $\mathbb{R}$ de los números reales. Los puntos $E_1(\mathbb{R})$ y $E_2(\mathbb{R})$ se han representado en la Figura 1.



(a) $E_1 : y^2 = x^3 - x$ (b) $E_2 : y^2 = x^3 + x$

Figura 1: Curvas elípticas sobre $\mathbb{R}$

1.1.2 *Ecuaciones de Weierstrass simplificadas*

Nuestro objetivo es transformar la ecuación (1) por una ecuación más sencilla. En este apartado veremos varias transformaciones según la característica del cuerpo base.

Definición 1.4

Dos curvas elípticas E_1 y E_2 definidas sobre K y dadas por las ecuaciones de Weierstrass:

$$E_1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$E_2 : y^2 + a_1'xy + a_3'y = x^3 + a_2'x^2 + a_4'x + a_6'$$

se dicen que son *isomorfas sobre K* si existen $u, r, s, t \in K,\ u \neq 0$, tal que el cambio de variables lineal

$$(x, y) \mapsto (u^2x + r, u^3y + u^2sx + t) \tag{3}$$

transforma la ecuación E_1 en la ecuación E_2. La transformación (3) se llama un cambio de variables admisible.

El cambio de variables (3) es el único que deja «fijo» el punto del infinito y preserva la forma de la ecuación de Weierstrass. No vamos a entrar en más detalle, pero puede consultar [4, prop. III.3.1b] para más informácion.

Una ecuación de Weierstrass

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

puede simplificarse considerablemente aplicando cambios de variables admisibles. Veamos el caso en el que la característica del cuerpo base no es ni 2 ni 3.

Lema 1.5

Si $\mathrm{char}(K) \neq 2, 3$, entonces el cambio de variables admisible

$$(x, y) \mapsto \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{240}\right)$$

transforma E en la curva

$$y^2 = x^3 + ax + b \tag{4}$$

donde $a, b \in K$. El discriminante de esta curva es $\Delta = -16(4a^3 + 27b^2)$.

Demostración. Basta aplicar el cambio de variables y ver que efectivamente se obtiene la expresión (4). En su lugar, veamos el proceso por el cual se obtuvo dicha simplificación.

Dada la ecuación de Weierstrass (1), sumamos en ambos lados por $(a_1a_3x)/2 + a_3^2/4 + (a_1^2x^2)/4$ (podemos dividir por 2 ya que $\text{char}(K) \neq 2$) para completar el cuadrado:

$$\left(y + \frac{a_1x}{2} + \frac{a_3}{2}\right)^2 = x^3 + \left(a_2 + \frac{a_1^2}{4}\right)x^2 + \left(a_4 + \frac{a_1a_3}{2}\right)x + \left(a_6 + \frac{a_3^2}{4}\right)$$

Haciendo $y_1 = y + (a_1x)/2 + a_3/2$, obtenemos

$$y_1^2 = x^3 + a_2'x^2 + a_4'x + a_6'$$

para algunas constafntes $a_2', a_4', a_6' \in K$. Finalmente, sustituyendo $x_1 = x + a_2'/3$ (podemos dividir por 3 ya que $\text{char}(K) \neq 3$) resulta

$$y_1^2 = x_1^3 + ax_1 + b$$

para algunas constante $a, b \in K$. Para obtener el discriminante Δ basta sustiuir el valor de las constantes $a_4 = a,\ a_6 = b$ y $a_1 = a_3 = a_2 = 0$ en (2). □

Nota 1.6 (comentarios del lema 1.5)**.**

- Si el cuerpo base tiene característica 2, la ecuación anterior (4) no es válida ya que tiene puntos singulares.
- Para cuerpos base con característica 3, la ecuación anterior (4) si es válida, pero existan curvas que no tienen esta forma.

En la mayor parte del trabajo, desarollaremos la teoría de curvas elípticas utilizando la ecuación de Weierstrass simplificada (4). Sin embargo, como los cuerpos finitos de característica dos son de especial interés en computación, ocasionalmente señalaremos que modificaciones son necesarias para los cuerpos base de característica dos. Veamos la primera modificación.

Lema 1.7

Si la característica de K es 2, hay dos casos que considerar. Si $a_1 \neq 0$, entonces el cambio de variables admisible

$$(x, y) \mapsto \left(a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2a_4 + a_3^2}{a_1^3}\right)$$

transforma E en la curva

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

donde $a, b \in K$. Tales curvas se llaman *no supersingulares* (véase 1.34) y tienen discriminante $\Delta = b$. Si $a_1 = 0$, entonces el cambio de variables admisible

$$(x, y) \mapsto (x + a_2, y)$$

transforma E en la curva

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

donde $a, b, c \in K$. Tales curvas se llaman *supersingulares* (véase 1.34) y tienen discriminante $\Delta = c^4$.

Demostración. Basta sustituir el cambio de variables en la ecuación general de Weierstrass y operar. □

1.1.3 *Ley de grupo*

En este apartado veremos como dotar de estructura de grupo al conjunto de puntos una curva elíptica. Para ello definiremos una ley de composición o ley de grupo y le daremos sentido a la «suma» de puntos.

Sea E una curva elíptica definida sobre un cuerpo K. El siguiente método geométrico permite dados dos puntos en $E(K)$ producir un tercero en $E(K)$. Este método será la base para definir la ley de grupo.

Método de la cuerda y la tangente 1. Dados dos puntos P y Q , veamos como producir un tercer punto R. En primer lugar si P y Q son distintos, los pasos son:

1. Se dibuja una recta L de P a Q.
2. Esta recta intersecta la curva elíptica en un tercer punto.
3. Tomamos R como la reflexión de este punto sobre el eje-x.

Si los puntos P y Q son iguales, los pasos son:

1. Se dibuja la línea tangente L a la curva elíptica en P.
2. Esta línea intersecta la curva elíptica en un segundo punto.
3. Tomamos R como la reflexión de este punto sobre el eje-x.

Esto método se puede apreciar en la figura 2.

Nota 1.8 (comentarios del algoritmo 1.13)**.** El hecho de que $L \cap E$, contando multiplicidades, consiste en exactamente tres puntos (no necesariamente distintos) es un caso especial del teorema de Bézout [2, sec. I.7.8]. Sin embargo, como a continuación vamos a dar fórmulas explícitas, haremos la demostración utilizando dichas fórmulas y no será necesario utilizar un teorema tan general.

Inspirándonos en el método de la cuerda y la tangente, definimos la siguiente ley de composición para el grupo de puntos de una curva elíptica.

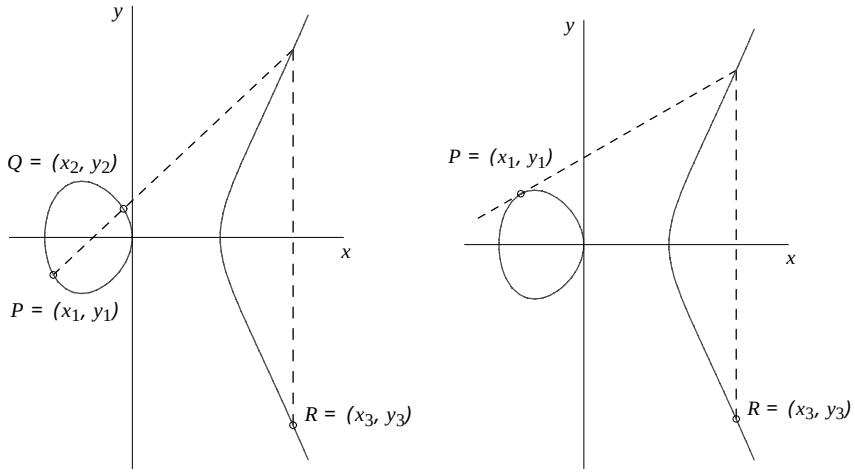


Figura 2: Método de la cuerda y la tangente

Definición 1.9 (ley de grupo)

Sea E una curva elíptica definida por la ecuación $y^2 = x^3 + ax + b$ sobre un cuerpo K de característica distinta de 2 y 3. Definimos la operación binaria

$$+ : E(K) \times E(K) \to E(K)$$

como sigue:

1. $P + \infty = \infty + P = P$, para todo $P \in E(K)$.
2. Si $P = (x, y) \in E(K)$, entonces $(x, y) + (x, -y) = \infty$. El punto $(x, -y)$ se denotará por $-P$ y se llamará el *opuesto* de P. Además, $-\infty = \infty$.
3. Sea $P = (x_1, y_1) \in E(K)$ y $Q = (x_2, y_2) \in E(K)$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde
$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$
$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$
4. Sea $P = (x_1, y_1) \in E(K)$, donde $P \neq -P$. Entonces $2P = (x_3, y_3)$ donde:
$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$
$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Demostración. Tenemos que comprobar que $+$ es una operación binaria válida, esto es, que a cada par de elementos de $E(K) \times E(K)$ le corresponde un único elemento de $E(K)$. Como la casuística anterior es total y exclusiva, basta ver que $+$ es una operación cerrada. Los casos a) y b) son triviales. Veamos los otros dos casos con detalle.

CASO c) Supongamos $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $P, Q \in E(K)$ con $P \neq \pm Q$. Consideramos la recta que los contiene:

$$L : y = m(x - x_1) + y_1, \text{ donde } m = \frac{y_2 - y_1}{x_2 - x_1}$$

Nótese que $x_2 \neq x_1$ ya que $P \neq \pm Q$. Para hallar la intersección de L con E sustituimos y:

$$(m(x - x_1) + y_1)^2 = x^3 + ax + b$$

Podemos reescribir esto de la forma

$$0 = x^3 - m^2x^2 + b'x + c' \tag{5}$$

para algunas constantes $b', c' \in K$. Así, las raíces de esta cúbica es justamente $L \cup E$.

Sabemos que las raíces de un polinomio están relacionadas con sus coeficientes. De hecho, para un polinomio cúbico mónico $x^3 + c_2x^2 + c_1x + c_0$ con raíces r, s, t se tiene:

$$\begin{aligned} x^3 + c_2x^2 + c_1x + c_0 &= (x - r)(x - s)(x - t) \\ &= x^3 - (r + s + t)x^2 + (rs + rt + st)x - rst \end{aligned}$$

En particular, $r + s + t = -c_2$. Como P y Q están en la intersección, x_1 y x_2 son dos raíces de (5), luego la tercera raíz α es $m^2 - x_1 - x_2$. Sustituyendo α en L resulta $\beta = m(x_3 - x_1) + y_1$, luego $(\alpha, \beta) \in E(K)$. Entonces $(\alpha, -\beta) = (x_3, y_3) \in E(K)$.

CASO d) Sea $P = (x_1, y_1)$, donde $P \neq -P$. Consideramos la recta tangente a E en P

$$L : y = m(x - x_1) + y_1, \text{ donde } m = \frac{3x_1^2 + a}{2y_1}$$

Nótese que $y_1 \neq 0$ ya que si no estaríamos en el caso b). Hallamos la intersección con E de forma análoga al caso c) y obtenemos la cúbica:

$$0 = x^3 - m^2x^2 + b'x + c'$$

para algunas constantes $b', c' \in K$. Análogamente al caso c), como x_1 es una raíz doble de la cúbica (derívese y evalúe en x_1) tenemos que la tercera raíz α es $m^2 - 2x_1$. Sustituyendo α en L resulta $\beta = m(x_3 - x_1) + y_1$, luego $(\alpha, \beta) \in E(K)$. Entonces $(\alpha, -\beta) = (x_3, y_3) \in E(K)$. □

Nota 1.10 (comentarios de la definición 1.9)**.** Para cuerpos base con característica 2 o 3, las fórmulas cambian. Por ejemplo, si E es una curva elíptica definida sobre un cuerpo K por la ecuación general de Weierstrass (1), el opuesto de un punto $P = (x, y) \in E(K)$ viene dado por

$$-P = (x, -a_1 x - a_3 - y)$$

En el apartado 1.3.1 veremos la ley de composición para una curva elíptica sobre un cuerpo finito de característica 2.

