Teoría de Números y Criptografía

F. J. Lobillo

2018/2019

Parte II

Criptografía

Capítulo 1

Criptografía simétrica

1.1 Cifrado y secreto

La primera tarea de la criptografía es proporcionar confidencialidad mediante métodos de cifrado.
Dados conjuntos

- $\mathcal{M}$ el conjunto se los mensajes, textos en claro o *plaintexts*,
- $\mathcal{C}$ el conjunto de los criptogramas o *cyphertexts*,
- $\mathcal{K}_\mathrm{p} \times \mathcal{K}_\mathrm{s}$ el espacio de claves o *key space*,

un cripsosistema viene definido por dos aplicaciones

$$\mathsf{E} : \mathcal{K}_\mathrm{p} \times \mathcal{M} \to \mathcal{C},$$
$$\mathsf{D} : \mathcal{K}_\mathrm{s} \times \mathcal{C} \to \mathcal{M},$$

tales que para cualquier clave $\mathtt{k}_\mathrm{p} \in \mathcal{K}_\mathrm{p}$, existe una clave $\mathtt{k}_\mathrm{s} \in \mathcal{K}_\mathrm{s}$ de manera que dado cualquier mensaje $\mathtt{m} \in \mathcal{M}$,

$$\mathsf{D}(\mathtt{k}_\mathrm{s}, \mathsf{E}(\mathtt{k}_\mathrm{p}, \mathtt{m})) = \mathtt{m}. \tag{1.1}$$

Fijadas claves $k_p \in \mathcal{K}_p$ y su correspondiente $k_s \in \mathcal{K}_s$, se suele utilizar la notación

$$E_{k_p} : \mathcal{M} \to \mathcal{C}, \left[E_{k_p}(m) = E(k_p, m)\right]$$
$$D_{k_s} : \mathcal{C} \to \mathcal{M}, [D_{k_s}(c) = D(k_s, c)]$$

para las funciones de cifrado y descifrado. La propiedad (1.1) se transforma en

$$D_{k_s}(E_{k_p}(m)) = m.$$

En la criptografía clásica, también llamada simétrica, se tiene que $\mathcal{K}_p = \mathcal{K}_s = \mathcal{K}$ y $k_s = k_p = k \in \mathcal{K}$, o al menos hay métodos eficientes para conocer k_s a partir de k_p y viceversa. En la criptografía asimétrica, también llamada de clave pública, no se conocen métodos eficientes para conocer k_s a partir de k_p.

1.2 Objetivos de la criptografía

Confidencialidad La información solo puede ser accesible por las entidades autorizadas.

Integridad La información no ha sido alterada en el envío.

Autenticidad La información proviene de quien afirma haberla enviado.

No repudio El emisario de una información no puede a posteriori negar que ha realizado tal envío.

1.3 Ataques

El criptoanálisis es la disciplina encargada de tratar de averiguar el mensaje o la clave empleada. Desde el punto de vista de la seguridad se parte del conocido como *Principio de Kerckhoffs*, que establece que el adversario conoce todos los detalles del criptosistema excepto la clave empleada. Es decir, la seguridad debe recaer en el secreto de la clave empleada para descifrar.
Los posibles ataques se clasifican como sigue:

Criptograma El adversario conoce el criptograma. Esta situación siempre se da.

Mensaje conocido El atacante conoce parejas mensaje/criptograma cifradas con una misma clave.

Mensaje escogido El atacante puede generar criptogramas para mensajes de su elección. Una vez obtenidas dichas parejas, trata de averiguar el mensaje correspondiente a un criptograma desconocido.

Mensaje escogido-adaptativo El atacante no solo puede generar parejas mensaje/criptograma a su elección, sino que puede hacerlo tantas veces como quiera realizando los análisis que considere oportunos entre medias.

Criptograma escogido y escogido-adaptativo Similar a los anteriores pero partiendo del criptograma, es decir, tiene acceso a descifrar los criptogramas que desee, inicialmente o a lo largo del proceso. Evidentemente este ataque busca la clave.

1.4

Seguridad probable

El primer intento de formalizar el concepto de seguridad de un criptosistema se debe a C. E. Shannon. Define lo que se conoce como un cifrado *perfectamente secreto*, aquél que resiste cualquier ataque al criptograma. Es decir, el criptograma no aporta ninguna información sobre el mensaje, incluso bajo la hipótesis de que el atacante dispone de capacidad de cálculo y tiempo ilimitados. El cifrado de Vernam es el más conocido entre los perfectamente secretos. Si asumimos que nuestro mensaje es una cadena de bits, que podemos identificar con $\mathbb{Z}_2$, la clave va a ser una cadena aleatoria del mismo tamaño, y el cifrado consiste en realizar la suma módulo 2 de cada bit del mensaje con cada bit de la clave[1]. El descifrado consiste en realizar la suma del criptograma con la misma clave.

Este cifrado es tan seguro como "inútil", ya que transferir la clave cuesta el mismo trabajo que transmitir el mensaje. Si hay un canal seguro para la clave, se puede emplear el mismo canal para el mensaje. Además, cada clave debe usarse una sola vez, ya que

$$c = m \oplus k \text{ y } c' = m' \oplus k \Rightarrow c \oplus c' = m \oplus m',$$

por lo que perdemos la aleatoriedad.

La dificultad de usar cifrados perfectamente secretos ha conducido a analizar la seguridad desde puntos de vista mas laxos. Concretamente se abandona la hipótesis de que el atacante tiene capacidad de cálculo ilimitada. Se pasa a considerar criptosistemas que resisten ataques

[1] La suma en $\mathbb{Z}_2 = \mathbb{B}$ suele representarse en el mundo de la informática mediante el símbolo $\oplus$ o mediante `XOR`.

factibles. Un ataque factible es aquel que puede realizarse mediante un algoritmo eficiente. El punto de vista más aceptado establece que los algoritmos polinomiales son eficientes, mientras que los no polinomiales no lo son. Hablaremos de la complejidad algorítmica en el Capítulo 2. La Teoría de Números viene siendo empleada desde los años 70 del siglo pasado como fuente de problemas que mezclan algoritmos eficientes con otros que no lo son.

1.5 Criptografía simétrica

Un criptosistema simétrico, como hemos dicho antes, viene determinado por dos aplicaciones

$$\begin{aligned} E &: \mathcal{K} \times \mathcal{M} \to \mathcal{C}, \\ D &: \mathcal{K} \times \mathcal{C} \to \mathcal{M}, \end{aligned}$$

tales que para cualquier clave $k \in \mathcal{K}$ y cualquier mensaje $m \in \mathcal{M}$,

$$D(k, E(k, m)) = m. \tag{1.2}$$

Por supuesto, fijada $k \in \mathcal{K}$ se suele utilizar la notación

$$\begin{aligned} E_k &: \mathcal{M} \to \mathcal{C}, \\ D_k &: \mathcal{C} \to \mathcal{M}, \end{aligned}$$

para las funciones de cifrado y descifrado. La propiedad (1.2) se transforma en

$$D_k(E_k(m)) = m.$$

La criptografía simétrica moderna es criptografía digital. Los conjuntos $\mathcal{M}$, $\mathcal{C}$ y $\mathcal{K}$ son cadenas de bits, los primeros $\mathcal{M} = \mathcal{C} = \mathbb{B}^*$ y el espacio de claves $\mathcal{K} = \mathbb{B}^K$.

1.6 Cifrados de flujo

Sean $\mathcal{M} = \mathcal{C} = \mathbb{B}^*$ y $\mathcal{K} = \mathbb{B}^K$ para cierto $K \in \mathbb{N}$. Sea

$$f : \mathcal{K} \times \mathcal{M} \to \mathbb{B}^*$$

una aplicación tal que $f(k, m)_i$ sólo depende de $m_{[0,i-1]} = m_0 \cdots m_{i-1}$. El cifrado de flujo asociado a f es

$$E_k(m) = (m_i \oplus f(k, m_{[0,i-1]})_i)_{i \geq 0}$$

El cifrado de Vernam es un cifrado de flujo en el que f no depende del mensaje y genera una sucesión aleatoria.
Los más famosos son

- RC4
- A5/1, A5/2
- Portfolio eSTREAM
- Cualquier cifrado de bloque en modo feedback.

Cifrados de flujo síncronos. La sucesión criptográfica se genera independientemente del mensaje y del criptograma, es decir,

$$f : \mathcal{K} \to \mathbb{B}^*$$

Satisface las ecuaciones:

$$\begin{aligned} \sigma_{i+1} &= g(\sigma_i, k), \\ z_i &= f(\sigma_i, k), \\ c_i &= z_i \oplus m_i, \end{aligned}$$

donde σ_0 es el estado inicial, k es la clave, f es la función *siguiente estado*, f produce la sucesión criptográfica z_i que es sumada (XOR) con mensaje m_i para generar el criptograma c_i.

Cifrados de flujo autosincronizables. La sucesión criptográfica se genera a partir de la clave y de una cantidad fija de bits en el criptograma, matemáticamente

$$\begin{aligned} \sigma_i &= (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}), \\ z_i &= f(\sigma_i, k) \\ c_i &= z_i \oplus m_i, \end{aligned}$$

donde $\sigma_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$ es el estado inicial, k es la clave, f es la función que genera la sucesión criptográfica z_i que es sumada (XOR) con el mensaje m_i para generar el criptograma c_i.

1.7 Cifrados de bloque

Son criptosistemas en los que mensajes, criptogramas y claves están limitados a cadenas de una longitud fija, formalmente,

$$E : \mathbb{B}^K \times \mathbb{B}^N \to \mathbb{B}^N,$$
$$D : \mathbb{B}^K \times \mathbb{B}^N \to \mathbb{B}^N,$$

o para cada clave $k \in \mathbb{B}^K$,

$$E_k : \mathbb{B}^N \to \mathbb{B}^N,$$
$$D_k : \mathbb{B}^N \to \mathbb{B}^N.$$

N se conoce como el tamaño del bloque y a K como el tamaño de la clave.
Algunos de los más conocidos son

- DES
- IDEA
- Blowfish
- AES (Rijndael)

1.7.1 *Modos de operación*

Cómo se extiende E de $\mathbb{B}^N$ a $\mathbb{B}^*$ es competencia de los modos de operación. Estos modos dependen del tamaño del bloque, no de la clave.

Electronic CodeBook.

1.1 (ECBencrypt)**. Input:** $m \in \mathbb{B}^*$
Output: $c \in \mathbb{B}^*$
Dividimos $m = m_{[1]} \cdots m_{[l]}$ con $m_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**
$c_{[i]} = E_k(m_{[i]})$.
return $c_{[1]} \cdots c_{[l]}$

1.2 (ECBdecrypt)**. Input:** $c \in \mathbb{B}^*$
Output: $m \in \mathbb{B}^*$
Dividimos $c = c_{[1]} \cdots c_{[l]}$ con $c_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**
$m_{[i]} = D_k(c_{[i]})$.
return $m_{[1]} \cdots m_{[l]}$

Cipher-Block Chaining.

1.3 (CBCencrypt)**. Input:** $m \in \mathbb{B}^*$
Output: $c \in \mathbb{B}^*$
$c_{[0]} \in \mathbb{B}^N$ {Es lo que se conoce como IV}
dividimos $m = m_{[1]} \cdots m_{[l]}$ con $m_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**
$c_{[i]} = E_k(m_{[i]} \oplus c_{[i-1]})$.
return $c_{[0]} \cdots c_{[l]}$

1.4 (CBCdecrypt)**. Input:** $c \in \mathbb{B}^*$
Output: $m \in \mathbb{B}^*$
dividimos $c = c_{[0]} \cdots c_{[l]}$ con $c_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**

$m_{[i]} = D_k(c_{[i]}) \oplus c_{[i-1]}$.
return $m_{[1]} \cdots m_{[l]}$

Cipher FeedBack.

1.5 (CFBencrypt)**. Input:** $m \in \mathbb{B}^*$, $1 \leq r \leq N$
Output: $c \in \mathbb{B}^*$
$x_{[1]} \in \mathbb{B}^N$ {Es lo que se conoce como IV}
dividimos $m = m_{[1]} \cdots m_{[l]}$ con $m_{[i]} \in \mathbb{B}^r$.
for $i = 1, \ldots, l$ **do**
$c_{[i]} = m_{[i]} \oplus \text{msb}_r(E_k(x_{[i]}))$.
$x_{[i+1]} = \text{lsb}_{N-r}(x_{[i]}) \| c_{[i]}$
return $c_{[1]} \cdots c_{[l]}$

1.6 (CFBdecrypt)**. Input:** $c \in \mathbb{B}^*$, $1 \leq r \leq N$, $x_{[1]} \in \mathbb{B}^N$
Output: $m \in \mathbb{B}^*$
dividimos $c = c_{[1]} \cdots c_{[l]}$ con $c_{[i]} \in \mathbb{B}^r$.
for $i = 1, \ldots, l$ **do**
$m_{[i]} = c_{[i]} \oplus \text{msb}_r(E_k(x_{[i]}))$.
$x_{[i+1]} = \text{lsb}_{N-r}(x_{[i]}) \| c_{[i]}$
return $m_{[1]} \cdots m_{[l]}$

Output FeedBack.

1.7 (OFBencrypt)**. Input:** $m \in \mathbb{B}^*$
Output: $c \in \mathbb{B}^*$
$x_{[0]} \in \mathbb{B}^N$ {Es lo que se conoce como IV}
dividimos $m = m_{[1]} \cdots m_{[l]}$ con $m_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**

$x_{[i]} = E_k(x_{[i-1]})$
$c_{[i]} = m_{[i]} \oplus x_{[i]}$.
return $c_{[1]} \cdots c_{[l]}$

1.8 (OFBdecrypt)**. Input:** $c \in \mathbb{B}^*$, $x_{[0]} \in \mathbb{B}^N$
Output: $m \in \mathbb{B}^*$
dividimos $c = c_{[1]} \cdots c_{[l]}$ con $c_{[i]} \in \mathbb{B}^N$.
for $i = 1, \ldots, l$ **do**
$x_{[i]} = E_k(x_{[i-1]})$
$m_{[i]} = c_{[i]} \oplus x_{[i]}$.
return $m_{[1]} \cdots m_{[l]}$

Ejercicio 1.1 (MiniAES)**.** MiniAES fue publicado en el año 2002 por R. C-W Phan, y es una versión reducida de AES que conserva su estructura pero permite hacer los cálculos de forma más comprensible para mejorar el entendimiento académico. La versión aquí propuesta difiere de la original en la función de sustitución.
En primer lugar se trabaja con el cuerpo de 16 elementos en lugar de con el cuerpo de 256 elementos. Dicho cuerpo se representa mediante $\mathbb{F}_{16} = \mathbb{Z}_2(\xi)_{\xi^4+\xi+1}$, es decir, la suma es bit a bit pero el producto es el producto polinomial módulo $\xi^4 + \xi + 1$. Los elementos de $\mathbb{F}_{16}$ se representan como cadenas de 4 bits o como un dígito hexadecimal, según el esquema

$$1011 = \texttt{0xB} = \xi^3 + \xi + 1,$$

es decir, identificamos $\mathbb{F}_{16} \cong \mathbb{F}_2^4$ a través de la base $\{\xi^3, \xi^2, \xi, 1\}$. La función de sustitución aquí empleada $\gamma : \mathbb{F}_{16} \to \mathbb{F}_{16}$ es la siguiente. Dado $x_3x_2x_1x_0 \in \mathbb{F}_2^4$, denotamos

$$\mathrm{inv}(x_3x_2x_1x_0) = \begin{cases} (x_3x_2x_1x_0)^{-1} & \text{si } x_3x_2x_1x_0 \neq 0000, \\ 0000 & \text{si } x_3x_2x_1x_0 = 0000. \end{cases}$$

Nuestra función es

$$\gamma(x_3x_2x_1x_0) = \mathrm{inv}(x_3x_2x_1x_0) \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} + (0011).$$

El bloque básico sobre el que actúa Mini AES es un bloque de 16 bits que se representa como una matriz 2×2 con coeficientes en $\mathbb{F}_{16}$ rellena por columnas, es decir,

$$\begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$

La función de sustitución `Sub` es la extensión de γ a cada valor de la matriz, es decir,

$$\gamma \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix} = \begin{bmatrix} \gamma(a_0) & \gamma(a_2) \\ \gamma(a_1) & \gamma(a_3) \end{bmatrix}.$$

La función `ShiftRow`, representada por π, realiza un desplazamiento en la última fila del bloque, es decir,

$$\pi \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_2 \\ a_3 & a_1 \end{bmatrix}$$

La función `MixColumn`, representada por θ, actúa mediante la siguiente descripción,

$$\theta \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix} = \begin{bmatrix} 0011 & 0010 \\ 0010 & 0011 \end{bmatrix} \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$

Por último la clave se añade de la misma forma que en el AES general, mediante la suma lógica XOR, y se representa mediante σ_{K_i}. Mini AES consta de dos rondas más una ronda cero al igual que AES. La última ronda no incluye `MixColumn`, así tenemos la siguiente descripción:

$$E_K = \sigma_{K_2} \circ \pi \circ \gamma \circ \sigma_{K_1} \circ \theta \circ \pi \circ \gamma \circ \sigma_{K_0}$$

La claves de ronda se obtienen

K_0	$w_0 = k_0$
	$w_1 = k_1$
	$w_2 = k_2$
	$w_3 = k_3$
K_1	$w_4 = w_0 \oplus \gamma(w_3) \oplus 0001$
	$w_5 = w_1 \oplus w_4$
	$w_6 = w_2 \oplus w_5$
	$w_7 = w_3 \oplus w_6$
K_2	$w_8 = w_4 \oplus \gamma(w_7) \oplus 0010$
	$w_9 = w_5 \oplus w_8$
	$w_{10} = w_6 \oplus w_9$
	$w_{11} = w_7 \oplus w_{10}$

donde $K = (k_0, k_1, k_2, k_3)$ es la clave de 16 bits.

1. Calcula explícitamente la función γ.
2. Calcula γ', θ', π', K_0', K_1', K_2' tales que
$$D_K = \sigma_{K_2'} \circ \pi' \circ \gamma' \circ \sigma_{K_1'} \circ \theta' \circ \pi' \circ \gamma' \circ \sigma_{K_0'}.$$
3. Calcula $c = E_{\mathtt{dni}}(\mathtt{0xA136})$ donde $\mathtt{dni}$ es el número de tu DNI módulo 65536 en binario.
4. Calcula $D_{\mathtt{dni}}(c)$.
5. Calcula $E_{\mathtt{dni}}(\mathtt{0xA036})$. Compara el resultado con c.

Ejercicio 1.2. Una de las herramientas más empleadas en el diseño de cifrados de flujo son los LFSR (Registros lineales retroalimentados de

desplazamiento). Un LFSR de orden t (homogéneo) viene dado por una aplicación lineal

$$f : \mathbb{F}_q^t \to \mathbb{F}_q, \quad [(x_0, \dots, x_{t-1}) \mapsto k_0 x_0 + \cdots + k_{t-1} x_{t-1}]$$

para ciertos $k_0, \dots, k_{t-1} \in \mathbb{F}_q$, que consideramos como la clave del criptosistema. Dado un estado inicial $(z_0, \dots, z_{t-1}) \in \mathbb{F}_q$, el vector de inicialización IV, se construye la sucesión

$$z_n = f(z_{n-t}, z_{n-t+1}, \dots, z_{n-1}).$$

La función de cifrado es

$$E_{k_0,\dots,k_{t-1}}(m_0 m_1 \cdots) = c_0 c_1 \cdots$$

donde

$$c_i = m_i + z_i.$$

En este ejercicio estamos considerando como alfabeto $\mathbb{F}_q$ en lugar del usual $\mathbb{B} = \mathbb{F}_2$. Supongamos que $q = 2^5$ y vemos $\mathbb{F}_{2^5} = \mathbb{F}_2(\xi)_{\xi^5+\xi^2+1}$. Identificamos los elementos de $\mathbb{F}_{2^5} = \mathbb{F}_2^5$, es decir, cada elemento se representa como una cadena de 5 bits según el esquema

$$10110 = \xi^3 + \xi^2 + 1.$$

Codificamos caracteres mediante 5 bits, el 00000 corresponde al espacio, los números del 00001 al 11011 son los 27 caracteres del alfabeto latino incluyendo la letra Ñ, los números del 11100 al 11111 son los signos de puntuación `.,;:` respectivamente. Este sistema nos permite convertir los caracteres a una cadena de bits y de cadena de bits a lista de caracteres.

Partimos de una clave $\mathtt{key} = (k_0, k_1, k_2, k_3) \in \mathbb{F}_{2^5}^4$. Ciframos un cierto texto, y obtenemos el criptograma

```
.LJMJ,RYRQVSNNQTFDWH.Ñ UAQTGI;Ñ.L,
```

donde los cuatro primeros caracteres se corresponden con el IV. Sabiendo que firmo mis mensajes con la cadena `JAVIER.`, descifra el mensaje.

Ejercicio 1.3. Demostrar que un cifrado por bloques en modo CFB y OFB es un cifrado de flujo. ¿De qué tipo?

Ejercicios de evaluación de Criptosistemas simétricos

Ejercicio. Consideremos el cifrado por bloques MiniAES descrito en el ejercicio 1.1.

1. Calcula $E_{dni}(0x01234567)$ usando el modo OFB e IV $= 0x0001$.
2. Calcula $E_{dni}(0x01234567)$ usando el modo CFB, $r = 3$ o $r = 10$, y vector de inicialización IV $= 0x0001$.

Capítulo 2

Complejidad algorítmica

2.1 Introducción

Notación big-$\mathcal{O}$. Sean $f, g : \mathbb{N}^r \to \mathbb{R}$. Decimos que $f = \mathcal{O}(g)$ si existe una constante $c \in \mathbb{R}$ tal que $f(n_1, \ldots, n_r) \leq cg(n_1, \ldots, n_r)$ para cualquier $(n_1, \ldots, n_r) \in \mathbb{N}^r$.
Esta notación se emplea para analizar el tiempo esperado que va a tardar un algoritmo en realizar una tarea.

Número de dígitos. Todo entero $b^{k-1} \leq n < b^k$ tiene exactamente k dígitos en base b. Por tanto, el número de dígitos en base b de un entero positivo n viene dado por la fórmula

$$\lfloor \log_b n \rfloor + 1 = \left\lfloor \frac{\log n}{\log b} \right\rfloor + 1.$$

Complejidad de la aritmética. Sean $n, m \in \mathbb{N}$. Para calcular $n + m$ empleamos la representación en alguna base de numeración de ambos números. El algoritmo escolar va calculando los dígitos de la suma uno

a uno. Para cada dígito hay que emplear un máximo de dos sumas, los dígitos correspondientes más el posible acarreo de la suma anterior. El caso de la resta es análogo.

Proposición 2.1. *Si* $t(n, m)$ *es la función que mide el tiempo necesario para calcular* $n \pm m$*, entonces*

$$t = \mathcal{O}\left(\text{máx}(\log n, \log m)\right).$$

La complejidad del producto es también sencilla de calcular.

Proposición 2.2. *Si* $t(n, m)$ *es la función que mide el tiempo necesario para calcular* nm*, entonces*

$$t = \mathcal{O}\left((\log n)(\log m)\right).$$

Demostración. Para multiplicar n por un dígito en base b, hay que realizar $\lfloor \log_b n \rfloor + 1$ multiplicaciones de un dígito más $\lfloor \log_b n \rfloor$ sumas correspondientes al acarreo. Para multiplicar por un m cualquiera hay que realizar la tarea anterior $\lfloor \log_b m \rfloor + 1$ veces, más $\lfloor \log_b m \rfloor$ sumas de números con $\lfloor \log_b n \rfloor + \lfloor \log_b m \rfloor + 1$ dígitos. Por tanto

$$\begin{aligned} t = \mathcal{O}\,(2(\lfloor \log_b n \rfloor + 1)(\lfloor \log_b m \rfloor + 1) \\ + \lfloor \log_b n \rfloor + \lfloor \log_b m \rfloor + 1) = \mathcal{O}\left((\log n)(\log m)\right). \end{aligned}$$

□

Proposición 2.3. *Sean* $n \geq m \in \mathbb{N}$*. Si* $t(n, m)$ *es la función que mide el tiempo necesario para calcular cociente y resto de la división de* n *entre* m *escritos en binario, entonces*

$$t = \mathcal{O}\left((\log n)(\log m)\right).$$

Demostración. Durante $\lfloor \log_2 n \rfloor - \lfloor \log_2 m \rfloor$ veces debemos hacer un máximo de $\lfloor \log_2 m \rfloor$ comparaciones de bits y la resta de un número de $\lfloor \log_2 m \rfloor + 1$ dígitos y otro de $\lfloor \log_2 m \rfloor$. Por tanto

$$t = \mathcal{O}\left(\left(\lfloor \log_2 n \rfloor - \lfloor \log_2 m \rfloor\right) \lfloor \log_2 m \rfloor\right),$$

lo que implica $t = \mathcal{O}\left((\log n)(\log m)\right)$ □

Corolario 2.4. *El tiempo empleado en representar un número* n *en binario a una base* b *es* $\mathcal{O}\left((\log n)^2\right)$.

Demostración. Debemos realizar

$$\lfloor \log_b n \rfloor + 1 = \left\lfloor \frac{\log n}{\log b} \right\rfloor + 1$$

divisiones de números menores o iguales que n entre b. Por la Proposición 2.3, cada una de estas divisiones tiene un coste $\mathcal{O}\left((\log n)(\log b)\right)$, de donde se obtiene el resultado ya que $\log b$ es constante. □

Corolario 2.5. *El tiempo empleado en pasar a binario un entero* n *escrito en base* b *es* $\mathcal{O}\left((\log n)^2\right)$.

Demostración. Calcular cociente y resto obtenidos al dividir n en base b entre 2 requiere $\lfloor \log_b n \rfloor + 1$ multiplicaciones y restas. Esta tarea hay que repetirla con los cocientes, cuyo tamaño siempre es menor que el de n, y hay a lo sumo $\lfloor \log_b n \rfloor$ divisiones. □

Corolario 2.6. *El tiempo empleado sumar dos elementos de* $\mathbb{Z}_n$ *es* $\mathcal{O}(\log n)$. *El producto de dos elementos en* $\mathbb{Z}_n$ *puede hacerse en tiempo* $\mathcal{O}\left((\log n)^2\right)$.

Proposición 2.7. *Sean* $n \geq m \in \mathbb{N}$. *Si* $t(n,m)$ *es la función que mide el tiempo necesario para calcular* d, u, v *tales que* $d = (n,m)$ *y* $d = un + vm$ *mediante el algoritmo extendido de Euclides, entonces*

$$t = \mathcal{O}\left((\log n)^3\right).$$

Demostración. Recordemos que el algoritmo extendido de Euclides puede presentarse de la siguiente forma:

Input: $n \geq m$.
Output: d, u, v tales que $un + vm = d = (n,m)$.

1: $r_0 \leftarrow n$, $r_1 \leftarrow m$
2: $u_0 \leftarrow 1$, $u_1 \leftarrow 0$
3: $v_0 \leftarrow 0$, $v_1 \leftarrow 1$
4: **repeat**
5: $q \leftarrow r_0 \div r_1$, $r \leftarrow r_0$ mód r_1 {Cociente y resto de la división euclídea}
6: $u \leftarrow u_0 - qu_1$, $v \leftarrow v_0 - qv_1$
7: $r_0 \leftarrow r_1$, $r_1 \leftarrow r$
8: $u_0 \leftarrow u_1$, $u_1 \leftarrow u$
9: $v_0 \leftarrow v_1$, $v_1 \leftarrow v$
10: **until** $r = 0$
11: **return** r_0, u_0, v_0.