Con la operación binaria 1.9, el conjunto de puntos de una curva elíptiptica es un grupo abeliano.

Teorema 1.11

La suma 1.9 de puntos en una curva elíptica E sobre un cuerpo K de característica distinta de 2 y 3 satisface la siguientes propiedades:

- *Conmutatividad*:

$$P_1 + P_2 = P_2 + P_1, \ \forall P_1, P_2 \in E(K).$$

- *Existencia de elemento neutro*:

$$P + \infty = P, \ \forall P \in E(K).$$

- *Existencia de elemento opuesto*:

$$P + (-P) = \infty, \ \forall P \in E(K).$$

- *Asociatividad*:

$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3), \ \forall P_1, P_2, P_3 \in E(K).$$

En otras palabras, $(E(K), +, \infty)$ es un grupo abeliano.

Demostración. La conmutatividad es trivial en los casos a), b) y d). Para el caso c) también es fácil ya que la recta que une P_1 y P_2 es la misma que la recta que une P_2 y P_1. La existencia de elemento neutro e inverso también es directo de la definición 1.9.

La asociatividad puede probarse utilizando las fórmulas caso por caso, pero supone un esfuerzo demasiado laborioso. En su lugar, puede abordarse de forma más sofisticada bien estudiando las líneas y sus intersecciones con la curva elíptica en el plano proyectivo [5, sec. 2.4] o bien usando teoremas más generales como el de Riemann-Roch [4, teo. III.3.4.e]. □

Nota 1.12 (comentarios del teorema 1.11)**.** Puede encontrar una versión más general del teorema anterior para cuerpos base de cualquiera característica en [4].

1.1.4 *Multiplicación escalar*

En este apartado veremos un método eficiente para calcular la duplicación reiterada de un punto.

Sea P es un punto de una curva elíptica y k un entero positivo. Denotaremos kP a la suma $P + \ldots + P$ de k-sumandos. El siguiente algoritmo calcula kP más rápido que el método directo (sumar P consigo mismo repetidamente).

Algoritmo 1.13 (multiplicación por duplicación)**.** Sea k un entero positivo y sea P un punto de una curva elíptica. El siguiente algoritmo calcula kP.

1. Se empieza con $a = k$, $B = \infty$ y $C = P$.
2. Si a es par, se toma $a = a/2$, $B = B$ y $C = 2C$.
3. Si a es impar, se toma $a = a - 1$, $B = B + C$ y $C = C$.
4. Si $a \neq 0$, se va al paso 2.
5. Se devuelve B.

La salida B es kP.

Nota 1.14 (comentarios del algoritmo 1.13)**.**

- El único problema de este método es que el tamaño de las coordenadas incrementa muy rápidamente.
- Si trabajos sobre un cuerpo finito, podemos evitar este incoveniente reduciendo módulo p en cada operación.

1.1.5 *Puntos proyectivos*

En este apartado introduciremos los puntos proyectivos y veremos de donde procede el punto del infinito de una curva elíptica. Como en la mayor parte del trabajo no vamos a trabajar con puntos proyectivos, veremos este apartado de manera más informal.

Sea K un cuerpo. El *espacio proyectivo* dos dimensional sobre K, $\mathbb{P}^2(K)$, esta dado por clases de equivalencia de ternas (x, y, z) con $x, y, z \in K$ y al menos algún x, y, z no nulo. Dos ternas (x_1, y_1, z_1) y (x_2, y_2, z_2) se dicen que son *equivalentes* si existe un elemento no nulo $\lambda \in K$ tal que

$$(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_3)$$

y en tal caso escribiremos $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$. La clase de equivalencia de una terna solo depende de los ratios entre x, y, z. Por ello, la clase de equivalencia de (x, y, z) la denotaremos por $(x : y : z)$ y diremos que es un *punto proyectivo*.

Si $(x : y : z)$ es un punto proyectivo con $z \neq 0$, entonces $(x : y : z) = (x/z : y/z : 1)$ y de hecho $(x/z, y/z, 1)$ es el único representante de esta clase de equivalencia con $z = 1$. Tenemos así una correspondencia $1 - 1$ entre el conjunto de puntos proyectivos

$$\mathbb{P}^2(K)^* = \{(x : y : z) : x, y, z \in K,\ z \neq 0\}$$

y el *plano afín*

$$\mathbb{A}(K) = \{(x, y) : x, y \in K\}.$$

Si $z = 0$, el conjunto de puntos proyectivos de la forma $(x : y : 0)$ se llaman *recta del infinito* ya que sus puntos no se corresponden con ningúno del plano afín.

La *forma proyectiva* de una ecuación de Weierstrass de una curva elíptica E definida sobre K se obtiene remplazando x por x/z, y por y/z y quitando denominadores. Si alguna terna (x, y, z) no nula satisface la ecuación proyectiva entonces también las satisfacen las ternas $(x', y', z') \in (x : y : z)$. Podemos decir entonces que un punto proyectivo $(x : y : z)$ está en E. Tenemos así una correspondencia 1-1 entre los puntos del plano afín que están en E y los puntos proyectivos de $P^2(K)^*$ que están en E.

Si hacemos $z = 0$ en la forma proyectiva de la ecuación, obtenemos $0 = x^3$ y como alguna componente tiene que ser no nula, tenemos $y \neq 0$. Así, el único punto de la recta del infinito que está en E es el punto $(0 : y : 0) = (0 : 1 : 0)$. Este pusto se corresponde con el punto ∞ de la definición 1.1.

Hay situaciones en la que usar coordenadas proyectivas puede ser ventajoso (véase [5, sec 2.6]). Sin embargo, nosotros utilizaremos las coordenadas del plano afín y trataremos el punto del infinito como caso especial cuando sea necesario.

1.1.6 *Endomorfismos*

El principal objetivo de este apartado es probar la proposición 1.19 que será utilizada en la demostración del teorema de Hasse 1.33. Para ello, es necesario utilizar algunos resultados técnicos, los cuales solo los enunciaremos (puede encontrar su demostración en [5, sec. 2.9]).

Definición 1.15

Sea E una curva elíptica definida sobre un cuerpo K y sea $\overline{K}$ su clasura algebraica. Un *endomorfismo* de E es un homomorfismo $\alpha : E(\overline{K}) \to E(\overline{K})$ dado por funciones racionales (cocientes

de polinomios). Dicho de otro modo, α preserva la suma y el elemento neutro de $E(\overline{K})$.

El endomorfismo trivial que lleva cada punto a ∞ lo denotaremos por o.

Supondremos que α es no trivial a partir de ahora. El siguiente resultado técnico nos facilitará el manejo de endomorfismos de una curva elíptica.

Lema 1.16

Si α es un endomorfismo de una curva elíptica definida por la ecuación de Weierstrass simplificada (4), entonces α se puede escribir como

$$\alpha(x, y) = (r_1(x), r_2(x)y)$$

donde

- $r_1(x) = p(x)/q(x)$, con $p(x), q(x)$ polinomios sin factores comunes.
- Si $q(x) = 0$ para algún punto (x, y), entonces definimos $\alpha(x, y) = \infty$.
- Si $q(x) \neq 0$, entonces $r_2(x)$ está definida.

Definición 1.17

El *grado* de un endomorfismo α es

$$\deg(\alpha) = \text{máx}\{\deg p(x), \deg q(x)\}$$

si α es no trivial. Si $\alpha = 0$, definimos $\deg(0) = 0$.

Definición 1.18

Un endomorfismo α no trivial es *separable* si la derivada $r_1(x)'$ no es idénticamente cero.

El siguiente resultado será crucial en la demostración del teorema de Hasse 1.33. Denotaremos por $|C|$ al número de elementos de un conjunto C.

Proposición 1.19

Sea $\alpha \neq 0$ un endomorfismo separable de una curva elíptica E. Entonces

$$\deg(\alpha) = |\ker(\alpha)|$$

donde $\ker(\alpha)$ es el núcleo del homomorfismo $\alpha : E(\overline{K}) \to E(\overline{K})$. Si $\alpha \neq 0$ no es separable, entonces

$$\deg(\alpha) > |\ker(\alpha)|$$

Demostración. Por el lema 1.16, $\alpha(x, y) = (r_1(x), r_2(x)y)$ con $r_1(x) = p(x)/q(x)$. Supongamo que α es separable. Entonces $r_1' \neq 0$, por lo que $p'q - pq'$ no es el polinomio cero.

Sea S el conjunto de $x \in \overline{K}$ tal que $(pq' - p'q)(x)q(x) = 0$. Sea $(u, v) \in E(\overline{K})$ tal que

1. $u \neq 0,\ v \neq 0,\ (u, v) \neq \infty$,
2. $\deg(p(x) - uq(x)) = \max\{\deg(p), \deg(q)\} = \deg(\alpha)$,
3. $u \notin r_1(S)$ y
4. $(u, v) \in \alpha(E(\overline{K}))$.

Como $p'q - pq'$ no es el polinomio nulo, S es un conjunto finito, por lo que su imagen bajo α es finita. Por otro lado $\alpha(E(\overline{K}))$ es un conjunto infinito. Así, tal (u, v) existe.

Veamos que existen exactamente $\deg(\alpha)$ puntos $(x_1, y_1) \in E(\overline{K})$ tal que $\alpha(x_1, y_1) = (u, v)$. Para tales puntos, se tiene

$$\frac{p(x_1)}{q(x_1)} = u, \quad y_1 r_2(x_1) = v$$

Como $(u, v) \neq \infty$, $q(x_1) \neq 0$ luego $r_2(x_1)$ está definido. Como $v \neq 0$ y $y_1 r_2(x_1) = v$, se tendrá $y_1 = v/r_2(x_1)$, esto es, x_1 determina y_1, por lo que solo tenemos que contar valores de x_1.

Por la propiedad (2), $p(x) - uq(x)$ tiene $\deg(\alpha)$ raíces, contando multiplicidades. Tenemos que ver que $p - uq$ no tiene raíces múltiples. Supongamos que x_0 es una raíz múltiple. Entonces

$$p(x_0) - uq(x_0) = 0, \quad p'(x_0) - uq'(x_0) = 0$$

Multiplicando las ecuaciones $p = uq$ y $uq' = p'$ resulta

$$up(x_0)q'(x_0) = up'(x_0)q(x_0)$$

Como $u \neq 0$, esto implica que x_0 es una raíz de $pq' - p'q$, por lo que $x_0 \in S$. Así $u = r_1(x_0) \in r_1(S)$, contrario u la propiedad (3). Concluimos que $p - uq$ no tiene raíces múltiples y por ello tiene $\deg(\alpha)$ raíces distintas.

Como hay exactamente $\deg(\alpha)$ puntos (x_1, y_1) con $\alpha(x_1, y_1) = (u, v)$, el núcleo de α tiene $\deg(\alpha)$ elementos.

Nótese que como α es un homomorfismo, para cada $(u, v) \in \alpha(E(\overline{K}))$ hay exactamente $\deg(\alpha)$ puntos (x_1, y_1) con $\alpha(x_1, y_1) = (u, v)$. Las hipótesis sobre (u, v) se hicieron para obtener el resultado para al menos un punto, lo cual es suficiente.