Cada iteración realiza una división euclídea, con un coste $\mathcal{O}\left((\log n)^2\right)$ por la Proposición 2.3. Nos queda acotar el número de repeticiones que realiza el algoritmo. Observemos que $r < \frac{r_0}{2}$: si $r_1 \leq \frac{r_0}{2}$, obviamente $r < r_1 \leq \frac{r_0}{2}$; si $r_1 > \frac{r_0}{2}$, tenemos que $q = 1$ y $r = r_0 - r_1 < r_0 - \frac{r_0}{2} = \frac{r_0}{2}$. Por tanto, el número de repeticiones viene acotado por $2\log_2 n$. En consecuencia, el algoritmo extendido de Euclides es $\mathcal{O}\left((\log n)^3\right)$. □

Proposición 2.8. *El tiempo empleado en calcular* b^m mód n *está en* $\mathcal{O}\left((\log n)^2(\log m)\right)$.

Demostración. Mediante el método de los cuadrados iterados, para calcular b^m mód n hay que realizar $3\log_2 m$ multiplicaciones en $\mathbb{Z}_n$. □

Ejercicios de Complejidad algorítmica

Ejercicio 2.1. Calcula una cota superior para el tiempo estimado en calcular $n!$.

Ejercicio 2.2. Calcula una cota superior para el tiempo estimado en calcular $\binom{n}{m}$

Ejercicio 2.3. Supongamos que $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lambda \in \mathbb{R}$. Demuestra que $f = \mathcal{O}(g)$. Como consecuencia demuestra que $\log n = \mathcal{O}(n^\epsilon)$ para cualquier $\epsilon > 0$.

Ejercicio 2.4. Sean $f_1 \in \mathcal{O}(g_1)$ y $f_2 \in \mathcal{O}(g_2)$. Calcula una cota para la complejidad de $f_1 + f_2$ y $f_1 f_2$.

Ejercicio 2.5. Estudia la complejidad de la suma y el producto de dos polinomios de grados n_1 y n_2 con coeficientes en $\mathbb{Z}_m$.

Ejercicio 2.6. Sabemos que

$$\sum_{j=1}^{n} j^2 = \frac{n(n+1)(2n+1)}{6}.$$

Estudia la complejidad del cálculo de cada uno de los términos de la fórmula anterior.

Ejercicio 2.7. Supongamos que n es un primo escrito en binario. Da un algoritmo que calcule $\lfloor \sqrt{n} \rfloor$ en $\mathcal{O}\left((\log n)^3\right)$.

Ejercicio 2.8 (Algoritmo de Karatsuba)**.** Sea $x = x_0 + x_1\beta + \cdots + x_{n-1}\beta^{n-1}$ con n par. En base $\beta^{\frac{n}{2}}$ se representa:

$$x = x_{(0)} + x_{(1)}\beta^{\frac{n}{2}}$$

Sean a, b enteros de n dígitos en base β. El producto clásico se describe como

$$a \cdot b = a_{(0)} \cdot b_{(0)} + (a_{(1)} \cdot b_{(0)} + a_{(0)} \cdot b_{(1)}) \cdot \beta^{\frac{n}{2}} + a_{(1)} \cdot b_{(1)} \cdot \beta^n$$

La idea de Karatsuba:

$$\begin{aligned} a \cdot b = {} & a_{(0)} \cdot b_{(0)} \\ & + (a_{(0)} \cdot b_{(0)} + a_{(1)} \cdot b_{(1)} + (a_{(1)} - a_{(0)}) \cdot (b_{(0)} - b_{(1)})) \cdot \beta^{\frac{n}{2}} \\ & + a_{(1)} \cdot b_{(1)} \cdot \beta^n \end{aligned}$$

Útil para $n = 2^m$ dígitos. El algoritmo podemos escribirlo como sigue.

KARATSUBA(a, b, m, β) :
Input: a, b, m, β
Output: ab
  if $m = 0$ **then**
    return $a \cdot b$
  else
    $a1 \leftarrow 2^{m-1}$ dígitos más significativos de a
    $a0 \leftarrow 2^{m-1}$ dígitos menos significativos de a
    $b1 \leftarrow 2^{m-1}$ dígitos más significativos de b
    $b0 \leftarrow 2^{m-1}$ dígitos menos significativos de b

$t1 \leftarrow \text{KARATSUBA}(a1, b1, m-1, \beta)$
$t2 \leftarrow \text{KARATSUBA}(a1 - a0, b0 - b1, m-1, \beta)$
$t3 \leftarrow \text{KARATSUBA}(a0, b0, m-1, \beta)$
return $t1 \cdot \beta^{2^m} + (t1 + t2 + t3) \cdot \beta^{2^{m-1}} + t3$

Calcula la complejidad del mismo.

Ejercicios de evaluación de Complejidad algorítmica

Ejercicio 2.9. Supongamos que tenemos almacenada una lista de todos los primos menores que $\sqrt{n}$. Estudia la complejidad estimada en determinar si n es primo dividiendo entre todos los primos de la lista anterior. Recordemos que el teorema del número primo dice

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\log n} = 1,$$

donde $\pi(n)$ es en número de primos menores que n.

Ejercicio 2.10. Calcula una estimación del tiempo necesario para saber n es divisible por un primo $\leq m$, suponiendo que tenemos una lista con todos los primos menores o iguales que m.

Ejercicio 2.11. Calcula la complejidad del producto de matrices con coeficientes en $\mathbb{Z}_m$ en función del tamaño de las mismas.

Capítulo 3

Criptosistema de Rivest-Shamir-Adleman

3.1 Función unidireccional

Sean p, q dos primos impares, $n = pq$, e un entero primo relativo con $\varphi(n)$, i.e. $(e, \varphi(n)) = 1$ y d su inverso, es decir

$$ed \equiv 1 \mod \varphi(n).$$

Definimos la función

$$\mathrm{RSA}_{n,e} : \mathbb{Z}_n \to \mathbb{Z}_n \left[\mathrm{RSA}_{n,e}(m) = m^e \mod n\right].$$

Teorema 3.1. *La función* $\mathrm{RSA}_{n,e}$ *es biyectiva con inversa* $\mathrm{RSA}_{n,e}^{-1} = \mathrm{RSA}_{n,d}$.

Demostración. Por la simetría de la construcción, basta comprobar que $m = \mathrm{RSA}_{n,d}(\mathrm{RSA}_{n,e}(m)) = m^{ed} \mod n$ para cualquier $m \in \mathbb{Z}_n$. Sea

$$\begin{aligned} \Phi : \mathbb{Z}_n &\to \mathbb{Z}_p \times \mathbb{Z}_q \\ x &\mapsto (x \mod p, x \mod q) \end{aligned} \tag{3.1}$$

el isomorfismo de anillos canónico dado por el Teorema Chino de los Restos (CRT). Sea $m \in \mathbb{Z}_n$, entonces

$$\Phi(m)^{ed} = (m^{ed} \text{ mód } p, m^{ed} \text{ mód } q).$$

Dado que

$$ed = 1 + t\varphi(n) = 1 + t(p-1)(q-1),$$

si $(m, p) = 1$, se deduce del Teorema Pequeño de Fermat que

$$m^{ed} = m(m^{p-1})^{t(q-1)} \equiv m \mod p,$$

y si $m = ap$, obviamente $m^{ed} \equiv m \mod p$, de donde

$$m^{ed} \equiv m \mod p$$

para cualquier $m \in \mathbb{Z}_n$. Análogamente

$$m^{ed} \equiv m \mod q,$$

de donde

$$\Phi(m)^{ed} = \Phi(m)$$

para todo $m \in \mathbb{Z}_n$. Como Φ es isomorfismo de anillos, tenemos que

$$m^{ed} \equiv m \mod n$$

para todo $m \in \mathbb{Z}_n$, lo que demuestra el teorema. □

Por la Proposición 2.8, la función $RSA_{n,e}$ es una función rápida de calcular. Concretamente, el cálculo

$$m^e \text{ mód } n$$

está en $\mathcal{O}\left((\log n)^3\right)$ por la Proposición 2.8, ya que $\varphi(n) < n$.
El Teorema 3.1 nos dice que, conocido d, $\mathrm{RSA}_{n,e}^{-1}$ también es fácil de calcular. No hay publicados algoritmos de complejidad polinomial para el cálculo de $\mathrm{RSA}_{n,e}^{-1}(m) = \sqrt[e]{m}$ mód n sin el conocimiento de d.
Por la Proposición 2.7, es computacionalmente eficiente conocer d si se conoce $\varphi(n)$.

Proposición 3.2. *Supongamos que es sabido que n es el producto de dos primos p, q. Dados p, q podemos calcular $\varphi(n)$ en $\mathcal{O}(\log n)$ operaciones. Dados n y $\varphi(n)$, se pueden calcular p, q en $\mathcal{O}\left((\log n)^2\right)$ operaciones.*

Demostración. Dado que $\varphi(n) = (p-1)(q-1) = n + 1 - (p + q)$, calcular $\varphi(n)$ requiere una suma y una resta, por tanto estamos en $\mathcal{O}(\log n)$.
Supongamos que conocemos n y $\varphi(n)$. Podemos suponer que n es impar, pues el caso par es trivial. Como

$$\begin{aligned}(x-p)(x-q) &= x^2 - (p+q)x + pq \\ &= x^2 - (n+1-\varphi(n))x - n,\end{aligned}$$

tenemos que p y q son raíces del polinomio anterior, es decir, $p, q = b \pm \sqrt{b^2 - n}$, donde $p + q = 2b$. Como el cálculo de raíces cuadradas es $\mathcal{O}\left(\left(\log(b^2-n)\right)^2\right) = \mathcal{O}\left((\log n)^2\right)$, tenemos el resultado. □

Ejemplo 3.3. Vamos a descomponer $n = 1189$ sabiendo que $\varphi(n) = 1120$. Tenemos que $2b = p+q = n+1-\varphi(n) = 1189+1-1120 = 70$, luego $b \pm \sqrt{b^2 - n} = 35 \pm \sqrt{35^2 - 1189} = 35 \pm \sqrt{1125 - 1189} = 35 \pm \sqrt{36} = 35 \pm 6 = 41,\ 29$.

Computacionalmente, calcular $\varphi(n)$ y factorizar n son problemas equivalentes. Una forma de romper RSA sin factorizar n pasa por calcular d tal que

$$a^{ed} \equiv a \mod n$$

para cualquier $a \in \mathbb{Z}_n$ primo con n. Vía el isomorfismo Φ en (3.1), tenemos que $k = ed - 1$ es múltiplo de $[p-1, q-1]$. Supongamos que conocemos k tal que $a^k \equiv 1 \mod n$ para cualquier a primo con n. En particular k es par porque $(-1)^k \equiv 1 \mod n$.

Lema 3.4. *Supongamos que existe* $a_0 \in \mathcal{U}(\mathbb{Z}_n)$ *tal que* $a_0^{k/2} \not\equiv 1 \mod n$. *Entonces*

$$\left|\left\{a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \not\equiv 1 \mod n\right\}\right| \geq \frac{\varphi(n)}{2}.$$

Demostración. Sean

$$A_+ = \{a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \equiv 1 \mod n\}$$
$$A_- = \{a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \not\equiv 1 \mod n\}$$

Tenemos que $a_0 \in A_-$. La aplicación

$$a_0 \cdot : \mathcal{U}(\mathbb{Z}_n) \to \mathcal{U}(\mathbb{Z}_n)$$

es una biyección que satisface

$$a_0 A_+ \subseteq A_-,$$

por lo que $|A_+| \leq |A_-|$. Como $\mathcal{U}(\mathbb{Z}_n) = A_+ \cup A_-$ siendo la unión disjunta, tenemos

$$\varphi(n) = |\mathcal{U}(\mathbb{Z}_n)| = |A_+| + |A_-| \leq 2|A_-|,$$

lo que implica $|A_-| \geq \frac{\varphi(n)}{2}$. □

Por tanto, calculando $a^{k/2}$ mód n para suficientes $a \in \mathcal{U}(\mathbb{Z}_n)$, podemos comprobar que $A_- \neq \varnothing$ o afirmar con probabilidad tan alta como queramos que $a^{k/2} \equiv 1 \mod n$ para todo $a \in \mathcal{U}(\mathbb{Z}_n)$. En este último caso reemplazamos k por $k/2$ y volvemos a empezar.
Tras los reemplazos necesarios, podemos suponer que $A_- \neq \varnothing$, es decir,

$$\forall a \in \mathcal{U}(\mathbb{Z}_n), \quad a^k \equiv 1 \mod n$$

y

$$\exists a_0 \in \mathcal{U}(\mathbb{Z}_n), \quad a_0^{k/2} \not\equiv 1 \mod n.$$

Tenemos dos posibilidades

(I) $\frac{k}{2}$ es múltiplo de $p-1$ o $q-1$ pero no de ambos, pongamos $p-1$. En este caso $a^{k/2} \equiv 1 \mod p$ para todo $a \in \mathcal{U}(\mathbb{Z}_n)$ y existe $a_0 \in \mathcal{U}(\mathbb{Z}_n)$ tal que

$$a_0^{k/2} \not\equiv 1 \mod n$$

o, equivalentemente,

$$a_0^{k/2} \equiv -1 \mod q.$$

Usando (3.1), la multiplicación

$$a_0 \cdot : \mathcal{U}(\mathbb{Z}_n) \to \mathcal{U}(\mathbb{Z}_n)$$

es una biyección que satisface

$$a_0 \left\{ a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \equiv 1 \mod q \right\} = \left\{ a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \equiv -1 \mod q \right\}.$$

En consecuencia

$$\left|\left\{a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \equiv -1 \mod q\right\}\right| = \frac{\varphi(n)}{2}.$$

(II) $\frac{k}{2}$ no es múltiplo de $p-1$ ni $q-1$. Sean

$$A_{\pm\pm} = \{a \in \mathcal{U}(\mathbb{Z}_n) \mid a^{k/2} \equiv \pm 1 \mod p, \quad a^{k/2} \equiv \pm 1 \mod q\}.$$

Como $p-1 \nmid \frac{k}{2}$, existe $a_0 \in \mathcal{U}(\mathbb{Z}_n)$ tal que

$$a_0^{k/2} \equiv -1 \mod p.$$

Reemplazando eventualmente a_0 por la solución del sistema

$$\begin{aligned} x &\equiv a_0 \mod p \\ x &\equiv 1 \mod q \end{aligned}$$

podemos suponer que

$$a_0^{k/2} \equiv 1 \mod q.$$

Análogamente, existe $a_1 \in \mathcal{U}(\mathbb{Z}_n)$ tal que

$$a_1^{k/2} \equiv -1 \mod q \quad \text{y} \quad a_1^{k/2} \equiv 1 \mod p.$$

Las aplicaciones $a_0\cdot, a_1\cdot : \mathcal{U}(\mathbb{Z}_n) \to \mathcal{U}(\mathbb{Z}_n)$ son biyectivas. Dado que $a_0 A_{++} = A_{-+}$, $a_0 A_{--} = A_{+-}$, $a_1 A_{++} = A_{+-}$ y $a_1 A_{--} = A_{-+}$, tenemos que

$$|A_{++}| = |A_{+-}| = |A_{-+}| = |A_{--}|,$$

lo que implica

$$|A_{+-} \cup A_{-+}| = \frac{\varphi(n)}{2}.$$

En cualquiera de las dos posibilidades, encontramos con probabilidad $\frac{1}{2}$ un $a \in \mathbb{Z}_n$ tal que

$$a^{k/2} \equiv 1 \mod p \quad \text{y} \quad a^{k/2} \equiv -1 \mod q$$

o

$$a^{k/2} \equiv -1 \mod p \quad \text{y} \quad a^{k/2} \equiv 1 \mod q.$$

En este caso $\left(a^{k/2} - 1, n\right)$ es un factor propio de n. En consecuencia, si fuésemos capaces de calcular d tal que

$$m^{ed} \equiv m \mod n$$

para cualquier $m \in \mathbb{Z}_n$ primo con n, podríamos factorizar n con probabilidad tan alta como quisiéramos. Por este motivo se asume que el coste de calcular $\mathrm{RSA}_{n,e}^{-1}$ es equivalente al coste de descomponer $n = pq$.

Ejemplo 3.5. Consideremos la clave pública $(n, e) = (1189, 257)$. Si $d = 353$, podemos encontrar una descomposición de n.

3.2 Descripción de RSA

Generación de claves. Para generar su pareja de claves, Alicia realiza los siguientes pasos.

(I) Selecciona aleatoriamente dos primos p_A, q_A de tamaño adecuado. Cada primo lo selecciona de la siguiente forma. Para un primo de b bits seleccionamos aleatoriamente $b-2$ bits, agregamos al principio y al final dos 1 para garantizar que tenemos un número impar de exactamente b bits. Le aplicamos un test de primalidad. Si el resultado es positivo, ya tenemos el primo, En caso contrario sumamos 2 al número obtenido y repetimos el proceso.

(II) Calcula $n_A = p_A q_A$ y $\varphi(n_A) = n_A + 1 - (p_A + q_A)$.

(III) Elige un e_A primo con $\varphi(n_A)$. Esta elección puede hacerse de muchas formas. Una de ellas es proceder de forma análoga a la elección de p_A y q_A, generando aleatoriamente un número impar y comprobando si es primo con $\varphi(n_A)$. Otra forma es elegir un primo entre $\text{máx}\{p_A, q_A\}$ y $\varphi(n_A)$, lo que garantiza que será primo con $\varphi(n_A)$. Muchos sistemas toman como $e_A = 65537 = 2^{16} + 1$ por eficiencia de las implementaciones.

(IV) Calcula $d_A = e_A^{-1}$ mód $\varphi(n_A)$.

(V) La clave pública es la pareja (n_A, e_A).

(VI) La clave privada es (p_A, q_A, d_A).

Función de cifrado. La función de cifrado es

$$\text{RSA}_{n_A, e_A}(m) = m^{e_A} \text{ mód } n_A.$$

La elección de e_A puede acelerar la función de cifrado. Por ejemplo, si $e_A = 65537$, para cifrar hay que realizar 5 multiplicaciones módulo

n_A. Consecuentemente, RSA_{n_A,e_A} está en $\mathcal{O}\left((\log n_A)^2\right)$, eliminando el factor $\log e_A$ que aparece en la Proposición 2.8. Otras elecciones como $e_A = 3, 17$, que también dan implementaciones eficientes, pueden acarrear problemas de seguridad.

Función de descifrado. La función de descifrado es

$$\mathrm{RSA}^{-1}_{n_A,e_A}(c) = c^{d_A} \bmod n_A.$$

El Teorema 3.1 garantiza que la función anterior es efectivamente la función de descifrado. Por la Proposición 2.8,

$$\mathrm{RSA}^{-1}_{n_A,e_A} = \mathcal{O}\left((\log n_A)^2(\log d_A)\right).$$

El isomorfismo Φ de (3.1), permite acelerar el proceso de descifrado, concretamente,

$$\mathrm{RSA}^{-1}_{n_A,e_A}(c) = \mathrm{RSA}_{n_A,d_A}(c) = \\ \Phi^{-1}\left(c^{d_A \bmod (p_A-1)} \bmod p_A, c^{d_A \bmod (q_A-1)} \bmod q_A\right).$$

Aunque la complejidad algorítmica no se ve reducida, en la práctica el tamaño de p_A y q_A es la mitad de n_A, y lo mismo podemos decir de $d_A \bmod p_A - 1$ y $d_A \bmod q_A - 1$ con respecto a d_A, con lo que se reduce el tiempo empleado en el descifrado.

Ejemplo 3.6. $p = 31, q = 17, n = 527, \varphi(n) = 480, e = 17, d = 113$.

Supongamos que $m = 130$.

$$\begin{aligned} c &= \mathrm{RSA}_{527,17}(130) \\ &= \left(\left((130^2)^2\right)^2\right)^2 130 \text{ mód } 527 \\ &= \left((36^2)^2\right)^2 130 \text{ mód } 527 \\ &= \left(242^2\right)^2 130 \text{ mód } 527 \\ &= 67^2 130 \text{ mód } 527 = 273 \cdot 130 \text{ mód } 527 \\ &= 181. \end{aligned}$$

Para descifrar c,

$$
\begin{aligned}
&\mathrm{RSA}_{527,17}^{-1}(c) = \mathrm{RSA}_{527,113}(181) = 181^{113} \text{ mód } 527 \\
&= \left(\left(\left(\left((181^2 181)^2 181\right)^2\right)^2\right)^2\right)^2 181 \text{ mód } 527 \\
&= \left(\left(\left(\left((87 \cdot 181)^2 181\right)^2\right)^2\right)^2\right)^2 181 \text{ mód } 527 \\
&= \left(\left(\left((280^2 181)^2\right)^2\right)^2\right)^2 181 \text{ mód } 527 \\
&= \left(\left(\left((404 \cdot 181)^2\right)^2\right)^2\right)^2 181 \text{ mód } 527 \\
&= \left(\left((398^2)^2\right)^2\right)^2 181 \\
&= \left((304^2)^2\right)^2 181 \text{ mód } 527 \\
&= (191^2)^2 181 = 118^2 181 \text{ mód } 527 \\
&= 222 \cdot 181 \text{ mód } 527 = 130.
\end{aligned}
$$

Para el descifrado acelerado,

$$\begin{aligned} c^{d \bmod p-1} \bmod p &= 181^{113 \bmod 30} \bmod 31 = 26^{23} \bmod 31 \\ &= \left(\left(\left(26^2\right)^2 26\right)^2 26\right)^2 26 \bmod 31 \\ &= \left(\left(25^2 26\right)^2 26\right)^2 26 \bmod 31 \\ &= \left((5 \cdot 26)^2 26\right)^2 26 \bmod 31 \\ &= \left(6^2 26\right)^2 26 \bmod 31 = (5 \cdot 26)^2 26 \bmod 31 \\ &= 6^2 26 \bmod 315 \cdot 26 \bmod 31 = 6. \end{aligned}$$

y

$$c^{d \bmod q-1} \bmod q = 181^{113 \bmod 16} \bmod 17 = 11^1 \bmod 17 = 11,$$

por tanto

$$m = \Phi^{-1}(6, 11) = 130.$$

3.3 Ataques

Ataque con primos muy próximos (Factorización de Fermat). Si $n = pq$ con $p > q$, entonces

$$n = \left(\tfrac{p+q}{2}\right)^2 - \left(\tfrac{p-q}{2}\right)^2.$$

Si p y q son cercanos, $s = \frac{p-q}{2}$ es pequeño y $t = \frac{p+q}{2}$ es un entero ligeramente mayor que $\sqrt{n}$ tal que $t^2 - n$ es un cuadrado perfecto.

Probando sucesivamente con valores mayores que $\sqrt{n}$ hasta encontrar una descomposición $n = t^2 - s^2$, tenemos que $p = t + s$ y $q = t - s$.

Ejemplo 3.7. Calculamos la descomposición como producto de primos de 1591. Como $\sqrt{1591} \approx 39{,}89$ empezamos con $t = 40$.

t	$t^2 - n$
40	$9 = 3^2$

Por tanto $p = 40 + 3 = 43$ y $q = 40 - 3 = 37$

Ataque del módulo común. Supongamos que tenemos dos claves públicas con el mismo módulo, (n, e_1) y (n, e_2), tales que $(e_1, e_2) = 1$. Supongamos que $re_1 + se_2 = 1$ con $r < 0 < s$. Sea

$$c_i = m^{e_i} \text{ mód } n$$

para $i = 1, 2$. Si $(c_1, n) \neq 1$, podemos factorizar n y romper la clave, por lo que podemos suponer que $c_1 \in \mathcal{U}(\mathbb{Z}_n)$. Calculamos c_1^{-1} con el algoritmo extendido de Euclides. Tenemos

$$(c_1^{-1})^{-r} c_2^s \equiv (m^{e_1})^r (m^{e_2})^s \equiv m^{e_1 r + e_2 s} = m \quad \text{mód } n,$$

por lo que averiguamos el mensaje.
Por otra parte, si las claves públicas con el mismo módulo se corresponden con dos usuarios distintos, cada uno de ellos puede calcular la clave privada del otro al conocer la factorización del módulo.

Ejemplo 3.8. $n = 1537$, $e_1 = 17$, $e_2 = 37$, $c_1 = 1298$, $c_2 = 614$. Calculamos r, s,

$$1 = re_1 + se_2 = (-13)17 + 6 \cdot 37.$$

Así,

$$\begin{aligned}(c_1^{-1})^{-r}c_2^s &= (1298^{-1})^{13}(614)^6 \equiv 1164^{13}614^6 \mod 1537\\ &\equiv 341\cdot 661 \mod 1537 \equiv 999 \mod 1537\end{aligned}$$

Ataque del exponente de cifrado bajo. Enviamos el mismo mensaje m a varios recipientes, con claves públicas (n_i, e), con $1 \le i \le r$. Supondremos que $(n_i, n_j) = 1$ si $i \neq j$, pues en otro caso podríamos factorizar el módulo correspondiente mediante el cálculo del máximo común divisor. Llamamos $c_i = m^e \bmod n_i$ para cada $1 \le i \le r$. Supongamos que $e \le r$. Seleccionamos $\{i_1, \ldots, i_e\} \subseteq \{1, \ldots, r\}$ Empleando el inverso del isomorfismo de anillos

$$\Phi : \mathbb{Z}_{n_{i_1}\cdots n_{i_e}} \to \mathbb{Z}_{n_{i_1}} \times \cdots \times \mathbb{Z}_{n_{i_e}}$$

dado por el Teorema Chino del Resto, podemos calcular

$$\Phi^{-1}(c_{i_1}, \ldots, c_{i_e}) = m^e \bmod n_{i_1}\cdots n_{i_e},$$

Dado que $m^e < n_{i_1}\cdots n_{i_e}$, podemos calcular m calculando la raíz e–ésima en $\mathbb{Z} \subseteq \mathbb{R}$, que se puede hacer eficientemente mediante métodos numéricos, siempre que e sea pequeño.

Ejemplo 3.9. Supongamos que tenemos las claves públicas $(n_1, e) = (731, 3)$, $(n_2, e) = (943, 3)$ y $(n_3, e) = (611, 3)$. Ciframos un mismo mensaje y obtenemos los criptogramas $c_1 = 505$, $c_2 = 876$ y $c_3 = 372$. La solución del sistema de congruencias

$$\begin{aligned}x &\equiv 505 \mod 731\\ x &\equiv 876 \mod 943\\ x &\equiv 372 \mod 611\end{aligned}$$

es $x = 124251499$, cuya raíz cúbica es $m = 499$.

Ataque del criptograma elegido. Supongamos que Bob tiene como clave pública la pareja (n, e). Se cifra un mensaje m como $c = \mathrm{RSA}_{n,e}(m) = m^e \bmod n$. La atacante Eve[1] elige aleatoriamente $r \in \mathcal{U}(\mathbb{Z}_n)$ y solicita el descifrado del mensaje

$$\mathrm{RSA}_{n,e}^{-1}(r^e c \bmod n) = rm \bmod n.$$

El aspecto de $r^e c \bmod n$ es aleatorio y no parece tener que ver con c. Finalmente,

$$m \equiv r^{-1}(rm) \mod n,$$

por lo que se recupera el mensaje.

Ejemplo 3.10. Ciframos un mensaje con la clave pública $(n, e) = (989, 17)$, obteniendo el criptograma $c = 256$. Aleatoriamente $r = 357$, con lo que $r^e c \bmod n = 855$. El descifrado que pedimos es

$$\mathrm{RSA}_{989,17}^{-1}(855) = 315.$$

Finalmente, $m = 315 * 357^{-1} \bmod 989 = 699$.