Si α no es separable, entonces los pasos de la demostración siguen siendo válidos, excepto que $p' - uq'$ es siempre el polinomio cero en este caso, por lo que $p(x) - uq(x) = 0$ tiene siempre raíces múltiplces y por ello tiene menos de $\deg(\alpha)$ soluciones. □

Veamos un par de resultados que serán útiles para describir la estructura de los puntos de torsión del apartado 1.2.

Proposición 1.20

Sea E una curva elíptica definida sobre el cuerpo K. Sea $\alpha \neq 0$ un endomorfismo de E. Entonces α es sobreyectiva.

Demostración. Sea $(u, v) \in E(\overline{K})$. Como $\alpha(\infty) = \infty$, supongamos que $(u, v) \neq \infty$. Por el lema 1.16, α será de la forma $\alpha(x, y) = (r_1(x), r_2(x)y)$ con $r_1(x) = p(x)/q(x)$. Consideramos el polinomio $p(x) - uq(x)$. Distinguimos dos casos.

Supongamos que $p(x) - uq(x)$ no es un polinomio constante. Entonces tendrá una raíz x_0. Como p y q no tiene raíces en común, $q(x_0) \neq 0$. Sea $y_0 \in \overline{K}$ una raíz cuadrada de $x_0^3 + ux_0 + v$. Como $q(x_0) \neq 0$, $r_2(x)$ está definido y por lo tanto $\alpha(x_0, y_0)$ también y valdrá (u, v') para algún v'. Como $v'^2 = u^3 + au + b = v^2$, tenemos $v' = \pm v$. Si $v' = v$, hemos terminado. Si $v' = -v$, entonces $\alpha(x_0, -y_0) = (u, -v') = (u, v)$.

Supongamos que $p - uq$ es un polinomio constante. Como $E(\overline{K})$ no es finito y el núcleo de α sí es finito, solo un número finito de puntos de $E(\overline{K})$ puede tener como imagen un punto con una componente x dada. Así, bien $p(x)$ o $q(x)$ no es constante. Si p y q son dos polinomios no constantes, entonces hay como mucho una constante u tal que $p - uq$ es constante (si u' fuera otra constante que lo verificara, se tendría $(u' - u)q = (p - uq) - (p - u'q)$ es constante y $(u - u')p = u'(p - uq) - u(p - u'q)$ es constante, lo que implicaría que p y q son constantes). Así, hay al menos dos puntos, (u, v) y $(u, -v)$ para algún v, que no están en la imagen de α. Sea (u_1, v_1) otro punto. Entonces $\alpha(P_1) = (u_1, v_1)$ para algún P_1. Podemos elegir (u_1, v_1) tal que $(u_1, v_1) + (u, v) \neq (u, \pm v)$, por lo que existe P_2 con $\alpha(P_2) = (u_1, v_1) + (u, v)$. Entonces $\alpha(P_2 - P_1) = (u, v)$ y $\alpha(P_1 - P_2) = (u, -v)$.

□

Proposición 1.21

Sea E una curva elíptica definida sobre un cuerpo K y sea n un entero no cero. Consideramos el endomorfismo *multiplicación* por n dado por

$$n(P) = nP, \ \forall P \in E(\overline{K})$$

Supongamos que está dado por funciones racionales R_n y S_n, esto es,

$$n(x, y) = (R_n(x), yS_n(x))$$

para todo $(x, y) \in E(\overline{K})$. Entonces

$$\frac{R_n'(x)}{S_n(x)} = n.$$

Por tanto, la multiplicación por n es separable si y sólo si n no es un múltiplo de la característica del cuerpo base.

Para demostrar esta proposición, necesitamos un resultado técnico. La demostración de este lema se puede encontrar en [5, sec 2.9].

Lema 1.22

Sean α_1, α_2, α_3 endomorfismos no triviales de una curva elíptica E con $\alpha_1 + \alpha_2 = \alpha_3$. Supongamos que cada endomorfismo está dado de la siguiente forma

$$\alpha_j(x, y) = (R_{\alpha_j}(x), yS_{\alpha_j}(x))$$

y que existen constantes $c_{\alpha_1}, c_{\alpha_2}$ tal que

$$\frac{R_{\alpha_1}'(x)}{S_{\alpha_1}(x)} = c_{\alpha_1}, \quad \frac{R_{\alpha_2}'(x)}{S_{\alpha_2}(x)} = c_{\alpha_2}.$$

Entonces

$$\frac{R_{\alpha_3}'(x)}{S_{\alpha_3}(x)} = c_{\alpha_3} + c_{\alpha_3}$$

Demostración de la proposición 1.21. Dado un endomorfismo cualquiera α, se tiene

$$\alpha(x, -y) = \alpha(-(x, y)) = -\alpha(x, y).$$

En particular, $R_{-n} = R_n$ y $S_{-n} = -S_n$. Luego $R_n'/S_n = -R_n'/S_n$ y basta probar el resultado para n positivos.

Para $n = 1$, la primera parte de la proposición es cierta. Aplicando el lema anterior, si es cierta para n, también es cierta para $n + 1$ (la suma de n y 1). Así,

$$\frac{R_n'(x)}{S_n(x)} = n.$$

Por otro lado, $R'_n(x) \neq 0$ si y solo si $R'_n(x)/S_n(x) \neq 0$, que es equivalente a que la característica de K no divida a n. Por la definición de separabilidad, esto prueba la segunda parte. □

1.2 PUNTOS DE TORSIÓN

En este apartado introduciremos los puntos de torsión y su estructura que jugarán un papel importante en las curvas elípticas sobre cuerpos finitos. También veremos el emparejamiento Weil el cual utilizaremos en la demostración del teorema de Hasse 1.33.

Definición 1.23

Dada una curva elíptica E, un elemento del grupo $E(\overline{K})$ cuyo orden es finito se llamará *punto de torsión*.

Definición 1.24

Llamaremos *subgrupo de n-torsión* al subgrupo de puntos $\overline{K}$-racionales

$$E[n] = \{P \in E(\overline{K}) \mid nP = \infty\}.$$

Estos conjuntos son subgrupos ya que son los núcleos del endomorfismo multiplicación por n (definido en el apartado 1.1.6).

El siguiente resultado da la estructura de los subgrupos de torsión.

Teorema 1.25

Sea E una curva elíptica sobre un cuerpo K y sea n un entero positivo. Si la característica de K no divide a n, o es cero, entonces

$$E[n] \simeq \mathbb{Z}_n \oplus \mathbb{Z}_n.$$

donde $\oplus$ denota la suma directa de grupos. Por otro lado, si la característica de K es $p > 0$ y $p|n$, entonces

$$E[n] \simeq \mathbb{Z}_{n'} \oplus \mathbb{Z}_{n'} \text{ o } \simeq \mathbb{Z}_n \oplus \mathbb{Z}_{n'}$$

donde $n = p^r n'$ con $p \nmid n'$.

Para demostrar este teorema vamos a usar un resultado cuya demostración técnica, extensa y laboriosa omitiremos. Puede consultar su demostración en [5, sec. 3.2].

Proposición 1.26

Sea E una curva elíptica. El endomorfismo multiplicación por n de E tiene grado n^2.

Demostración del teorema 1.25. Supongamos primero que n no es múltiplo de la característica p del cuerpo. Por la proposicion 1.21, como n

no es múltiplo de la característica p, el endomorfismo multiplicación por n es separable. Por la proposición 1.19 y 1.26, el núcleo de este endomorfismo, $E[n]$, tiene orden n^2.

Por el teorema de estructura para grupos abelianos finitos, $E[n]$ es isomorfo a

$$\mathbb{Z}_{n_1} \simeq \mathbb{Z}_{n_2} \simeq \ldots \simeq \mathbb{Z}_{n_k}$$

para algunos enteros $n_1, n_2, ..., n_k$ con $n_i | n_{i+1}$ $\forall i$ y donde Z_{n_i} denota el grupo de enteros módulo n_i.

Sea l un primo que divide a n_1. Por lo que acabamos de ver, $E[l]$ tiene orden l^2. Tenemos así que $l \mid n_i$ $\forall i$ pero $E[l] \subset E[n]$, luego $k = 2$.

Multiplicar por n anula todos los elementos de $E[n] \simeq \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$, por lo que $n_2 | n$. Como $n^2 = |E[n]| = n_1 n_2$, se tiene $n_1 = n_2 = n$. Así,

$$E[n] \simeq \mathbb{Z}_n \oplus \mathbb{Z}_n.$$

Supongamos ahora que la característica p divide a n. Primero veamos la estructura de los subgrupos de torsión $E[p^k]$ con $k \geqslant 1$.

Por la proposición 1.21, el endomorfismo multiplicación por p no es separable. Por tanto, por la proposición 1.19, el núcleo $E[p]$ del endomorfismo multiplicación por p tiene orden estrictamente menor que el grado de este endomorfismo, que es p^2 por la proposición 1.26. Como todo elemento de $E[p]$ tiene orden 1 o p, el orden de $E[p]$ es una potencia de p, por lo que debe ser 1 o p. Si $E[p] = \infty$, entonces $E[p^k]$ debe ser trivial $\forall k$.

Supongamos ahora que $E[p]$ tiene orden p. Entonces $E[p^k]$ es cíclico (basta usar como antes el teorema de estructura de grupos abelianos y ver que $k = 1$). Así, $E[p^k]$ no puede tener como orden $p^{k'}$ con $k' > k$. Veamos que justamente el orden es p^k.

Supongamos existe un elemento P de orden p^j. Por la proposicion 1.20, el endomorfismo multiplicación por p es sobreyectivo, así que existirá un punto Q tal que $pQ = P$. Como

$$p^j Q = p^{j-1} P \neq \infty \quad \text{pero} \quad p^{j+1} Q = p^j P = \infty$$

Q tiene orden p^{j+1}. Por inducción, hay puntos de orden p^k $\forall k$. Por tanto, $E[p^k]$ es cíclico de orden p^k.

Escribiendo $n = p^r n'$ con $r \geqslant 0$ y $p \nmid n'$, resulta

$$E[n] \simeq E[n'] \oplus E[p^r].$$

donde $E[n'] \simeq \mathbb{Z}_{n'} \oplus \mathbb{Z}_{n'}$, ya que $p \nmid n'$. Acabamos de ver que $E[p^r] \simeq 0$ o $\mathbb{Z}_{p^r}$. El teorema chino del resto nos dice que

$$\mathbb{Z}_{n'} \oplus \mathbb{Z}_{p^r} \simeq \mathbb{Z}_{n'p^r} \simeq \mathbb{Z}_n$$

por lo que concluimos con

$$E[n] \simeq \mathbb{Z}_{n'} \oplus \mathbb{Z}_{n'} \quad \text{o} \quad \mathbb{Z}_{n'} \oplus \mathbb{Z}_n.$$

□

1.2.1 *Endomorfismos y matrices*

Los subgrupos de torsión nos permiten reducir cuestiones sobre los endomorfismos a cálculos con matrices. Para ello, dado un endomorfismo α, vamos a asociarle una matriz con entradas en $\mathbb{Z}_n$ que describirá la acción de α sobre una base de $E[n]$.

Definición 1.27

Sea n un entero positivo no divisible por la característica de K y $\alpha : E(\overline{K}) \to E(\overline{K})$ un homomorfismo (no necesariamente dado por funciones racionales). Definimos la matriz α_n dada por

$$\alpha_n = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

donde los $a, b, c, d \in \mathbb{Z}_n$ se obtienen de la siguiente forma:

- Se elige una *base* $\{\beta_1, \beta_2\}$ de $E[n] \simeq \mathbb{Z}_n \oplus \mathbb{Z}_n$.
- Se expresan las imagénes de β_1, β_2 por α en función de dicha base y se toman como a, b, c, d los coeficientes, esto es,
$$\alpha(\beta_1) = a\beta_1 + c\beta_2, \quad \alpha(\beta_2) = b\beta_1 + d\beta_2.$$

Los resultados que simplifican el cálculo del grado de un endomorfismo al cálculo del determinante de la matriz asociada son los siguientes.

Proposición 1.28

Sea α un endomorfismo de una curva elíptica E definido sobre un cuerpo K. Sea n un entero positivo no divisible por la característica de K. Entonces

$$\det(\alpha_n) \equiv \deg(\alpha) \pmod{n}.$$

Proposición 1.29

Sean α y β endomorfismos de E y sean a, b enteros. Consideramos el endomorfismo $a\alpha + b\beta$ definido por

$$(a\alpha + b\beta)(P) = a\alpha(P) + b\beta(P).$$

Entonces

$$\deg(a\alpha + b\beta) = a^2 \deg(\alpha) + b^2 \deg(\beta) + ab(\deg(\alpha + \beta) - \deg(\alpha) - \deg(\beta))$$

Para demostrar ambas proposiciones, necesitamos introducir el emparejamiento Weil.

1.2.2 *Emparejamiento Weil*

El emparejamiento de Weil sobre un subgrupo de torsión de una curva elíptica es una herramienta principal en el estudio de las curvas elípticas. En este trabajo se utilizará para probar la proposición 1.28 que será necesaria para probar el teorema de Hasse 1.33.

Sea E una curva elíptica sobre un cuerpo K y sea n un entero no divisible por la característica de K. Consideramos el grupo μ_n de las raíces n-ésimas de la unidad en $\overline{K}$, esto es,

$$\mu_n = \{x \in \overline{K} \mid x^n = 1\}.$$

Como la característica de K no divide a n, la ecuación $x^n = 1$ no tiene raíces múltiples y por lo tanto μ_n es un grupo cíclico de orden n.

Proposición 1.30

Sea E una curva elíptica sobre un cuerpo K y sea n un entero positivo no divisible por la característica de k. Entonces existe un emparejamiento

$$e_n : E[n] \times E[n] \to \mu_n$$

llamado *emparejamiento Weil* que satisface las siguientes propieades:

1. e_n es bilineaer en cada variable, esto es,

$$e_n(S_1 + S_2, T) = e_n(S_1, T)e_n(S_2, T)$$
$$e_n(S, T_1 + T_2) = e_n(S, T_1)e_n(S, T_2)$$

   $\forall S, S_1, S_2, T, T_1, T_2 \in E[n]$.

2. e_n es no degenerada en cada variable, esto es,

$$e_n(S, T),\ \forall T \in E[n] \implies S = \infty$$
$$e_n(S, T),\ \forall S \in E[n] \implies T = \infty$$

3. $e_n(T, T) = 1,\ \forall T \in E[n]$.

4. $e_n(T, S) = e_n(S, T)^{-1},\ \forall S, T \in E[n]$.

5. $e_n(\sigma S, \sigma T) = \sigma(e_n(S, T))$ para todo automorfismo (endomorfismo biyectivo) σ de $\overline{K}$ tal que σ sea la identidad en los coeficientes de E.

6. $e_n(\alpha S, \alpha T) = e_n(S,T)^{\deg(\alpha)}$ para todo endomorfismo α de E .

La proposición anterior requiere un estudio avanzado y detallado sobre curvas elípticas para realizar su demostración. Como no es nuestro objetivo hacer una teoría exhaustiva sobre curvas elípticas, el lector interesado puede consultar [5, cap. 11] para más información.

Corolario 1.31

Sea $\{T_1, T_2\}$ una base de $E[n]$. Entonces $e_n(T_1, T_2)$ es un raíz n-ésima de la unidad que genera μ_n (también llamada raíz *primitiva*).

Demostración. Vamos a usar la siguiente caracterización. Si ζ es una raíz n-ésima de la unidad, entonces ζ es una raíz primitiva si y solo si

$$\zeta^k = 1 \iff n \mid k.$$

Supongamos $e_n(T_1, T_2) = \zeta$ con $\zeta^d = 1$. Entonces $e_n(T_1, dT_2) = 1$. Por (1) y (3), $e_n(T_2, dT_2) = e_n(T_2, T_2)^d = 1$. Sea $S \in E[n]$. Entonces $S = aT_1 + bT_2$ para algunos enteros a, b. Se verifica

$$e_n(S, dT_2) = e_n(T_1, dT_2)^a e_n(T_2, dT_2)^b = 1.$$

Como esto es válido para cualquier S, (2) implica que $dT_2 = \infty$. Pero $dT_2 = \infty$ si y solo si $n \mid d$, luego ζ es una raíz primitiva. □

Ya podemos realizar las demostraciones pendientes de las proposiciones 1.28 y 1.29.

Demostración de la proposición 1.28. Por el corolario 1.31, $\zeta = e_n(T_1, T_2)$ es una raíz n-ésima primitivda de la unidad. Por la propiedades del endomorfismo Weil 1.30, se tiene

$$\begin{aligned}\zeta^{\deg(\alpha)} &= e_n(\alpha(T_1), \alpha(T_2)) = e_n(aT_1 + cT_2, bT_1 + dT_2)\\ &= e_n(T_1, T_1)^{ab} e_n(T_1, T_2)^{ad} e_n(T_2, T_1)^{cb} e_n(T_2, T_2)^{cd}\\ &= \zeta^{ad-bc}\end{aligned}$$

donde los coeficientes a, b, c, d son las entradas de la matriz α_n de la definición 1.27. Usando que ζ es una raíz primitiva, tenemos

$$\deg(\alpha) \equiv ad - bc \pmod{n}.$$

□

Demostración de la proposición 1.29. Sea n un entero no divisible por la característica de K. Representamos α y β por matrices α_n y β_n (con respecto a alguna base de $E[n]$). Entonces $a\alpha_n + b\beta_n$ representa la acción de $a\alpha + b\beta$ sobre $E[n]$. Un cálculo directo resulta

$$\begin{aligned}\det(a\alpha_n + b\beta_n) = {} & a^2 \det(\alpha_n) + b^2 \det(\beta_n) \\ & + ab(\det(\alpha_n + \beta_n) - \det(\alpha_n) - \det(\beta_n))\end{aligned}$$

para cualquier matriz α_n y β_n. Por tanto,

$$\begin{aligned}\deg(a\alpha + b\beta) \equiv {} & a^2 \deg(\alpha) + b^2 \deg(\beta) \\ & + ab(\deg(\alpha + \beta) - \deg(\alpha) - \deg(\beta)) \pmod{n}\end{aligned}$$

Como esto es válido para cualquier n, debe ser una igualdad. □

1.3 CURVAS ELÍPTICAS SOBRE CUERPOS FINITOS

Sea $\mathbb{F}_q$ un cuerpo finito de q elementos, donde q es la potencia n-ésima de un primo p y sea E una curva elíptica definida sobre $\mathbb{F}_q$. Como el número de pares (x, y) con $x, y \in \mathbb{F}_q$ es finito, el grupo de puntos $E(\mathbb{F}_q)$ es finito. En este aparatado estudiaremos las propiedades de este grupo, como el orden, que serán importantes en muchos contextos.

Los dos resultados más importantes se dan en los siguientes dos teoremas.

Teorema 1.32

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$. Entonces

$$E(\mathbb{F}_q) \simeq \mathbb{Z}_n \text{ or } \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$$

para algún entero $n \geqslant 1$ o para algunos enteros $n_1, n_2 \geqslant 1$ con $n_1 \mid n_2$.

Demostración. Como $E(\mathbb{F}_q)$ es un grupo abeliano finito, será isomorfo a una suma directa de grupos cíclicos

$$\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2} \oplus \dots \mathbb{Z}_{n_r}$$

con $n_i \mid n_{i+1}$. Como, para cada i, el grupo Z_{n_i} tiene n_1 elementos de orden un divisor de n_1, tenemos que $E(\mathbb{F}_q)$ tiene n_1^r elementos de orden un divisor de n_1. Por el teorema 1.25, hay hasta n_1^2 (incluso si permitimos coordenadas en la clasura algebraica de $\mathbb{F}_q$). Por tanto, $r \leqslant 2$. □

Teorema 1.33 (Teorema de Hasse)

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$. Entonces el orden de $E(\mathbb{F}_q)$ verifica

$$|q + 1 - |E(\mathbb{F}_q)|| \leqslant 2\sqrt{q}.$$

Demostración. Véase apartado 1.3.3. □

1.3.1 *Ley de grupo para cuerpos base de característica 2*

En el apartado 1.1.2, clasificamos las curvas elípticas sobre $\mathbb{F}_{2^m}$ según eran supersingulares o no. Ahora podemos definir este concepto.

Definición 1.34

Sea E una curva elíptica sobre un cuerpo base de característica p. Diremos que E es *supersingular* si $E[p] = \{\infty\}$.

Una ventaja de las curvas supersingulares es que los cálculos involucrados en la multiplicación de un punto por un escalar se pueden hacer con aritmética de cuerpos finitos, que es en general más rápida que la aritmética de curvas elípticas [5, cap. 4]. Sin embargo, estas curvas no son seguras en criptografía ya que son vulnerables a un tipo de ataques conocidos como ataques de emparejamiento Weil y Tate [1, cap. 4]. Por este último motivo, sólo vamos a dar las fórmulas de adicción para curvas no supersingulares sobre $\mathbb{F}_{2^m}$.

Fórmulas de adicción 1.35

Sea E una curva elíptica definida por la ecuación $y^2 + xy = x^3 + ax^2 + b$ sobre el cuerpo finito $\mathbb{F}_{2^m}$. Las fórmulas de adicción son las siguientes:

a) $P + \infty = \infty + P = P$, para todo $P \in E(\mathbb{F}_{2^m})$

b) Si $P = (x, y) \in E(\mathbb{F}_{2^m})$, entonces $(x, y) + (x, x + y) = \infty$ ya que $-P = (x, x + y)$. Además, $-\infty = \infty$.

c) Sea $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ y $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde

$$x_3 = \lambda^2\lambda + x_1 + x_2 + a, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

con $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

d) Sea $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, donde $P \neq -P$. Entonces $2P = (x_3, y_3)$ donde:

$$x_3 = \lambda^2\lambda + a = x_1^2 + b/x_1^2, \quad y_3 = x_1^2 + \lambda x_3 + x_3$$

con $\lambda = x_1 + y_1/x_1$.

1.3.2 *Endomorfismo de Frobenius*

En este apartado veremos un ejemplo importante de endomorfismo. Jugará una papel crucial en la teoría de curvas elípticas sobre cuerpos finitos.

Proposición 1.36

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$. La aplicación definida por

$$\phi_q(x,y) = (x^q, y^q), \quad \phi_q(\infty) = \infty$$

es un endomorfismo y se llama *endormofirsmo de Frobenius*.