Método de factorización $P-1$ de Pollard. Sea $n = pq$. Supongamos que existe b tal que todos los divisores potencia de primos de $p-1$ son menores que b,[2] pero no todos los divisores potencia de primos primos

[1] Suele emplearse este nombre por la similitud con *eavesdrop*, palabra inglesa cuyo significado es escuchar a escondidas.

[2] Se dice que $p-1$ es b-potencia suave

de $q-1$ son menores que b. Podemos esperar por tanto que $p-1 \mid b!$, pero $q-1 \nmid b!$. Supongamos que podemos calcular

$$a = 2^{b!} \text{ mód } n.$$

Como $p-1 \mid b!$, tenemos que $a \equiv 1 \mod p$. Por otra parte, dado que $q-1 \nmid b!$, $a \not\equiv 1 \mod q$. Por tanto $p \mid a-1$ pero $q \nmid a-1$. Podemos encontrar $p = (n, a-1)$.
Se puede demostrar que este método tiene complejidad

$$\mathcal{O}\left(b(\log b)(\log n)^2 + (\log n)^3\right).$$

Ejemplo 3.11. Sea $n = 1457$. Como no conocemos b, vamos a probar con varios. Empezamos con $b = 3$. Tenemos que

$$2^{3!} = 64 \text{ mód } 1457, (64-1, 1457) = 1.$$

Para $b = 4$,

$$2^{4!} = 64^4 \equiv 1318 \mod 1457, (1318-1, 1457) = 1.$$

Para $b = 5$,

$$2^{5!} = 1318^5 \equiv 32 \mod 1457, (32-1, 1457) = 31.$$

Ejercicios de RSA

Ejercicio 3.1. Si la clave pública de un criptosistema RSA es $(2291, 17)$,

1. calcula $2291 = pq$ sabiendo que $\varphi(n) = 2184$,
2. calcula $d = e^{-1}$ mód 2184,
3. calcula $2291 = pq$ sabiendo que $d = 257$,
4. calcula $\mathrm{RSA}_{2291,17}(1116)$,
5. descifra el mensaje anterior empleando el Teorema Chino del Resto,
6. descifra el mensaje anterior sin emplear el Teorema Chino del Resto.

Ejercicio 3.2. Comprueba que existe una biyección entre divisores de n mayores que $\sqrt{n}$ y descomposiciones de n como diferencia de dos cuadrados.

Ejercicio 3.3. Intenta romper el criptosistema RSA con clave pública $(n, e) = (536813567, 3602561)$.

Ejercicio 3.4. Factoriza 23360947609 mediante la factorización de Fermat.

Ejercicio 3.5. Factoriza 332483, 279533 mediante el algoritmo $P - 1$ de Pollard.

Ejercicios de evaluación del Criptosistema RSA

Ejercicio. Este ejercicio es individualizado. Cada uno parte del número de su DNI, supongamos por ejemplo 12340987. Dividimos dicho número en dos bloques, 1234 y 0987. Si alguno de ellos es menor que 1000, rotamos las cifras a la izquierda hasta obtener un número mayor o igual que 1000, en nuestro caso 9870 (si alguien tuviese como un bloque el 0000, que coja un número mayor que 1000 cualquiera a su elección. Sean p y q los primeros primos mayores o iguales que los bloques anteriores. Concretamente, en el ejemplo $p = 1237$ y $q = 9871$. Sea $n = pq$ y e el menor primo mayor o igual que 11 que es primo relativo con $\varphi(n)$. Sea $d = e^{-1}$ mód $\varphi(n)$.

1. Cifra el mensaje $m = 0xCAFE$ (recordad que $0x$ indica que el número está escrito en hexadecimal).

2. Descifra el criptograma anterior.

3. Intenta factorizar n mediante el método $P - 1$ de Pollard. Para ello llega, como máximo a $b = 8$.

4. Intenta factorizar n a partir de $\varphi(n)$.

5. Intenta factorizar n a partir de e y d.

Capítulo 4

Criptosistemas basados en el logaritmo discreto

4.1 Problema del logaritmo discreto

Sea G un grupo y $b \in G$ un elemento de orden n. Dado $h \in \langle b \rangle$, el problema del logaritmo discreto consiste en encontrar $0 \leq m \leq n-1$ tal que $h = b^m$. En este caso decimos

$$m = \log_b(h) = \mathrm{idx}_b(h).$$

Si empleamos notación aditiva para la operación en G, el problema se traduce en encontrar $0 \leq m \leq n-1$ tal que $h = mb$.
La Proposición 2.8 establece el coste de calcular potencias en el grupo multiplicativo $\mathcal{U}(\mathbb{Z}_n)$.

Proposición 4.1. *El tiempo empleado en calcular* g^m *es*

$$\mathcal{O}\left(M(G)(\log m)\right),$$

donde $M(G)$ *es el tiempo empleado en realizar la multiplicación en* G.

Demostración. Análogo a la demostración de la Proposición 2.8. □

Por tanto, si hay una forma eficiente de calcular la operación en G, podremos eficientemente calcular potencias. Sin embargo no se conocen algoritmos eficientes para calcular logaritmos discretos en general. Presentamos algunos de los clásicos, aunque ninguno tiene complejidad polinomial.

4.1.1 *Paso de bebé – Paso de gigante.*

Este algoritmo se conoce a veces como algoritmo de Shanks, en honor a Daniel Shanks, el primero que lo publicó, aunque algunos autores aseguran que era conocido previamente.

Teorema 4.2. *El Algoritmo 1 calcula el logaritmo discreto, si existe.*

Demostración. Observemos que $h_i = hb^{-if}$. Por tanto, si $h_i = b^j$, tenemos que

$$h = hb^{-if}b^{if} = b^j b^{if} = b^{j+if},$$

de donde $j + if = \log_b(h)$. Por tanto, el algoritmo da la salida correcta si existe el logaritmo discreto.
Por otra parte, si $h = b^m$, dividimos m entre f, obteniendo $m = if + j$ donde $0 \leq j < f$. Dado que $m < n$, necesariamente $i < f$, por lo que

$$h_i = hb^{-if} = b^m b^{-if} = b^{if+j}b^{-if} = b^j,$$

es decir, si existe el logaritmo discreto el algoritmo debe encontrarlo. □

Algorithm 1 Baby step – Giant step

Input: $b \in G$ de orden n, y $h \in \langle b \rangle$
Output: $\log_b h$

1: $f \leftarrow \lceil \sqrt{n} \rceil$
2: $\mathtt{table} \leftarrow [\,]$
3: **for** $0 \le i \le f-1$ **do**
4: $\quad \mathtt{table} \leftarrow \mathtt{table} + \left[(i, b^i)\right]$
{La tabla construida es

$$\mathtt{table} = \left[\begin{array}{cccc} 0 & 1 & \dots & f-1 \\ \hline b^0 & b^1 & \dots & b^{f-1} \end{array}\right]$$

}
5: Calcula $b^{-f} = b^{n-f}$
6: $h_0 \leftarrow h$
7: **for** $0 \le i \le f-1$ **do**
8: $\quad$ **if** $\exists (j, h_i) \in \mathtt{table}$ **then**
9: $\quad\quad$ **return** $j + if$
10: $\quad$ **else**
11: $\quad\quad$ $h_{i+1} \leftarrow h_i b^{-f}$
12: **return** Error, no existe el logaritmo.

Este algoritmo nos fuerza a realizar $f = \lceil \sqrt{n} \rceil$ multiplicaciones para calcular $\mathtt{table}$, $3\log_2(n-f)$ multiplicaciones para calcular b^{-f}, y un máximo de f multiplicaciones para calcular los h_i. Además hay que realizar en media $\frac{f}{2}$ comparaciones, u ordenar inicialmente $\mathtt{table}$. Esto hace que este algoritmo sea impracticable para valores grandes de n.

Ejemplo 4.3. Sea $p = 251$ y $b = 6$, un elemento primitivo módulo p. Vamos a calcular $\log_6(20)$ empleando el algoritmo de Shanks. Para ello,

$$f = \left\lceil \sqrt{p-1} \right\rceil = 16.$$

Calculamos $\mathtt{table}$, y obtenemos

0	1	2	3	4	5	6	7
1	6	36	216	41	246	221	71

8	9	10	11	12	13	14	15
175	46	25	150	147	129	21	126

El siguiente paso consiste en calcular

$$b^{-f} \equiv b^{p-1-f} = 6^{234} \equiv 84 \mod 251.$$

Empezando por $h_0 = 20$, vamos calculando parejas (i, h_i) donde $h_i = h_{i-1}b^{-f}$ mód p hasta encontrar una en la que $h_i \in \mathtt{table}$. Las parejas calculadas son las siguientes

$$(0,20), (1,174), (2,58), (3,103), (4,118), (5,123), (6,41)$$

y como $(4,41) \in \mathtt{table}$, tenemos que $\log_6(20) = j + if = 4 + 6\times 16 = 100$.

Algorithm 2 Silver-Pohlig-Hellman

Input: $b \in G$ de orden n, y $h \in \langle b \rangle$, $n = \prod_{i=1}^{r} p_i^{e_i}$
Output: $\log_b h$

1: **for** $1 \leq i \leq r$ **do**
2: **for** $0 \leq j \leq p_i - 1$ **do**
3: $r_{i,j} \leftarrow b^{jn/p_i}$
{Estos valores son las raíces p_i-ésimas de la unidad en $\langle b \rangle$}
4: **for** $0 \leq k \leq e_i - 1$ **do**
5: $y_k \leftarrow h/b^{x_0 + x_1 p_i + \cdots + x_{k-1} p_i^{k-1}}$
6: $x_k \leftarrow j$ tal que $y_k^{n/p_i^{k+1}} = r_{i,j}$
7: $m_i \leftarrow x_0 + x_1 p_i + \cdots + x_{e_i - 1} p_i^{e_i - 1}$
8: **return** m tal que $m \equiv m_i \mod p_i^{e_i}, 1 \leq i \leq r$

4.1.2 *El algoritmo de Silver-Pohlig-Hellman*

Teorema 4.4. *El Algoritmo 2 calcula el logaritmo discreto.*

Demostración. Supongamos que $m = \log_b h$. Dado que $n = \prod_{i=1}^{r} p_i^{e_i}$, por el Teorema Chino del Resto es suficiente con calcular $m_i = m \mod p_i^{e_i}, 1 \leq i \leq r$ para conocer m. Para simplificar la notación, sean $i \in \{1, \ldots, r\}$, $p = p_i$ y $e = e_i$. Sea, ademas $r_j = b^{jn/p}$ para $0 \leq j \leq p - 1$. Dado que $b^n = 1$, tenemos que $\{r_0, \ldots, r_{p-1}\}$ son todas las raíces p-ésimas de la unidad. Supongamos que

$$m \equiv x_0 + x_1 p + \cdots + x_{e-1} p^{e-1} \mod p^e$$

con $0 \leq x_k < p$. Como $h = b^m$, tenemos que

$$h^{n/p} = b^{mn/p} = b^{x_0 n/p} = r_{x_0},$$

por tanto calculamos x_0 encontrando el valor de $0 \leq j \leq p-1$ tal que $h^{n/p} = r_j$. Supongamos que hemos calculado $x_0, \ldots, x_{k-1}$ y sea

$$y_k = h/b^{x_0 + x_1 p + \cdots + x_{k-1} p^{k-1}}.$$

Como

$$\begin{aligned} y_k &= b^{m-(x_0 + x_1 p + \cdots + x_{k-1} p^{k-1})} \\ &= b^{x_k p^k + \cdots + x_{e-1} p^{e-1} + \alpha p^e}, \end{aligned}$$

tenemos que

$$\begin{aligned} y_k^{n/p^{k+1}} &= b^{(x_k p^k + \cdots + x_{e-1} p^{e-1} + \alpha p^e) n/p^{k+1}} \\ &= b^{(x_k + \cdots + x_{e-1} p^{e-1-k} + \alpha p^{e-k}) n/p} \\ &= b^{x_k n/p} b^{(x_{k+1} + \cdots + x_{e-1} p^{e-1-k-1} + \alpha p^{e-k-1}) n} \\ &= b^{x_k n/p} \\ &= r_{x_k}, \end{aligned}$$

por tanto x_k es el valor $0 \leq j \leq p-1$ tal que $y_k^{n/p^{k+1}} = r_j$. Así calculamos $x_0, \ldots, x_{e-1}$, lo que termina el teorema. □

Este algoritmo sólo en útil si podemos factorizar n, lo cual es posible si los primos que aparecen en la factorización de n son pequeños. Además, si uno de los factores de n es grande, el número de raíces a precalcular hace también inaplicable el Algoritmo 2.

Ejemplo 4.5. Sea $p = 397$ y $G = \mathbb{Z}_p^*$ un grupo cíclico de orden $p-1 = 396 = 2^2 \cdot 3^2 \cdot 11$. Un generador de G es $b = 5$. Vamos a calcular $\log_5(337)$.

La primera vuelta del algoritmo trabaja con $p_1 = 2$ y $e_1 = 2$. Tenemos que

$$r_{1,0} = 1, r_{1,1} = 396,$$

y la sucesión de coeficientes

$$y_0 = 337, x_0 = 1, y_1 = 385, x_1 = 1,$$

por lo que $m_1 = 1 + 1 \cdot 2 = 3$.
En la segunda vuelta, $p_2 = 3$, $e_2 = 2$,

$$r_{2,0} = 1, r_{2,1} = 362, r_{2,2} = 34,$$

y

$$y_0 = 337, x_0 = 0, y_1 = 337, x_1 = 1,$$

obteniendo $m_2 = 0 + 1 \cdot 3 = 3$.
Finalmente, en la tercera vuelta $p_3 = 11$ y $e_3 = 1$. La raíces son

$$r_{3,0} = 1, r_{3,1} = 290, r_{3,2} = 333, r_{3,3} = 99,$$
$$r_{3,4} = 126, r_{3,5} = 16, r_{3,6} = 273, r_{3,7} = 167,$$
$$r_{3,8} = 393, r_{3,9} = 31, r_{3,10} = 256$$

y el coeficiente

$$y_0 = 337, x_0 = 9,$$

de donde $m_3 = 9$.
Finalmente, el logaritmo buscado es la solución al sistema de ecuaciones

$$\begin{aligned} m &\equiv 3 \mod 4 \\ m &\equiv 3 \mod 9 \\ m &\equiv 9 \mod 11, \end{aligned}$$

es decir, $m = 75$.