Demostración. Veamos primero que $\phi_q(x,y) \in E(\overline{\mathbb{F}_q})$. Usaremos las siguientes propiedades de los cuerpos finitos:

$$(a+b)^q = a^q + b^q,\ \forall a, b \in \overline{\mathbb{F}_q}$$
$$a^q = a,\ \forall a \in \mathbb{F}_q.$$

Como la demostración es esencialmente la misma para la ecuación de Weirstrass general y la ecuación simplificada para cuerpos base de característica distinta de 2 y 3, usaremos está última. Tenemos

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

donde $a, b \in Fq$. Elevando todo a la potencia q-ésima obtenemos

$$(y^q)^2 = (x^q)^3 + a(x^q) + b.$$

Luego (x^q, y^q) está en E. Veamos ahora ϕ_q es un homorfismo. Sea $(x_1,y_1),(x_2,y_2) \in E(\overline{\mathbb{F}_q})$ con $x_1 \neq x_2$. La suma es (x_3,y_3) con

$$x_3 = m^2 - x_1 - x_2,\ y_3 = m(x_1 - x_3) - y_1,\ \text{donde } m = \frac{y_2 - y_1}{x_2 - x_1}$$

Elevando todo a la potencia q-ésima obtenemos

$$x_3^q = m'^2 - x_1^q - x_2^q,\ y_3^q = m'(x_1^q - x_3^q) - y_1^q,\ \text{donde } m' = \frac{y_2^q - y_1^q}{x_2^q - x_1^q}.$$

Por tanto,

$$\phi_q(x_3,y_3) = \phi_q(x_1,y_1) + \phi_q(x_2,y_2).$$

Los casos donde $x_1 = x_2$ o uno de los dos puntos es ∞ se comprueban fácilmente. Sin embargo, el caso de añadir un punto consigo mismo presenta una sutileza. Usando las fórmulas, tenemos que $2(x_1,y_1) = (x_3,y_3)$ donde

$$x_3 = m^2 - 2x_1,\ y_3 = m(x_1 - x_3) - y_1,\ \text{donde } m = \frac{3x_1^2 + a}{2y_1}.$$

Cuando elevamos a la potencia q-ésima, obtenemos

$$x_3^q = m'^2 - 2^q x_1^q,\ y_3^q = m'(x_1^q - x_3^q) - y_1^q,\ \text{donde } m' = \frac{3^q (x_1^q)^2 + a^q}{2^q y_1^q}.$$

Como $2, 3, a \in \mathbb{F}_q$, tenemos $2^q = 2,\ 3^q = 3,\ a^q = a$. Luego hemos obtenido la fórmula para duplicar el punto (x_1^q, y_1^q) de E.

Finalmente, como ϕ_q es un homomorfismo dado por funciones racionales, es un endomorfismo de E. □

Nótese que este endomorfismo no es más que aplicar el *automorfismo de Frobenius*

$$\begin{aligned}\overline{\mathbb{F}_q} &\to \overline{\mathbb{F}_q}\\ x &\mapsto x^q\end{aligned}$$

sobre las componentes de un punto. Veamos ahora una cuantas propiedades de este endomorfismo.

Lema 1.37

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$ y sea $(x,y) \in E(\overline{\mathbb{F}_q})$. Entonces

$$(x,y) \in E(\mathbb{F}_q) \iff \phi_q(x,y) = (x,y).$$

Demostración. Usando $a^q = a$, $\forall a \in \mathbb{F}_q$, tenemos

$$\begin{aligned}(x,y) \in E(\mathbb{F}_q) &\iff x,y \in \mathbb{F}_q\\ &\iff \phi_q(x) = x \text{ y } \phi_q(y) = y\\ &\iff \phi_q(x,y) = (x,y).\end{aligned}$$

□

Lema 1.38

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$. Entonces ϕ_q no es separable y es de grado q.

Demostración. Como $q = 0$ en $\mathbb{F}_q$, la derivada de x^q es idénticamente cero, luego ϕ_q no es separable. Por otro lado, ϕ_q tiene grado q aplicando la definición de grado de un endomorfismo. □

Proposición 1.39

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$, donde q es una potencia de un primo p. Sean r, s enteros no nulos. El endomorfismo $r\phi_q + s$ es separable si y solo si $p \nmid s$.

Demostración. Podemos escribir el endomorfismo multiplicación por r como

$$r(x,y) = (R_r(x), yS_r(x))$$

por el lema 1.16. Entonces

$$\begin{aligned}(R_{r\phi_q}(x), yS_{r\phi_q}(x)) = (\phi_q r)(x,y) &= (R_r^q(x), y^q S_r^q(x))\\ &= (R_r^q(x), y(x^3 + ax + b)^{(q-1)/2} S_r^q(x)).\end{aligned}$$

Así,

$$c_{r\phi_q} = R'_{r\phi_q}/S_{r\phi_q} = qR_r^{q-1}R'_r/S_{r\phi_q} = 0.$$

Además, $c_s = R'_s/S_s = s$ por la proposición 1.21. Por el lema 1.22,

$$R'_{r\phi_q+s}/S_{r\phi_q+s} = c_{r\phi_q+s} = c_{r\phi_q} + c_s = 0 + s = s$$

Por lo tanto, $r\phi_q + s$ es separable si y solo si $R'_{r\phi_q+s} \neq 0$ y esto ocurre si y solo si $p \nmid s$. □

Proposición 1.40

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$ y consideremos el endomorfismo $\phi_q^n - 1$ con $n \geqslant 1$. Entonces

1. $\ker(\phi_q^n - 1) = E(\mathbb{F}_{q^n})$.
2. $\phi_q^n - 1$ es separable, por lo que $|E(\mathbb{F}_{q^n})| = \deg(\phi_q^n - 1)$.

Demostración. Como ϕ_q^n es el endomorfismo de Frobenius para el cuerpo $\mathbb{F}_{q^n}$, (1) es consecuencia del lema 1.37. (2) se tiene aplicando las proposiciones 1.39 y 1.19. □

Lema 1.41

Sean r, s enteros. Entonces $\deg(r\phi_q - s) = r^2q + s^2 - rsa$.

Demostración. La proposición 1.29 implica que

$$\begin{aligned}\deg(r\phi_q - s) = r^2\deg(\phi_q) + s^2\deg(-1) \\ + rs(\deg(\phi_q - 1) - \deg(\phi_q) - \deg(-1)).\end{aligned}$$

Como $\deg(\phi_q) = q$ y $\deg(-1) = 1$, el resultado se deduce de 1.40 y 1.32. □

1.3.3 *Teorema de Hasse*

Tenemos ya todas las herramientas para probar el teorema de Hasse.

Teorema (1.33)

Sea E una curva elíptica definida sobre un cuerpo finito $\mathbb{F}_q$. Entonces el orden de $E(\mathbb{F}_q)$ verifica

$$|q + 1 - |E(\mathbb{F}_q)|| \leqslant 2\sqrt{q}.$$

Demostración del teorema 1.33. Sea

$$a = q + 1 - |E(\mathbb{F}_q)| = q + 1 - \deg(\phi_q - 1)$$

donde estamos usando proposición 1.40. Veamos que $|a| \leqslant 2\sqrt{q}$.

Como $\deg(r\phi_q - s) \geqslant 0$, el lema 1.41 implica que

$$q\left(\frac{r}{s}\right)^2 - a\left(\frac{r}{s}\right)^2 + 1 \geqslant 0.$$

para cualesquiera enteros r, s. Como el conjunto de los racionales es *denso* en $\mathbb{R}$, tenemos que

$$qx^2 - ax + 1 \geqslant 0$$

para todo número real x. Así, el discriminante de este polinomio es negativo o cero, lo que implica que $a^2 - 4q \leqslant 0$, luego $|a| \leqslant 2\sqrt{q}$. □

Nota 1.42 (comentarios del teorema 1.33)**.**

- Como la ecuación de Weierstrass tiene al menos dos soluciones para todo $x \in \mathbb{F}_q$, una primera acotación del orden de $E(\mathbb{F}_q)$ es $|E(\mathbb{F}_q)| \in [1, 2q + 1]$. El teorema de Hasse proporciona cotas más optimas:

  $$|E(\mathbb{F}_q)| \in [q + 1 - 2\sqrt{q},\ q + 1 + 2\sqrt{q}].$$

  Como $2\sqrt{q}$ es pequeño respecto a q, $E(\mathbb{F}_q) \approx q$.

- Tres resultados fueron clave para la demostración:
  - La identificación de $E(\mathbb{F}_q)$ con el núcleo de $\phi_q - 1$.
  - La igualdad entre el orden del núcleo y el grado de $\phi_q - 1$ gracias a la separabilidad de $\phi_q - 1$
  - El emparejamiento Weil, especialmente la parte (6) del teorema 1.30 y su consecuencia 1.40.

2

DESARROLLO INFORMÁTICO

En este capítulo haremos el desarrollo informático sobre criptografía con curvas elípticas. En el apartado 2.1 hablaremos sobre criptografía asimétrica utilizando curvas elípticas, explicaremos algunos protocolos criptografícos con curvas elípticas en el apartado 2.2 y por último en el apartado 2.3 hablaremos del programa desarrollado para trabajar con curvas elípticas y protocolos criptográficos.

Las referencias utilizadas han sido sido [1], [3], [5] y [4].

2.1 CRIPTOGRAFÍA CON CURVAS ELÍPTICAS

La criptografía de llave pública (o asimétrica) fue inventada por Diffie y Hellman en 1976, aunque no fueron capaces de encontrar un método práctico para implementar su idea. El primer criptosistema de llave pública llevado a la práctica fue concebido por Rivest, Shamir y Adleman. El criptosistema RSA basa su seguridad en la dificultad de factorizar números grandes. Sin embargo, Diffie y Hellman describieron un algoritmo de intercambio de llaves cuya seguridad se basaba en el logaritmo discreto en $\mathbb{F}_q^*$ y posteriormente ElGamal creó un criptosistema de llave pública basado en el mismo problema. Koblitz y Miller propusieron remplazar el cuerpo finito $\mathbb{F}_q$ con una curva elíptica E, con la esperanza de que el logaritmo discreto en el grupo de puntos de una curva elíptca fuera más difícil de resolver que el logaritmo discreto en el grupo multiplicativo de un cuerpo finito. Su intuición conllevó la creación de la criptografía con curvas elípticas.

En esta sección trataremos el problema del logaritmo discreto y sus ataques y dos aspectos comunes de los protocolos criptográficos que vamos a ver: los parámetros de dominio y las parejas de llaves.

2.1.1 *Problema del logaritmo discreto*

La dificultad del problema del logaritmo discreto es esencial para la seguridad de la criptografía asimétrica sobre curvas elípticas.

Definición 2.1

El *problema del logaritmo discreto sobre curvas elípticas (ECDLP)* es: dado una curva elíptica E definida sobre un cuerpo finito $\mathbb{F}_q$, un punto $P \in E(\mathbb{F}_q)$ de orden n y un punto $Q \in \langle P \rangle$, encontrar el entero $k \in [0, n-1]$ tal que $Q = kP$. El entero k se

llama el *logaritmo discreto de* Q *respecto a la base* P y se denota $k = \log_P Q$.

El ataque más simple para resolver el ECDLP es el ataque por *fuerza bruta*, esto es, probar todos los valores posibles de k hasta dar con el válido. El tiempo de ejecución es aproximadamente n pasos en el peor caso y $n/2$ pasos en el caso medio. Así, el ataque por fuerza bruta puede ser evitado tomando puntos base cuyo orden n sea suficientemente largo (p. ej. $n > 2^{80}$).

El mejor ataque conocido para curvas arbitrarias es la combinación del *algoritmo de Pohlig-Hellman* y del *algoritmo rho de Pollard*, que tiene un complejidad temporal de $O(\sqrt{p})$, esto es, el tiempo de ejecución es exponencial en $\log p$. donde p es el divisor primo más grande de n. Para resistir este ataque, se deben elegir puntos base cuyo orden n sea divisible por un primo p suficiente grande (p. ej $p > 2^{160}$).

Además se deben evitar ciertos tipos de curvas para los cuales existen ataques específicos más rápidos que el algoritmo rho de Pollard, llamados *ataques por isomorfismo*. Así pues, se deben evitar curvas *anómalas* (curvas cuyo orden es de la forma $|E(\mathbb{F}_p)| = p$), curvas supersingulares (véase 1.34), curvas con órdenes de la forma $|E(\mathbb{F}_q)| = q - 1$ y curvas sobre $\mathbb{F}_{2^m}$ si m es compuesto. Adicionalmente, se debe comprobar que n no divide a $q^k - 1$ para todo $1 \leqslant k \leqslant C$, donde C es suficientemente grande (si $n > 2^{160}$, entonces $C = 20$ basta).

Si se siguen estas restricciones, se cree que el ECDLP no es factible con la tecnología actual. Nótese que no existe prueba matemática de la intratabilidad del ECDLP, esto es, nadie ha probado que no existe un algoritmo eficiente que lo resuelve. Para apreciar la dificultad de encontrar dicha prueba, la no existencia de un algoritmo en tiempo polinomial para el ECDLP implicaría que $P \neq NP$, una de las preguntas abiertas mas importantes de la teoría de la computación.

Puede encontrar más información sobre el ECDLP y sus ataques en [1, cap. 4] y en [5, cap. 4].

2.1.2 *Parámetros de dominio*

Los parámetros de dominio de un esquema criptográfico con curvas elípticas describen una curva elíptica E definida sobre un cuerpo finito $\mathbb{F}_q$, un punto base $P \in E(\mathbb{F}_q)$ y su orden n. Los paramétros deben ser elegidos para que el ECDLP sea resistente a los ataques conocidos.

Definición 2.2

Los *paramétros de dominio* $D = (q, a, b, P, n)$ están constituidos por:

- El *orden* q *del cuerpo finito*.
- Dos *coeficientes* $a, b \in \mathbb{F}_q$ que definen la ecuación de la curva eliptica E sobre $\mathbb{F}_q$ (p. ej. $y^2 = x^3 + ax + b$ si la característica del cuerpo finito es distinta de 2 y 3 o $y^2 + xy = x^3 + ax^2 + b$ si la característica es 2).
- Dos elementos $x_p, y_p \in \mathbb{F}_q$ que definen el punto $P = (x_p, y_p) \in E(\mathbb{F}_q)$. P se conoce como el *punto base*.
- El *orden* n de P, normalmente un número primo.

A la hora de elegir los parámetros de dominio, existen dos opciones. Por un lado, se pueden generar aleatoriamente y posteriormente validarlos para comprobar que describen una curva elíptica segura. Nótese que para cumplir las restricciones de seguridad es necesario determinar el número de puntos de una curva elíptica. Entre los distintos algoritmos para ello (fuerza bruta, el método de multiplicación compleja, ...) el mejor es el algoritmo de Schoof-Elkies-Atkin (SEA). En [1, cap. 4] puede encontrar los algoritmos de generación y validación de parámetros de dominio, mientras que en [4, cap. XI] puede encontrar un descripción del algoritmo SEA.

Por otro lado, los parámetros de dominio se pueden elegir de algún *estándar*. Organizaciones como el Instituto Nacional de Normas y Tecnología (NIST) o el Grupo de Estándares para la Criptografía Eficiente (SECG), entre muchos otros, publican parámetros de dominio verificados. En [1, apéndice B] puede encontrar algunas de estas organizaciones y los estándares que han publicado.

2.1.3 *Pareja de llaves*

En los protocolos que vamos a ver, cada participante dispondrá de una pareja de llaves: una llave privada (conocida solo por dicho participante) y una llave pública (publicada al resto de participantes). Esta pareja de llaves estará asociada a unos parámetros de dominio particulares. Para generar la pareja de llaves, se elige un punto aleatorio Q en el grupo $\langle P \rangle$. La correspondiente llave privada es $d = \log_P Q$.

Algoritmo 2.3. Sea $D = (q, a, b, P, n)$ los parámetros de dominios. Los pasos para generar una pareja de llaves son:

1. Seleccionar $d \in [1, n-1]$ aleatoriamente.
2. Calcular $Q = dP$.
3. Devolver (Q, d) donde Q es la llave pública y d la llave privada.

Nótese que el problema de calcular la llave privada a partir de la llave pública es precisamente el ECDLP. Por eso es crucial que los parámetros de dominio sean seleccionados de tal forma que el ECDLP sea intratable. Además, es necesario que los participantes validen las llave públicas en todo procedimiento ya que existen ataques efectivos si no se hace algunas comprobaciones [1, cap. 4].

2.2 PROTOCOLOS CRIPTOGRÁFICOS

En este sección vamos ver dos ejemplos de protocolos criptográficos que usan curvas elípticas. El primero será un protocolo de intercambio de llaves y el segundo será un esquema de firma digital. Describiremos ambos protocolos a nivel de introducción. Para detalles técnicos véase [1, cap. 4].

Como es común en criptografía, vamos a representar a los participantes que desean comunicarse como Alicia y Bob y representaremos al atacante por Eva.

2.2.1 *Protocolo Diffie-Hellman*

El primer protocolo que vamos a ver es el *Protocolo de Intercambio de Llaves Diffie-Hellman para Curvas Elípticas* o ECDH.

Protocolo criptográfico 2.4 (ECDH)**.** El siguiente procedimiento permite a Alicia y Bob intercambiar de forma segura el valor de un punto en una curva elíptica aunque ninguno de los dos conozca inicialmente el valor del punto.

1. Alicia y Bob concuerdan unos parámetros de dominio $D = (q, a, b, P, n)$.
2. Alicia calcula su pareja de llaves (Q_A, d_A) según 2.3.
3. Bob calcula su pareja de llaves (Q_B, d_B) según 2.3.
4. Alicia y Bob intercambian sus llaves pública Q_A, Q_B.
5. Alicia calcula $d_A Q_B$ y Bob calcula $d_B Q_A$. Ambos cálculos devuelven el punto $(d_A d_B)P$.

La utilidad de este protocolo es que permite que Alicia y Bob intercambien una llave por un canal inseguro que pueda ser utilizada posteriormente para transmitir datos sobre un esquema de cifrado simétrico como el Advanced Encryption Standard o AES. Esta llave se suele obtener extrayendo unos cuantos bits de la coordenada x de $(d_A d_B)P$ o evaluando una función hash en dicha coordenada x.

El atacante, Eva, conoce los parámetros de dominio y las llaves públicas Q_A y Q_B, por lo que si Eva fuera capaz de resolver el ECDLP

podría encontrar las llaves privadas d_A y d_B y recuperar el punto secreto $(d_A d_B)P$. Sin embargo, en principio Eva no necesita encontrar d_A o d_B. Le basta con resolver el problema de Diffie-Hellman: dados tres puntos $P, d_A P, d_B P$ en $E(\mathbb{F}_q)$, calcular el punto $(d_A d_B)P$. Actualmente, la única forma de resolver el problema de Diffie-Hellman es resolver el problema del logaritmo discreto asociado pero no se sabe si ambos problemas son equivalentes [1].

2.2.2 *Protocolo ECDSA*

Un *esquema de firma digital* permite a Alicia usar su llave privada para firmar un documento (p. ej. un fichero del ordenador) de tal forma que Bob puede usar la llave pública de Alicia para verificar la validez de la firma. Un ejemplo de esto es el *Algoritmo de Firma Digital para Curvas Elípticas* o ECDSA.

Protocolo criptográfico 2.5 (ECDSA)**.** El siguiente procedimiento permite a Alicia firmar un documento digital y a Bob verificar que la firma es válida.

1. Alicia y Bob concuerdan unos parámetros de dominio $D = (q, a, b, P, n)$, con n primo.
2. Alicia calcula su pareja de llaves (Q, d) según 2.3.
3. Alicia representa el documento con un entero m (por ejemplo con una función hash).
4. Alicia calcula la firma (r, s) de m de la siguiente forma:
   - Selecciona $k \in [1, n-1]$ aleatoriamente.
   - Calcula el punto $kP = (x, y)$.
   - Calcula $\overline{x}$ como el representante entero del elemento $x \in \mathbb{F}q$.
   - Calcula $s \equiv k^{-1}(m + d\overline{x}) \pmod{n}$.
5. Alicia publica (m, r, s, Q).
6. Bob verifica la firma de la siguiente forma:
   - Calcula $u_1 \equiv ms^{-1} \pmod{n}$.
   - Calcula $u_2 \equiv rs^{-1} \pmod{n}$.
   - Calcula el punto $X = u_1 P + u_2 Q$.
   - Calcula v como el representante entero de la coordenada x de X.
   - Comprueba si $v \equiv r \pmod{n}$. En caso afirmativo, la firma es válida. En caso negativo, la firma no es válida.

Demostración de que la verificación de la firma funciona. Si una firma (r, s) de un mensaje m fue generada por el firmador legítimo, entonces $s \equiv k^{-1}(m + dr) \pmod{n}$. Reordenando se tiene

$$k \equiv s^{-1}(m + dr) \equiv s^{-1}m + s^{-1}rd \equiv u_1 + u_2 d \pmod{n},$$

Así $X = u_1 P + u_2 Q = (u_1 + u_2 P) = kP$, por lo que $v = r$ como se requería. □

Análogamente al protocolo Diffie-Hellman, si Eva, el atacante, pudiera calcular logaritmos discretos, entonces podría calcular la llave privada d y firmar como Alicia en cada mensaje. Por otro lado, si el número aleatorio k se reutiliza, Eva podría recuperar d sin necesidad de resolver el ECDLP. Por ello, k debe ser generando de forma segura, almacenado de forma segura y destruido de forma segura tras haber sido utilizado.

En el paso (3) es recomendable utilizar una función hash para firmar documentos grandes y prevenir varios tipos de ataques. Ejemplo de funciones hash comunes son MD5 (Message-Digest Algorithm 5) y la familia de algoritmos SHA (Secure Hash Algorithm). Puede encontrar más información sobre este protocolo en [1].

2.3 CRIPTOGRAFÍA CON CURVAS ELÍPTICAS CON PYTHON

En esta sección se explica el programa desarollado `Criptografía con Curvas Elípticas con Python` o `ccepy`. Este programa permite trabajar con el grupo de puntos de una curva elíptica y desarrollar protocolos criptográficos que usen curvas elípticas.

Puede consultar el código fuente en **??**. La documentación en formato de página web se adjutará a este documento.

2.3.1 *Motivación*

Existen diversos software de álgebra computacional que permiten hacer cálculos sobre curvas elípticas. [5] muestra como usar algunos de los principales ejemplos en este ámbito como `Pari`, `Magma` y `Sage`.

El principal motivo para implementar un programa y no utilizar uno ya existente ha sido para aplicar los conceptos aprendidos en el desarrollo matemático e implementar los algoritmos estudiados en los apartados 2.1 y 2.2 del desarrollo informático. Sin embargo, y no menos importante, otro motivo era crear un programa libre y gratuito, fácil de entender y extender y con una documentación extensa en español.