4.1.3 *Cálculo de índices en cuerpos finitos*

Sea $\mathbb{F}_q$ $q = p^n$, y sea $b \in \mathbb{F}_q^*$ un elemento primitivo. Supongamos que $\mathbb{F}_q = \mathbb{F}_p[\alpha]_{f(\alpha)}$ con f un polinomio irreducible de grado n. Identificamos los elementos de $\mathbb{F}_q$ con polinomios en α sobre $\mathbb{F}_p$ de grado menor que n. Sea $b' = b^{(q-1)/(p-1)}$. Tenemos que

$$(b')^p = (b^{(q-1)/(p-1)})^p = b^{\frac{p(q-1)}{(p-1)}} = b^{q-1} b^{(q-1)/(p-1)} = b',$$

por lo que $b' \in \mathbb{F}_p$ por ser un elemento invariante ante el automorfismo de Frobenius. De forma análoga,

$$(b')^{p-1} = b^{q-1} = 1$$

y si $(b')^r = 1$ para un divisor propio $r \mid p-1$, obtenemos que

$$b^{r(q-1)/(p-1)} = 1$$

con $\frac{r(q-1)}{p-1} \mid q-1$ un divisor propio, lo que contradice el que b es primitivo. Hemos demostrado que

$b' = b^{(q-1)/(p-1)}$ es un elemento primitivo de $\mathbb{F}_p$.

Como consecuencia, si podemos calcular logaritmos discretos en $\mathbb{F}_p$ con respecto de b', podemos calcular logaritmos discretos de elementos en $\mathbb{F}_p \subseteq \mathbb{F}_q$ con respecto de b.
Sea $B \subseteq \mathbb{F}_q^*$ el conjunto formado por todos los polinomios mónicos en $\mathbb{F}_p[\alpha]$ de grado menor que $m < n$. Se elige m de forma que $|B|$ tiene un tamaño intermedio entre p y q.

Elegimos aleatoriamente $1 \leq t \leq q-2$ y calculamos b^t, o equivalentemente,

$$c(\alpha) = b(\alpha)^t \text{ mód } f(\alpha).$$

Comprobamos si

$$c(\alpha) = c_0 \prod_{a(\alpha)\in B} a(\alpha)^{m_{c,a}}$$

Esto puede hacerse dividiendo los polinomios de B entre $c(\alpha)$ o factorizando $c(\alpha)$ y observando si los factores están en B. Si $c(\alpha)$ no es de esta forma repetimos el proceso para un nuevo t. Llegado el caso,

$$\log_b(c) - \log_b(c_0) \equiv \sum_{a\in B} m_{c,a} \log_b(a) \quad \text{mód } q-1.$$

Puesto que $\log_b(c) = t$, $\log_b(c_0)$ se puede calcular y los valores $m_{c,a}$ son conocidos, tenemos una ecuación lineal módulo $q-1$ cuyas incógnitas son $\log_b(a)$, $a \in B$. Repetimos el proceso tantas veces como necesitemos para obtener suficientes ecuaciones independientes en $\mathbb{Z}_{q-1}$ que nos permitan calcular $\log_b(a)$, $a \in B$. Con esto concluye la fase de precálculo.

Observemos que si m es demasiado pequeño será difícil encontrar valores t tales que $c = b^t$ descomponga como producto de elementos en B. Si m es demasiado grande, el tamaño de B será prohibitivamente grande para manejar los sistemas de ecuaciones. Sirva como ejemplo, que para $p = 2$ y $n = 127$, suele tomarse un valor $m = 17$, lo que nos lleva a $|B| = 16510$.

Para calcular $\log_b(y)$, se eligen aleatoriamente valores $1 \leq t \leq q-2$,

hasta que

$$y_1(\alpha) = y_0 \prod_{a \in B} a(\alpha)^{m_\alpha}$$

donde $y_1 = yb^t$. Llegados a este punto,

$$\begin{aligned} \log_b(y) &= \log_b(y_1) - t \\ &= \log_b(y_0) + \sum_{a \in B} m_\alpha \log_b(a) - t \mod q - 1, \end{aligned}$$

por lo que calculamos el logaritmo.

4.1.4 *Cálculo de índices en cuerpos primos*

Sea $\mathbb{F}_p = \mathbb{Z}_p$ con p primo. Sea b un elemento primitivo módulo p. Sea $2 \leq B < p$. La fase de precálculo vuelve a consistir en calcular $\log_b r$ para todo primo $r \leq B$. Para ello, se elige aleatoriamente $1 \leq t \leq p-2$ hasta que b^t mód p es B-suave, es decir,

$$b^t \equiv \prod_{\substack{r \leq B \\ r \text{ primo}}} r^{m_{t,r}} \mod p.$$

En consecuencia

$$t \equiv \sum_{\substack{r \leq B \\ r \text{ primo}}} m_{t,r} \log_b(r) \mod p - 1.$$

Repitiendo suficientes elecciones de t, obtendremos un sistema de ecuaciones lineales sobre $\mathbb{Z}_{p-1}$ cuya solución son los valores buscados

$$\{\log_b(r) \mid r \leq B, r \text{ primo}\}.$$

Para calcular $\log_b(y)$ con $y \in \mathbb{Z}_p^*$, volvemos a elegir valores aleatorios $1 \leq t \leq p-2$ hasta encontrar uno que satisfaga

$$yb^t \equiv \prod_{\substack{r \leq B \\ r \text{ primo}}} r^{m_r} \quad \text{mód } p.$$

Una vez encontrado,

$$\log_b(y) \equiv -t + \sum_{\substack{r \leq B \\ r \text{ primo}}} m_r \log_b(r) \quad \text{mód } p-1.$$

La usabilidad de este algoritmo requiere que p no sea demasiado grande para que haya suficientes elementos B-suaves en $\mathbb{Z}_p$.

Ejemplo 4.6. Repetimos el cálculo del Ejemplo 4.5 mediante el cálculo de índices. Recordemos que $p = 397$ y $b = 5$. Tomamos $B = 5$. Tenemos, por tanto, que calcular $\log_5(2), \log_5(3), \log_5(5)$. Elegimos valores aleatorios $1 \leq t \leq 395$ tales que

$$5^t \equiv 2^{m_{t,2}} 3^{m_{t,3}} 5^{m_{t,5}} \quad \text{mód } 397.$$

Para $t = 212$, $5^t \equiv 90 = 2 \cdot 3^2 \cdot 5$ mód 397, lo que nos da una primera ecuación

$$212 \equiv 1 \log_5(2) + 2 \log_5(3) + 1 \log_5(5) \quad \text{mód } 396.$$

Para $t = 360$, $5^t \equiv 256 = 2^8$ mód 397, lo que nos da la ecuación

$$360 \equiv 8 \log_5(2) + 0 \log_5(3) + 0 \log_5(5) \quad \text{mód } 396.$$

Para $t = 366$, $5^t \equiv 225 = 3^2 \cdot 5^2$ mód 397, lo que nos da la ecuación

$$366 \equiv 0 \log_5(2) + 2 \log_5(3) + 2 \log_5(5) \quad \text{mód } 396.$$

Aunque tenemos tres ecuaciones, el determinante de los coeficientes, -16, no es una unidad módulo $p-1=396$, por lo que debemos continuar. Para $t=391$, $5^t \equiv 288 = 2^5 \cdot 3^2 \mod 397$, lo que nos da la cuarta ecuación

$$391 \equiv 5\log_5(2) + 2\log_5(3) + 0\log_5(5) \mod 396.$$

Para $t=150$, $5^t \equiv 27 = 3^3 \mod 397$, lo que nos da la ecuación

$$150 \equiv 0\log_5(2) + 3\log_5(3) + 0\log_5(5) \mod 396.$$

Para $t=246$, $5^t \equiv 250 = 2\cdot 5^3 \mod 397$, lo que nos da la ecuación

$$246 \equiv 1\log_5(2) + 0\log_5(3) + 3\log_5(5) \mod 396.$$

Para $t=151$, $5^t \equiv 135 = 3^3\cdot 5 \mod 397$, lo que nos da la ecuación

$$151 \equiv 0\log_5(2) + 3\log_5(3) + 1\log_5(5) \mod 396.$$

Llegados a este punto, comprobamos que

$$\begin{aligned} 391 &\equiv 5\log_5(2) + 2\log_5(3) + 0\log_5(5) \mod 396 \\ 246 &\equiv 1\log_5(2) + 0\log_5(3) + 3\log_5(5) \mod 396 \\ 151 &\equiv 0\log_5(2) + 3\log_5(3) + 1\log_5(5) \mod 396 \end{aligned}$$

tiene solución única

$$\log_5(2) = 243, \log_5(3) = 182, \log_5(5) = 1.$$

Para calcular $\log_5(337)$, buscamos $1 \le t \le 395$ tal que

$$337 \cdot 5^t \equiv 2^{m_2}3^{m_3}5^{m_5} \mod 397.$$

Por ejemplo, $t = 260$ satisface

$$337 \cdot 5^{260} \equiv 200 = 2^3 \cdot 5^2 \equiv \quad \text{mód } 397,$$

de donde

$$\begin{aligned} \log_5(337) &= -260 + 3\log_5(2) + 0\log_5(3) + 2\log_5(5) \\ &= 471 \equiv 75 \quad \text{mód } 396. \end{aligned}$$

4.2 Protocolo de Diffie-Hellman

En esta sección $G = \mathcal{U}(\mathbb{Z}_p) = \mathbb{Z}_p \setminus \{0\}$ con p primo, y g es un generador de G.

Conjetura (Diffie y Hellman)**.** *Dados* g^a mód p *y* g^b mód p*, calcular* g^{ab} mód p *es computacionalmente equivalente a calcular uno de los logaritmos* $a = \log_g g^a$ mód p *o* $b = \log_g g^b$ mód p.

Basado en esta conjetura, el protocolo de Diffie y Hellman es un protocolo de intercambio de claves para ser empleado en un criptosistema simétrico.

Parámetros. Se elige un primo p tal que el problema del logaritmo discreto sea difícil en $\mathcal{U}(\mathbb{Z}_p)$. Como mínimo necesitamos que p sea grande para que el Algoritmo 1 no sea efectivo, y que $p - 1$ tenga factores grandes para que sea ineficaz el Algoritmo 2.
El siguiente paso es la elección de un elemento primitivo de $\mathbb{Z}_p$, es decir, un generador del grupo multiplicativo $\mathcal{U}(\mathbb{Z}_P) = \mathbb{Z}_p \setminus \{0\}$.

Lema 4.7. $g \in \mathbb{Z}_p \setminus \{0\}$ *es primitivo si y solo si*

$$g^{(p-1)/p_i} \not\equiv 1 \mod p$$

para cada factor primo p_i *de* $p-1$.

Demostración. Por el Teorema de Lagrange, si r es el menor exponente positivo tal que $g^r \equiv 1 \mod p$, entonces $r \mid p-1$. Por el Teorema Fundamental de la Aritmética, si $r \neq p-1$ entonces $r \mid \frac{p-1}{p_i}$ para algún divisor primo $p_i \mid p-1$, lo que implica que $g^{(p-1)/p_i} \equiv 1 \mod p$. □

Este lema nos proporciona un método eficiente para comprobar si un elemento es primitivo. Para encontrar un elemento primitivo vamos a seleccionar aleatoriamente elementos de $\mathbb{Z}_p \setminus \{0\}$ hasta encontrar uno.

Lema 4.8. *Hay* $\varphi(p-1)$ *elementos primitivos en* $\mathbb{Z}_p \setminus \{0\}$. *Sea* $\rho = \varphi(p-1)/(p-1)$. *La probabilidad de encontrar un elemento primitivo después de* ℓ *intentos es* $1-(1-\rho)^\ell$.

Demostración. Como $\mathcal{U}(\mathbb{Z}_p)$ es cíclico, hay al menos un elemento primitivo g, que tiene orden $p-1$. Cualquier potencia de g cuyo exponente sea primo con $p-1$ es de nuevo un elemento primitivo, y hay $\varphi(p-1)$ tales exponentes. Por otra parte, una potencia con exponente no primo con $p-1$ no puede ser elemento primitivo.
La segunda parte del Lema es inmediata dado que los procesos de elección son independientes. □

En [2, Theorem 328] se demuestra que $\varphi(p-1)/(p-1)$ está asintóticamente acotado por abajo por un múltiplo constante de $\log\log(p-1)$.

Si $p-1$ tiene un divisor primo $r > \sqrt{p}$, tenemos que

$$\begin{aligned}\varphi(p-1) &= \varphi\left(\frac{p-1}{r}\right)\varphi(r)\\ &= \varphi\left(\frac{p-1}{r}\right)(r-1)\\ &\geq r-1,\end{aligned}$$

luego

$$\frac{\varphi(p-1)}{p-1} \geq \frac{r-1}{p-1} \geq \frac{\sqrt{p}-1}{p-1} \approx \frac{1}{\sqrt{p}},$$

lo que permite encontrar con rapidez un elemento primitivo.

Generación de claves pública/privada. Una vez establecida la pareja de parámetros p, g, un usuario elige aleatoriamente un valor $2 \leq x \leq p-2$, y hace público el valor g^x mód p, manteniendo x en secreto.

Clave compartida Supongamos que Alicia tiene por pareja de claves $(a, g^a \text{ mód } p)$ y Bruno $(b, g^b \text{ mód } p)$. Ambos calculan

$$g^{ab} \text{ mód } p = (g^a)^b \text{ mód } p = (g^b)^a \text{ mód } p,$$

que será su clave compartida. Si la Conjetura de Diffie y Hellman es cierta, un atacante debe calcular un logaritmo discreto en $\mathcal{U}(\mathbb{Z}_p)$ si quiere conocer g^{ab} mód p a partir de g^a mód p y g^b mód p.