A diferencia de grandes soluciones como `Magma` o `Sage`, `ccepy` es framework *minimalista*, es decir, provee un conjunto de funcionalidades imprescindibles para cualquiera que desee trabajar con curvas elípticas y a partir de dichas funcionalidades es muy fácil añadir nuevas mejoras. Además, dichas funcionalidades se han implementado siguiendo un diseño lo más simple posible, de tal forma que entender el programa en su totalidad requiere un esfuerzo mínimo comparado con otros software.

2.3.2 *Herramientras utilizadas*

Se ha utilizado `python3` como lenguaje de programación para `ccepy`. La lógica del programa no usa ninguna biblioteca externa, solo utiliza la biblioteca estándar de `python`. Por eso, con tener `python3` instalado es suficiente para usar `ccepy`. En la documentación adjunta se detalla como usar este programa.

Sin embargo, para generar la documentación o ejecutar las pruebas, si es necesario instalar bibliotecas externas: `sphinx` y `hypothesis`. `sphinx` se ha utilizado para realizar la documentación mientras que `hypothesis` se ha utilizado para realizar pruebas basados en propiedades. Estos procesos están detallados en los apartados **??** y 2.3.5 respectivamente.

Como control de versiones se ha utilizado `git`.

Por último, aunque no es una herramienta, destacar que se ha utilizado la «Guía de Estilo de Google para Python» para mantener una convención es la escritura del código.

2.3.3 *Arquitectura*

El software `ccepy` consta de cuatro módulos principales:

```
Aritmética elemental
Cuerpos finitos
Curvas elípticas
Esquemas criptográficos
```

y un módulo secundario `Listado de curvas elípticas`.

Cada módulo principal usa los módulos anteriores, en el sentido que el módulo de cuerpos finitos usa el de aritmética elemental, el módulo de curvas elípticas usa el módulo de cuerpos finitos (y en consecuencia el de aritmética elemental) y el módulo de esquemas criptográficos usa el módulo de curvas elípticas (y en consecuencia el de cuerpos finitos y aritmética elemental).

2.3.4 *Implementación*

Para implementar `ccepy` hemos seguido un diseño orientado a objetos ya que los elementos algebraicos que queremos implementar (enteros módulo un primo, elementos de un cuerpo finito, puntos de una curva elítipca) se pueden manejar muy bien representándolos como objetos y clases.

La estructura en directorios es la siguiente:

```
ccepy/
    ccepy/
    docs/
    tests/
```

La carpeta `ccepy/ccepy` contiene el código fuente los módulos listados en 2.3.3, `ccepy/docs` contiene la documentación y el código que la genera y por último `ccepy/tests` contiene las pruebas unitarias y basadas en propiedades.

El objetivo final es conseguir implementar los protocolos criptográficos explicados en el apartado 2.2. Por ello, explicaremos primero lo más esencial, el módulo de aritmética elemental.

Nota 2.6. Este apartado *no* es la documentación. En esta sección se contarán los detalles técnicos de la implementación como la estructura de datos o lo algoritmos no triviales utilizados. Para ver un listado de las funciones, las clases con sus atributos y sus métodos diríjase a la documentación adjunta en formato de página web.

2.3.4.1 *Módulo de aritmética elemental*

Este módulo permite operar con enteros módulo un primo p y polinomios cuyos coeficientes sean enteros módulo un primo p. El código fuente de este módulo está en el archivo `aritmetica_elemental.py`.

Los enteros módulo un primo se han representado con la clase `EnteroModuloP`. Para tener clases distintas para módulos distintos (para que cada clase tengo como atributo de clase el primo con el que se hace el módulo), se ha imitado la filosofía del patrón *factoría*, esto es, hemos creado una función `Zp()` que acepta como parámetro un número primo y devuelve la clase que representa los enteros módulo dicho primo. Podemos ver la función `Zp` como el constructor de cuerpos $\mathbb{Z}_p$. Esto lo hemos podido hacer ya que `python` permite definir clases dentro de funciones y devolver definiciones de clases como valor de retorno de una función. Además, hemos utilizado el *decorador* `@lru_cache()` del módulo `functools` de la biblioteca estándar de `python` para que llamadas iguales sucesivas a `Zp()` devuelvan el mismo objeto y no se creen copias de una misma clase para un mismo primo.

Además, `EnteroModuloP` es subclase del tipo básico `int`. Así, hemos implementado los operadores $+$, $-$, $*$, $/$ y $**$ simplemente llamando a dichos operadores ya implementados para el tipo de dato `int` y haciendo $\%$ en cada operación. Además, gracias a esta herencia, se permite usar estas operaciones con un dato de tipo `EnteroModuloP` y otro de tipo `int`, devolviendo el resultado como dato de tipo `EnteroModuloP`.

La clase `EnteroModuloP` utiliza el *algoritmo extendido de Euclides* para el cálculo de inversos. El algoritmo, extraído de [3], es el siguiente:

Algoritmo 2.7 (Algoritmo extendido de Euclides para enteros)**.**
ENTRADA: dos enteros no negativos a y b con $a \geqslant b$.
SALIDA: $d = mcd(a, b)$ y enteros x, y satisfaciendo $ax + by = d$.

1. Si $b = 0$, tomar $d \leftarrow a,\ x \leftarrow 1,\ y \leftarrow 0$ y devolver (d, x, y).
2. Inicializar $x_2 \leftarrow 1,\ x_1 \leftarrow 0,\ y_2 \leftarrow 0,\ y_1 \leftarrow 1$.
3. Mientras $b > 0$, hacer lo siguiente:
   a) $q \leftarrow \lfloor a/b \rfloor, r \leftarrow a - qb,\ x \leftarrow x_2 - qx_1,\ y \leftarrow y_2 - qy_1$.
   b) $a \leftarrow b,\ b \leftarrow r,\ x_2 \leftarrow x_1,\ x_1 \leftarrow x,\ y_2 \leftarrow y_1$ y $y_1 \leftarrow y$.
4. Tomar $d \leftarrow a,\ x \leftarrow x_2,\ y \leftarrow y_2$ y devolver (d, x, y).

Este algoritmo está encapsulado en la función `alg_euclides()`. Así, el cálculo del inverso de un elemento $a \in \mathbb{Z}_p$ se reduce a calcular el algoritmo de Euclides para la pareja (a, p) y el inverso es el coeficiente x.

Los polinomios con coeficientes enteros módulo un primo se han representando con la clase `PolinomioZp`. Internamente, cada objeto de `PolinomioZp` contiene una lista de `python` de datos de tipo `EnteroModuloP`

(que representan los coeficientes del polinomio) y un entero (que representa el primo con el que se hizo módulo los coeficientes). La lista se ha definido como una *propiedad* de `python` para permitir solo los accesos de lectura.

`PolinomioZp` implementa los operadores $+$, $-$, $*$, $/$, $\%$ y $**$ para operar entre polinomios más cómodamente. También se ha implementado el método `__hash__()` para poder utilizar posteriormente el decorador `lru_cache()` sobre la función `Fq()` del módulo de cuerpos finitos ya que este decorador requiere que todos los argumentos de la función sobre la que se aplica tengan como representación un número entero y esta representación debe definirse en el método `__hash__()`.

De entre todos los métodos de `PolinomioZp`, destacamos `es_irreducible()`. Como este método utiliza el *algoritmo extendido de Euclides para polinomios*, vamos a detallar primero este.

Algoritmo 2.8 (Algoritmo extendido de Euclides para polinomios)**.**
ENTRADA: dos polinomios $g, h \in \mathbb{Z}_p[X]$.
SALIDA: $d = mcd(g, h)$ y polinomios $s, t \in \mathbb{Z}_p[X]$ satisfaciendo $sg + th = d$.

1. Si $h = 0$, tomar $d \leftarrow g,\ s \leftarrow 1,\ t \leftarrow 0$ y devolver (d, s, t).
2. Inicializar $s_2 \leftarrow 1,\ s_1 \leftarrow 0,\ t_2 \leftarrow 0,\ t_1 \leftarrow 1$.
3. Mientras $h \neq 0$, hacer lo siguiente:
   a) $q \leftarrow g \text{ div } h,\ r \leftarrow g - hq$.
   b) $s \leftarrow s_2 - qs_1,\ t \leftarrow t_2 - qt_1$.
   c) $g \leftarrow h,\ h \leftarrow r$.
   d) $s_2 \leftarrow s_1,\ s_1 \leftarrow s,\ t_2 \leftarrow t_1$ y $t_1 \leftarrow t$.
4. Tomar $d \leftarrow g,\ s \leftarrow s_2,\ t \leftarrow t_2$ y devolver (d, s, t).

Este algoritmo está encapsulado en la función `alg_euclides_polinomios()`. Así pues, el método `es_irreducible()` se encagar de ver si un polinomio es irreducible de manera eficiente en $\mathbb{Z}_p[X]$ y para ello utiliza el siguiente procedimiento.

Algoritmo 2.9.
ENTRADA: un primo p y un polinomio mónico $f(x)$ de grado m en $\mathbb{Z}_p[X]$.
SALIDA: verdadero o falso según $f(x)$ sea irreducible o no.

1. Inicializar $u(x) \leftarrow x$.
2. Para i desde 1 hasta $\lfloor m/2 \rfloor$, hacer lo siguiente:
   a) Calcular $u(x) \leftarrow u(x)^p \bmod f(x)$.
   b) Calcular $d(x) = mcd(f(x), u(x) - x)$ usando el algoritmo 2.8.
   c) Si $d(x) \neq 1$, devolver `Falso`.

3. Devolver `Verdadero`.

Nota 2.10. El algoritmo anterior está basado en los siguientes resultados. Sea p un primo y k un entero.

- El producto de todos los polinomios mónicos irreducibles de $\mathbb{Z}_p[X]$ de grado un divisor de k es igual a $x^{p^k} - x$.
- Sea $f(x)$ un polinomio de grado m en $\mathbb{Z}_p[X]$. Entonces $f(x)$ es irreducible en $\mathbb{Z}_p[X]$ si y solo si $mcd(f(x), x^{p^i} - x) = 1$ para cada i, $1 \leqslant i \leqslant \lfloor m/2 \rfloor$.

Estos dos algoritmos y este par de resultados han sido extraídos de [3].

2.3.4.2 *Módulo de cuerpos finitos*

Este módulo permite operar con elementos de un cuerpos finito de q elementos, donde q será la potencia de un primo. El código fuente de este módulo está en el archivo `cuerpos_finitos.py`.

Los elementos del cuerpo finito p elementos, con p primo, ya están implementados por la clase `EnteroModuloP`. Los elementos del resto de cuerpos finitos, esto es, aquellos con un número de elementos de la forma $q = p^n$ con $n > 1$ se han representado con la clase `ElementoFq`. Análogamente a `EnteroModuloP`, le hemos asociado una función factoría a `ElementoFq`, llamada `Fq()`, que devuelve cada clase para cada número de elementos de un cuerpo finito. También hemos utilizado el decorador `@lru_cache()` sobre `Fq()` para no tener cuerpos finitos iguales duplicados en memoria.

`ElementoFq` es definido como subclase de `PolinomioZp` ya que estamos representado los elementos de un cuerpo finito $\mathbb{F}_q = \mathbb{F}_{p^n}$ como polinomios de $\mathbb{Z}_p[X]$ de hasta grado $n-1$. El polinomio irreducible de $\mathbb{Z}_p[X]$ de grado n puede especificarse o no; si no se especifica, se elige un polinomio irreducible aleatorio de grado n. Los operadores $+$, $-$, $*$, $/$ y $**$ se han implementado usando los operadores ya implementados para `PolinomioZp`.

Para el cálculo de inversos en $\mathbb{F}_q$ se utiliza el algoritmo de Euclides para polinomios de forma análoga a como se hacía para $\mathbb{Z}$. Para calcular potencias hemos utilizado el *algoritmo de exponenciación binaria*, más eficiente que el algoritmo de multiplicar el elemento consigo mismo. El algoritmo, sacado de [3], es el siguiente.

Algoritmo 2.11.
ENTRADA: $g(x) \in \mathbb{F}_{p^n}$, un entero $0 \leqslant k < p^n - 1$ y su representación binaria $k = \sum_{i=0}^{t} k_i 2^i$.
SALIDA: $g(x)^k \bmod f(x)$.

1. Inicializar $s(x) \leftarrow 1$. Si $k = 0$, devolver $s(x)$.
2. Inicializar $G(x) \leftarrow g(x)$.

3. Si $k_0 = 1$, hacer $s(x) \leftarrow g(x)$.
4. Para `i` desde 1 hata `t`, hacer lo siguiente:
   a) Calcular $G(x) \leftarrow G(x)^2 \bmod f(x)$.
   b) Si $k_i = 1$, calcular $s(x) \leftarrow G(x) * s(x) \bmod f(x)$.
5. Devolver $s(x)$.

Para obtener la representación binaria de un entero se ha utilizado la función de la biblioteca estándar de `python` `bin()`.

2.3.4.3 *Módulo de curvas elípticas*

Este módulo permite operar con el grupo de puntos de una curva elíptica. El código fuente de este módulo está en el archivo `curvas_elipticas.py`.

Los puntos de una curva elíptca se han representando con distintas clases según el modelo de curva elítipca. Así, hay una clase para el grupo de puntos $E(\mathbb{F}_q)$, con $\mathbb{F}_q$ de característica distinta de 2 y 3, otra clase para $E(\mathbb{F}_{2^m})$ y una última clase para $E(\mathbb{Q})$. Sin embargo, todas heredan de la misma clase y empezaremos por esta clase madre.

La clase `PuntoRacional` es una clase *abstracta*. Esta clase define el comportamiento de algunos métodos y deja sin implementar otros (similiar a los métodos *virtuales* de C++). En `python`, este tipo de clases se crean utilizando la *metaclase* `ABCMeta` y el decorador `@abstractmethod`, ambos del módulo *abc* de la biblioteca estándar. Así, esta clase define el elemento neutro, que se representa como la tupla `(None, None)`, y especifica que los puntos deben representarse con un atributo para la componente x y otro para la componente y y además restringe el acceso para que sea de solo lectura, a través de las propiedades de `python`. Por último, indica que se deben implementar los métodos `contiene` (que evalúa la ecuación de la curva elíptica dado un par de componentes) y los operadores aritméticos básicos.

La clase que representa $E(\mathbb{F}_q)$ es la clase `PuntoFqRacional`. Análogamente a `EnteroModuloP`, le hemos asociado una función factoría a `PuntoFqRacional`, llamada `curva_eliptica_sobre_Fq()`, que pasandole como argumentos los coeficientes de la ecuación de Weierstrass 4 y el cuerpo base, devuelve `PuntoFqRacional` con sus atributos de clase (los coeficientes de la ecuación, el cuerpo base y el discriminate) inicializados. De esta forma, dos curvas sobre $\mathbb{F}_q$ pero con distinta ecuación (distintos coeficientes) son dos clases distintas.

`PuntoFqRacional` crea un punto de la curva pásandole como argumentos la componente x y la componente y. Si dicho punto no estuviera en la curva, se lanza una excepción. Además, `PuntoFqRacional` implementa los operadores $+$, $-$ y $*$ con su significado natural. Para ello, en los métodos especiales `__add__` y `__neg__` se incluyen las fórmulas de adicción (véase 1.9); el algoritmo eficiente de multiplicación de un punto por un escalar (véase 1.13) se incluye en el método `__mul__`.

Las clases `PuntoF2mRacional` y `PuntoQRacional` que representan respectivamente a $E(\mathbb{F}_{2^m})$ y $E(\mathbb{Q})$ se han implementado de forma similar a `PuntoFqRacional`: se crean mediante una función factoría (`curva_eliptica_sobre_F2m` y `curva_eliptica_sobre_Q` respectivamente), heredan de `PuntoRacional` e implementan los operadores $+$, $-$ y $*$ utilizando las fórmulas de adicción 1.9 y 1.35 respectivamente.

La clase `PuntoQRacional` se ha implementado para mostrar un ejemplo de una curva elítipca definida sobre un cuerpo distinto a los cuerpos finitos. Se ha utilizado el módulo `fractions` de `python` para representar a los racionales. Esta clase no será utilizada en el siguiente módulo a diferencia de las otras dos clases.

2.3.4.4 *Módulo de esquemas criptográficos*

Este módulo permite trabajar con protocolos criptográficos asimétricos, como el esquema Diffie-Hellman conocido como ECDH o el algoritmo de firmas digitales conocido como ECDSA. El código fuente de este módulo está en el archivo `esquemas_criptograficos.py`.

El esquema Diffie-Hellman con curvas elípticas (protocolo 2.4) se ha implementado con la clase `ECDH` mientras que el esquema de firmas digitales con curvas elítipcas (protocolo 2.5) se ha implementado con la clase `ECDSA`. Ambos clases siguen el mismo arquetipo: cada instancia representa un participante del protocolo, requieren como parámetros los parámetros de dominio (véase 2.2) y en la inicialización se generan la pareja de llaves siguiendo el algoritmo 2.3.

Hemos usado el módulo `hashlib` de la biblioteca estándar de `python`, en particular la función `sha1()`, para aplicarle el hash a un mensaje la clase `ECDSA`.

Nótese que la implementación de ambos protocolos se ha conseguido en apenas un par de líneas, similar a las descripciones 2.4 y 2.5. Esto se debe a la implementación de los operadores aritméticos para cada clase y el hecho de haber encapsulado toda la complejidad en las clases de los módulos anteriores, como `PuntoFqRacional` o `ElementoFq`. Gracias a esto, la implementación de un protocolo puede hacerse de forma directa e intuitiva.

2.3.4.5 *Listado de curvas elípticas*

Este módulo secundario consiste en una lista de parámetros de dominio obtenidos de los estándares **??**, **??** y **??**. El código fuente de este módulo está en el archivo `listado_curvas_elipticas.py`.

Este módulo ofrece la función `procesar_parametros_curva_eliptica()`. Pasándole como parámetro el nombre de alguna curva de la del listado, esta función devuelve los parámetros de dominio con los tipos de datos propios de `ccepy`, de forma que se puede usar, por ejemplo, para instanciar un participante de un esquema criptográfico. Los nombres de cada una de las curvas del listado son:

```
Anomalous
NIST P-224
BN(2,254)
brainpoolP256t1
ANSSI FRP256v1
NIST P-256
secp256k1
brainpoolP384t1
NIST P-384
```

2.3.5 *Pruebas*

Se han utilizado dos tipos de pruebas para garantizar la correción del programa: pruebas unitarias y pruebas basadas en propiedades.

Las pruebas unitarias se han añadido en los *docstring* de cada función o clase no trivial. Combinando los módulos `doctest` y `unittest` de la biblioteca estándar de `python`, estas pruebas unitarias se ejecutan automáticamente utilizando los distintos scripts ubicados en `ccepy/tests`. La ventaja de implementar las pruebas unitarias en los docstring es que dichos casos de prueba aparecerán en la documentación automáticamente como ejemplo de uso. Explicaremos esto con más detalle en el apartado **??** de generación de la documentación.

Sin embargo, como se han detectado más errores han sido con las pruebas basadas en propiedades. A diferencia de las pruebas unitarias, en la que se especifica para un caso de prueba la salida que tiene que producir, en las pruebas basadas en propiedades se especifican «propiedades» que debe verificar el código y estas propiedades se verifican para múltiples entradas generadas automáticamente.

Ejemplo 2.12. Supongamos que tenemos la función `suma(x, y)`. Una prueba unitaria en `python` sería:

```
assert suma(2, 2) == 4
```

Con esto sabemos que la función `suma()` funciona para la entrada `(2, 2)`, pero, ¿y el resto de posibles entradas?. El enfoque alternativo es pensar en propiedades que debe verificar la salida de la función, por ejemplo, la conmutatividad. Así, una prueba basada en propiedades sería:

```
assert suma(x, y) == suma(y, x)
```

y la biblioteca utilizada para hacer pruebas basadas en propiedades se encargaría de generar un gran número de posibles entradas y ver que se verifica dicha propiedad.

Como `ccepy` utiliza estructuras algebraicas, es muy fácil pensar en las propiedades que deben verificar las distintas funcionas o clases.

Por ejemplo, para las clases de `python` que representan cuerpos, anillos o grupos se han tomado como propiedades los axiomas de la definición de cuerpo, anillo o grupo respectivamente.

Para los protocolos criptográficos, también podemos encontrar propiedades. Así, para el protocolo ECDH (véase **??**) se comprueba que el punto que calculan Alicia y Bob es el mismo y si Eva intenta calcular el punto (con llave privada distinta) falla. Para el protocolo ECDSA (véase **??**) se comprueba que Bob siempre verifica la firma de Alicia de un mensaje, que dicha firma no sirve para otro mensaje y que si Eva intenta impersonar a Alicia o falsificar firmas, Bob siempre se percata.

Para implementar estas pruebas hemos usado `hypothesis`, la biblioteca de referencia de pruebas basadas en propiedades para el lenguaje `python`. Es fácil de utilizar y presente algunas ventajas, como la búsqueda eficaz de casos extremos o la simplificación de casos que producen fallos. La hemos utilizado junto con el módulo `unittest` para organizar las pruebas en clases y permitir su ejecución forma automática. Nótese que pare ejecutar los scripts con las pruebas debe tener instalado `hypothesis`.

2.3.6 *Generación de la documentación*

La documentación se ha generado con la biblioteca externa `sphinx`. Para ello se ha rellenado los docstring de las funciones, clases y métodos siguiendo la «Guía de Estilo de Google para Python», ya que `sphinx` tiene soporte completo para este estilo de docstring. Se ha decidido usar este estilo de docstring frente a otros ya que este nos ha parecido el más legible.

Para las funciones y los métodos, hemos incluido en los docstring una descripción de la funcionalidad, un ejemplo de uso, los argumentos que requiere y el valor que devuelve. Los docstring de las clases contienen una descripción de la entidad que representan, los argumentos para la creación de un objeto y los atributos tanto de clase como de instancia, además de un ejemplo de creación y uso de una instancia de la clase. Adicionalmente, también se ha rellenado el docstring de cada módulo con una descripción de la funcionalidad que ofrece y ejemplo de uso.

Además de los docstring, `sphinx` también necesita especificar el contenido de cada entrada en la documentación. Esto se hace con ficheros de texto y con el lenguaje de marcado `reStructuredText`. Así, en `ccepy/docs/source` se encuentra el código fuente para generar la documentación en formato `.rst`. El archivo `conf.py` en este mismo directorio alberga las distintas opciones de generación.

Para generar la documentación en formato de página web, basta ejecutar `make html` en el directorio `ccepy/docs` y `sphinx` se generará la página web en `ccepy/build`.

Hemos tenido un problema ya que `sphinx` no reconoce las clases que están definidas dentro de funciones. De este modo, si quiere que aparezcan dichas clases en la documentación, debe copiar la definición de la clase temporalmente fuera de la función, generar la documentación y borrar la definición copiada.