4.3

Criptosistema de ElGamal

Este criptosistema emplea el logaritmo discreto como función unidireccional. Sea $\mathbb{F}_q$ un cuerpo con q elementos. Recordemos que $\mathcal{U}(\mathbb{F}_q) = \mathbb{F}_q \setminus \{0\}$ es un grupo cíclico de orden $q-1$.

Generación de claves. El usuario Alicia toma aleatoriamente $1 < a < q-1$, calcula $g^a \in \mathbb{F}_q$ y hace público $(\mathbb{F}_q, g, g^a)$, manteniendo a como clave privada.

Algoritmo de cifrado. El mensaje es $m \in \mathbb{F}_q \setminus \{0\}$, y el algoritmo

$$E_{g^a}(m) = (g^k, m(g^a)^k)$$

para un valor aleatorio $1 < k < q-1$.

Algoritmo de descifrado. El descifrado es el siguiente,

$$D_a(x, y) = yx^{-a}.$$

Teorema 4.9. $D_a(E_{g^a}(m)) = m$.

Demostración. Si $1 < k < q-1$,

$$D_a(E_{g^a}(m)) = D_a(g^k, m(g^a)^k) = m(g^a)^k(g^k)^{-a} = m.$$

como queríamos. □

Como conocemos el orden del grupo $\mathcal{U}(\mathbb{F}_q)$, la igualdad

$$x^{-a} = x^{q-1-a}$$

nos permite calcular x^{-a} como potencia sin necesidad de calcular inversos. Lo usual es que $q = p$ primo, por lo que $\mathbb{F}_q = \mathbb{Z}_p$. Las claves y algoritmos pueden reescribirse como

$$(p, g, a, g^a \text{ mód } p),$$
$$E_{g^a}(m) = (g^k \text{ mód } p, m(g^a)^k \text{ mód } p),$$
$$D_a(x, y) = yx^{p-1-a} \text{ mód } p.$$

Este cifrado es aleatorio, es decir, si ciframos dos veces el mismo mensaje con la misma clave obtenemos criptogramas distintos. Para descifrar no es necesario conocer el valor de k. El criptograma contiene el mensaje enmascarado mg^{ak} junto con un desenmascarador g^k.
Para averiguar m a partir del criptograma es necesario calcular g^{ak}, y un atacante conoce g^a, la clave pública, y g^k, la primera parte del criptograma. Según la conjetura de Diffie y Hellman, el atacante debe, por tanto, resolver un logaritmo discreto.

4.4 Digital Signature Algorithm

Hay otra construcción criptográfica basada en el concepto de clave pública, el concepto de firma digital. Dada una pareja de claves (K, k) pública privada, un esquema de firma digital consta de un algoritmo de firma dependiente de la clave privada

$$\text{sgn}_k : \mathcal{M} \to \mathcal{F}$$

y un algoritmo de verificación dependiente de la clave pública

$$\mathrm{vfy}_K : \mathcal{M} \times \mathcal{F} \to \mathbb{B}$$

tales que

- $\mathrm{vfy}_K(m, \mathrm{sgn}_k(m)) = 1$
- Encontrar $(m, f) \in \mathcal{M} \times \mathcal{F}$ tales que $\mathrm{vfy}_K(m, f) = 1$ es computacionalmente equivalente a encontrar k.

Ejemplo 4.10. Sean (n, e) y (p, q, d) las claves pública y privada de un criptosistema RSA. El siguiente es un esquema de firma digital:

$$\mathrm{sgn}_{(p,q,d)}(m) = m^d \text{ mód } n$$
$$\mathrm{vfy}_{(n,e)}(m, f) = \begin{cases} 1 & \text{si } m \equiv f^e \mod n, \\ 0 & \text{en otro caso.} \end{cases}$$

El DSA (Digital Signature Algorithm) fue propuesto por el NIST en 1991, y adoptado como DSS (Digital Signature Standard) en 1993. Desde entonces se han propuesto cuatro revisiones del mismo, la última en 2013 [4].

Generación de claves

- Se elige un primo q de N bits.
- Se elige un primo p de L bits tal que $p = 2kq + 1$. Las combinaciones N, L permitidas son

L	1024	2048	2048	3072
N	160	224	256	256

- Elegimos $h \in \mathbb{Z}_p^*$ tal que $h^{(p-1)/q} \not\equiv 1 \mod p$. Llamamos $g = h^{(p-1)/q} \mod p$.
- Elegimos $1 \leq x \leq q-1$ aleatoriamente. Calculamos $y = g^x \mod p$.
- La clave privada (usada para firmar) es (p, q, g, x).
- La clave pública (usada para verificar) es (p, q, g, y).

Proceso de firma

- El mensaje es $m \in \mathbb{Z}_q$. Se obtiene como el resultado de aplicar una función hash a un mensaje mayor.
- Seleccionamos aleatoriamente $1 \leq k \leq q$.
- Calculamos $r = (g^k \mod p) \mod q$ y $s = k^{-1}(m + rx) \mod q$. Si $s = 0$ volvemos a seleccionar k, pero es muy improbable que ocurra.
- El mensaje firmado es (m, r, s).

Verificación

- El receptor conoce el mensaje firmado (m, r, s) y la clave pública (p, q, g, y).
- Calculamos $t = s^{-1} \mod q$ y $v = ((g^m y^r)^t \mod p) \mod q$.
- La verificación es afirmativa si y sólo si $v = r$.

Proposición 4.11. *Si la firma es correcta,* $v = r$.

Demostración.

$$\begin{aligned} v &= ((g^m y^r)^t \bmod p) \bmod q \\ &= (g^{mt} g^{rxt} \bmod p) \bmod q \\ &= (g^{(m+rx)t \bmod q} \bmod p) \bmod q \\ &= (g^k \bmod p) \bmod q \\ &= r \end{aligned}$$

donde los exponentes se calculan módulo q. □

Seguridad de DSA

- Un cálculo eficiente de logaritmos discretos permite calcular la clave privada a partir de la clave pública, y por tanto generar todas las firmas deseadas.
- Dados m, r, encontrar s tal que la terna (m, r, s) es una firma correcta es equivalente a resolver la ecuación

  $$r^s \equiv (g^m y^r) \bmod p,$$

  que es un logaritmo discreto.
- Dado m, para encontrar una firma correcta m, r, s podemos, de nuevo, tratar de resolver la ecuación

  $$r^s \equiv (g^m y^r) \bmod p,$$

  que no es un logaritmo discreto.

- Es fácil, para un atacante fabricar una terna m, r, s que sea una firma correcta, tal y como se describe en el Ejercicio 4.3. Por ello, para que el algoritmo DSA sea considerado un algoritmo de firma digital seguro es necesario utilizarlo con las llamadas funciones Hash.

Ejercicios de logaritmo discreto

Ejercicio 4.1. Intenta calcular los siguientes logaritmos discretos mediante alguno de los métodos descritos.

$$\log_{118}(2) \text{ mód } 127,$$
$$\log_{28}(115) \text{ mód } 379,$$
$$\log_{186}(87) \text{ mód } 223,$$
$$\log_{97}(103) \text{ mód } 127,$$
$$\log_{240}(20) \text{ mód } 347$$

Ejercicio 4.2. Sea $\mathbb{F}_{81} = \mathbb{Z}_3[\alpha]_{\alpha^4-\alpha^3-1}$, donde representamos $\mathbb{Z}_3 = \{-1, 0, 1\}$. Comprueba que α es un generador de $\mathbb{F}_{81}^*$. Calcula $\log_\alpha(\alpha+1)$ usando el cálculo de índices. Deduce del valor anterior que $\alpha + 1$ no es un generador de $\mathbb{F}_{81}$.

Ejercicio 4.3. Sea (p, q, g, y) la clave pública de un esquema de firma DSA. Sean $0 < a, b < q$, $r = \left(g^a y^b \text{ mód } p\right) \text{ mód } q$, $s = rb^{-1} \text{ mód } q$ y $m = arb^{-1} \text{ mód } q$. Comprueba que la firma (m, r, s) es una firma correcta.

Ejercicio 4.4. ¿Por qué no es bueno emplear el grupo aditivo $\mathbb{Z}_n$ para diseñar un protocolo tipo Diffie-Helmann?

Ejercicio 4.5.

1. Genera parámetros p, g donde p es un primo tal que $2^9 \leq p < 2^{10}$, y g es un generador de $\mathbb{F}_p^*$.

2. Genera dos parejas de claves pública/privada usando los parámetros anteriores para dos usuarios y simula un intercambio de claves de Diffie y Hellman, calculando la clave compartida desde el punto de vista de cada uno de los usuarios.

3. Cifra el mensaje $\mathtt{m} = \mathtt{0b101101100}$ mediante el criptosistema del ElGamal usando una de las dos claves públicas anteriores.

4. Descifra el criptograma obtenido.

Ejercicios de evaluación de logaritmo discreto

Ejercicio. Los parámetros de un criptosistema de ElGamal son $p = 211$ y $g = 3$, es decir, el criptosistema está diseñado en el cuerpo $\mathbb{F}_{211} = \mathbb{Z}_{211}$ y tomamos como generador de $\mathbb{F}_{211}^*$, $g = 3$. La clave pública empleada es $3^a = 109$ mód 211. Descifra el criptograma $(154, \texttt{dni} \text{ mód } 211)$, donde $\texttt{dni}$ es el número de tu DNI. Para calcular los logaritmos discretos necesarios emplea dos de los métodos descritos en la teoría.

Capítulo 5

Criptosistemas basados en curvas elípticas

5.1 Formas simplificadas.

Recordemos que una curva elíptica E sobre un cuerpo K es el conjunto de los $(x, y) \in K^2$ tales que

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{5.1}$$

junto con un punto $\mathcal{O}$.

Proposición 5.1. *Sea* $p = \mathrm{char}(K)$ *y sea* E *una curva elíptica definida sobre* K.

1. *Si* $p > 3$*, la ecuación de* E *puede simplificarse a*

$$y^2 = x^3 + ax + b, \tag{5.2}$$

   que recibe el nombre de forma de Weiertrass.

2. *Si* $p = 3$*, la ecuación de* E *puede simplificarse a*

$$y^2 = x^3 + ax^2 + bx + c. \tag{5.3}$$

3. Si $p = 2$*, la ecuación de* E *puede simplificarse a*

$$y^2 + xy = x^3 + ax^2 + b \text{ si } a_1 \neq 0, \tag{5.4}$$

$$y^2 + ay = x^3 + bx + c \quad \text{si } a_1 = 0, \tag{5.5}$$

denominadas también forma de Weiertrass.

Demostración. Los casos $p > 2$ ya han sido discutidos en la asignatura. Si $p = 2$, los cambios de variable

$$x = a_1^2 x + a_3 a_1^{-1}, \quad y = a_1^3 y + (a_1^2 a_4 + a_3^2) a_1^{-3}$$

cuando $a_1 \neq 0$, y

$$x = x + a_2$$

cuando $a_1 = 0$, transforman (5.1) en (5.4) y (5.5) respectivamente. □

Para sus aplicaciones en criptografía, las curvas interesantes son curvas sobre $\mathbb{Z}_p$ con $p > 3$ suficientemente grande, por lo que podemos suponer que satisfacen (5.2), y curvas sobre $\mathbb{F}_{2^l}$ con l grande y satisfaciendo la ecuación (5.4).
Como las primeras ya han sido tratadas con anterioridad, vamos a centrarnos en el segundo caso.

5.2

Característica 2

Proposición 5.2. *Una curva* $E(a, b)$ *sobre* $\mathbb{F}_{2^l}$ *satisfaciendo* (5.4) *tiene un punto singular si y solo si* $b = 0$.

Demostración. Sea $F(x,y) = y^2 + xy + x^3 + ax^2 + b$. E es singular en $(x_0, y_0) \in E$ si y solo si

$$\frac{\partial F}{\partial x}(x_0, y_0) = \frac{\partial F}{\partial y}(x_0, y_0) = 0.$$

Dado que

$$\frac{\partial F}{\partial x} = y + 3x^2 + 2ax = y + x^2$$

y

$$\frac{\partial F}{\partial y} = 2y + x = x,$$

el único punto donde puede haber una singularidad es $(0,0)$ que pertenece a la curva si y solo si $b = 0$. □

Lema 5.3. *Sea* $E = E(a,b)$ *una curva sobre* $\mathbb{F}_{2^l}$ *definida por la ecuación* (5.4). *Si* $(x_0, y_0), (x_0, y_1) \in E$, *entonces* $y_1 = y_0$ *o* $y_1 = x_0 + y_0$.

Demostración. Dado que

$$y_0^2 + x_0 y_0 = x_0^3 + a x_0^2 + b = y_1^2 + x_0 y_1,$$

tenemos que

$$(y_1 + y_0)^2 = y_1^2 + y_0^2 = (y_1 + y_0) x_0.$$

Si $y_0 \neq y_1$, tenemos que $y_0 + y_1 \neq 0$ y en consecuencia $x_0 = y_1 + y_0$, por tanto $y_1 = y_0$ o $y_1 = y_0 + x_0$. □

Proposición 5.4. *Sea* $E(a,b)$ *una curva elíptica sobre* $\mathbb{F}_{2^l}$ *satisfaciendo* (5.4). *Sean* $P = (x_0, y_0)$, $P_1 = (x_1, y_1)$ *y* $P_2 = (x_2, y_2)$ *puntos de* $E(a,b)$. *Entonces,*

(1) $-P = (x_0, x_0 + y_0)$.

(2) Si $P_2 = -P_1$, $P_1 + P_2 = \mathcal{O}$.

(3) Si $P_2 \neq -P_1$, $P_1 + P_2 = P_3$, *viene dado por*

$$\begin{aligned} P_3 &= (x_3, y_3) \\ &= (m^2 + m + a + x_1 + x_2, m(x_1 + x_3) + x_3 + y_1), \end{aligned}$$

donde

$$m = \begin{cases} (y_2 + y_1)(x_2 + x_1)^{-1} & \text{si } x_1 \neq x_2, \\ x_1 + y_1 x_1^{-1} & \text{si } x_1 = x_2. \end{cases}$$

Demostración. La ecuación (5.1) particulariza a (5.4) tomando $a_1 = 1, a_3 = 0, a_2 = a, a_4 = 0, a_6 = b$. Por tanto la aritmética es consecuencia de la aritmética definida para una curva elíptica genérica, observando que, por el Lema 5.3, $x_1 = x_2$ y $P_2 \neq -P_1$ implica $P_2 = P_1$ y por tanto $P_1 + P_2 = 2P_1$. □

Ejemplo 5.5. Sea $\mathbb{F}_8 = \mathbb{F}_2[\xi]_{\xi^3+\xi+1}$. Sea $E = E(\xi + 1, \xi)$, es decir, dada por la ecuación

$$y^2 + xy = x^3 + (\xi + 1)x^2 + \xi.$$

Vamos a calcular $P + Q$ con $P = (\xi + 1, \xi)$ y $Q = (\xi^2 + \xi, \xi^2)$. El coeficiente m es

$$\begin{aligned} m = (\xi + \xi^2)(\xi + 1 + \xi^2 + \xi)^{-1} &= (\xi + \xi^2)(1 + \xi^2)^{-1} \\ &= \xi^4 \xi^{-6} = \xi^5 = \xi^2 + \xi + 1. \end{aligned}$$

$P + Q = (x_3, y_3)$ donde

$$\begin{aligned} x_3 &= (\xi^2 + \xi + 1)^2 + \xi^2 + \xi + 1 + \xi + 1 + \xi + 1 + \xi^2 + \xi \\ &= \xi + 1 + 1 = \xi \end{aligned}$$

e

$$y_3 = (\xi^2 + \xi + 1)(\xi + 1 + \xi) + \xi + \xi = \xi^2 + \xi + 1,$$

luego $P + Q = (\xi, \xi^2 + \xi + 1)$.
Para calcular $2P = P + P$,

$$\begin{aligned} m &= \xi + 1 + \xi(\xi + 1)^{-1} = \xi + 1 + \xi(\xi^2 + \xi) = \xi^2, \\ x_3 &= (\xi^2)^2 + \xi^2 + (\xi + 1) + (\xi + 1) + (\xi + 1) \\ &= \xi^2 + \xi + \xi^2 + \xi + 1 = 1 \\ y_3 &= \xi^2(\xi + 1 + 1) + 1 + \xi \\ &= (\xi + 1) + 1 + \xi = 0, \end{aligned}$$

luego $2P = (1, 0)$.

Finalmente, el Teorema de Cassel es válido en característica 2, luego $E \cong \mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2}$, con $d_1 \mid d_2$ y $d_1 \mid 2^l - 1$.

5.3 Complejidad de la aritmética en EC

Proposición 5.6. *Sea* E *una curva elíptica dada por* (5.2) *o* (5.4) *en función de la característica del cuerpo base* $\mathbb{F}_q$. *Dados* $P, Q \in E$ *el cálculo* $P + Q$ *es* $\mathcal{O}\left((\log q)^3\right)$.

Demostración. Por la Proposición 5.4 en característica 2, si $P \neq Q$ hay que realizar 9 sumas, 3 multiplicaciones y 1 inverso, y si $P = Q$ una suma menos, por lo que la complejidad está dominada por el cálculo del inverso que es $\mathcal{O}\left((\log q)^3\right)$ por el análogo polinomial de la Proposición 2.7. En característica $p > 3$ el resultado es equivalente. □

Proposición 5.7. *Sea* E *una curva elíptica dada por* (5.2) *o* (5.4) *en función de la característica del cuerpo base* $\mathbb{F}_q$. *Dados* $P \in E$ *y* $m \in \mathbb{N}$, *el cálculo* mP *es* $\mathcal{O}\left((\log q)^3(\log m)\right)$.

Demostración. Empleando el equivalente a los cuadrados iterados pero en notación aditiva (duplicados iterados) tenemos que

$$mP = 2\,(2\,(\cdots 2\,(2\,(m_t P) + m_{t-1} P)\cdots) + m_1 P) + m_0 P$$

donde $m_t m_{t-1} \cdots m_1 m_0$ es la expresión binaria de m. Hay que realizar un máximo de $3\log_2 m$ sumas para calcular mP. El resultado se deduce de la Proposición 5.6. □

Estas dos proposiciones nos indican que la aritmética en una curva elíptica es una operación eficiente. El problema del logaritmo discreto en una curva elíptica E sobre un cuerpo finito consiste en dado $P \in E$ y $Q \in \langle P\rangle$, calcular $\log_P(Q)$, donde

$$\log_P(Q) = m \iff Q = mP.$$

Para el cálculo del logaritmo en curvas elípticas podemos emplear cualquiera de los dos algoritmos genéricos conocidos, el Algoritmo 1 de Shanks y el Algoritmo 2 de Silver-Pohlig-Hellman. Para aplicar el primero necesitamos una cota superior del orden de E, y para el segundo

necesitamos conocer explícitamente dicho orden y que se descomponga como producto de primos pequeños. En cualquier caso ninguno de estos algoritmos es polinomial.

5.4 Parámetros para uso criptográfico

Para su uso en criptografía, necesitamos los siguientes parámetros:

- El cuerpo base $\mathbb{F}_q$
- Los parámetros $a, b \in \mathbb{F}_q$ que definen la curva E mediante las ecuaciones (5.2) o (5.4).
- Un punto base $Q \in E$ cuyo orden n es un primo grande.
- El cofactor h tal que $|E| = hn$.

Lema 5.8. *Sea* E *una curva elíptica tal que* $|E| = hn$ *con* n *primo y* $h < n$. *Entonces* E *tiene un único subgrupo* E_n *de orden* n *que es cíclico y generado por cualquiera de sus elementos distintos de* $\mathcal{O}$.

Demostración. Por el Teorema de Cassel, $E \cong \mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2}$, con $d_1 \mid d_2$. Como $h < n$ tenemos que $n \mid d_2$ pero $n \nmid d_1$. Por tanto E_n se corresponde con el subgrupo $\{0\} \times \left\langle \frac{d_2}{n} \right\rangle \leq \mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2}$. □

Las curvas elípticas que satisfacen que $|E| = hn$ con n primo y h pequeño se llaman *curvas de orden próximo a primo*, y E_n es el *subgrupo de orden primo*.

Selección de la curva. Como el grupo que vamos a emplear para las claves es en realidad E_n, necesitamos que el problema del logaritmo discreto sea difícil en E_n. Para seleccionar la curva debemos evitar los siguientes casos

- Curvas supersingulares (por ejemplo las definidas por (5.5)). En estas curvas $E_n \cong \mathbb{F}_{q^l}^*$ con l pequeño. Este hecho fue demostrado por Menezes-Okamoto-Vanstone empleando el llamado par de Weil. En realidad basta con comprobar que $n \nmid q^l - 1$ para valores pequeños de l.
- Curvas sobre $\mathbb{F}_p$ tales que $|E| = p$. En este caso, Semaev, Smart y Satoh-Araki construyen un isomorfismo $E \cong \mathbb{F}_p$ mediante un algoritmo en tiempo polinomial, lo que reduce el logaritmo discreto al uso del algoritmo de Euclides extendido en $\mathbb{F}_p$.

La curva debe elegirse mediante una búsqueda aleatoria para evitar sesgos en familias que hipotéticamente pudieran ser comprobadas como inseguras en el futuro. Se eligen los parámetros $a, b \in \mathbb{F}_q$. Se calcula $|E(a,b)|$ y se observa si $|E(a,b)| = hn$ para h pequeño con n primo. Se comprueba que no son vulnerables a los ataques anteriores.
La parte difícil del procedimiento es calcular el orden de la curva. El Teorema de Hasse nos da una cota para el mismo. Esta versión del Teorema de Hasse, cuya demostración escapa del ámbito de este curso, sigue siendo cierta en característica 2.

Teorema 5.9 (Hasse)**.** *Sea* E *una curva elíptica sobre* $\mathbb{F}_q$ *dada por* (5.1) *y sea* $t = q + 1 - |E|$. *Entonces*

$$|t| \leq 2\sqrt{q}.$$

Un algoritmo debido a Schoof-Elkies-Atkin para cuerpos primos, y a Satoh para cuerpos binarios permite calcular dicho orden en tiempo polinomial.
Existen suficientes curvas con orden próximo a primo para que una búsqueda aleatoria sea efectiva en la práctica. Si el cuerpo base es primo, existen muchas curvas de orden primo. En característica 2 veremos que el orden es par.

Puntos de la curva. Sea $E = E(a, b)$ una curva elíptica sobre $\mathbb{F}_q$ el morfismo

$$\pi : E \setminus \{\mathcal{O}\} \to \mathbb{F}_q, \quad (x, y) \mapsto x$$

no es sobreyectivo, por lo que no todos los elementos de $\mathbb{F}_q$ son primera coordenada de un punto de la curva.

Caso $\mathbb{F}_p$. La curva viene dada por (5.2), debemos buscar valores $x_0 \in \mathbb{F}_p$ tales que $x_0^3 + ax_0 + b$ es un cuadrado perfecto. Necesitamos por tanto decidir si $\beta \in \mathbb{F}_p$ es residuo cuadrático y, en caso afirmativo, calcular sus raíces cuadradas.

Lema 5.10 (Criterio de Euler)**.** $\beta \in \mathbb{F}_p$ *es residuo cuadrático si y solo si* $\beta^{(p-1)/2} \equiv 1 \mod p$.

Demostración. Si $\beta = \gamma^2$, entonces $\beta^{(p-1)/2} = \gamma^{p-1} \equiv 1 \mod p$ por el Teorema pequeño de Fermat.
Recíprocamente, supongamos que $\beta^{(p-1)/2} \equiv 1 \mod p$. Sea $g \in \mathbb{F}_p$ un elemento primitivo tal que $\beta = g^j$. Si j es impar, $d = (j(p-1)/2, p-1)$ es un divisor propio de $p - 1$, y como

$$1 \equiv (g^j)^{(p-1)/2} = g^{j(p-1)/2} \mod p,$$

tenemos que $g^d \equiv 1 \mod p$, lo que contradice la primitividad de g. Por tanto j es par y β es un residuo cuadrático. □

Una vez que tenemos un procedimiento rápido para saber si un elemento dado tiene raíz cuadrada podemos calcular dicha raíz. Si $\beta \in \mathbb{F}_p$ es un residuo cuadrático y $p \equiv 3 \mod 4$, entonces $\left(\beta^{\frac{p+1}{4}}\right)^2 = \beta^{\frac{p+1}{2}} = \beta\beta^{\frac{p+1}{2}-1} = \beta\beta^{\frac{p-1}{2}} \equiv \beta \mod p$, por lo que $\pm\beta^{(p+1)/4} \mod p$ son sus dos raíces cuadradas.
Nos queda el caso $p \equiv 1 \mod 4$, en el que $\frac{p-1}{2}$ es par. Sea $\gamma \in \mathbb{F}_p$ un elemento no residuo cuadrático, es decir, satisface $\gamma^{(p-1)/2} \equiv -1 \mod p$, sean r impar y $l \geq 1$ tales que $\frac{p-1}{2} = 2^l r$. Tenemos que

$$\beta^{2^l r}\gamma^0 = \beta^{(p-1)/2}\gamma^0 \equiv 1 \mod p.$$

Supongamos que hemos calculado s_{i-1} tal que $2^{l-i+2} \mid s_{i-1}$ y

$$\beta^{2^{l-i+1}r}\gamma^{s_{i-1}} \equiv 1 \mod p.$$

Sea $y_i = \beta^{2^{l-i}r}\gamma^{s_{i-1}/2} \mod p$. Por nuestra hipótesis $y_i^2 \equiv 1 \mod p$, por lo que $y_i \equiv \pm 1 \mod p$. Si $y_i \equiv 1 \mod p$, llamamos $s_i = \frac{s_{i-1}}{2}$, por lo que $2^{l-i+1} \mid s_i$ y

$$\beta^{2^{l-i}r}\gamma^{s_i} = \beta^{2^{l-i}r}\gamma^{s_{i-1}/2} \equiv y_i \equiv 1 \mod p.$$

Si $y_i \equiv -1 \mod p$, llamamos $s_i = \frac{s_{i-1}}{2} + \frac{p-1}{2}$. Dado que $2^l \mid \frac{p-1}{2}$ tenemos que $2^{l-i+1} \mid s_i$. Además

$$\begin{aligned}\beta^{2^{l-i}r}\gamma^{s_i} &= \beta^{2^{l-i}r}\gamma^{s_{i-1}/2}\gamma^{(p-1)/2} \\ &\equiv y_1\gamma^{(p-1)/2} \equiv (-1)(-1) = 1 \mod p.\end{aligned}$$

En ambos casos, hemos encontrado s_i tal que $2^{l-i+1} \mid s_i$ y

$$\beta^{2^{l-i}r}\gamma^{s_i} \equiv 1 \mod p.$$

Después de l pasos, llegamos a $s = s_l$ tal que $2 = 2^{l-l+1} \mid s_l$ y

$$\beta^r\gamma^s = \beta^{2^{l-l}r}\gamma^{s_l} \equiv 1 \mod p.$$

Por tanto,

$$\left(\beta^{\frac{r+1}{2}}\gamma^{\frac{s}{2}}\right)^2 = \beta^{r+1}\gamma^2 \equiv \beta \mod p,$$

lo que nos da una raíz cuadrada de β.
El cálculo de raíces cuadradas nos permite encontrar puntos de la curva en el caso $\mathbb{F}_p$.

Ejemplo 5.11. Vamos a calcular los puntos de la curva $E = E(1,7)$ sobre $\mathbb{F}_{17}$, cuya ecuación es

$$y^2 = x^3 + x + 7.$$

Para cada $\alpha \in \mathbb{F}_{17}$ comprobamos si $\alpha^3 + \alpha + 7$ es residuo cuadrático, lo que podemos ver en el cuadro 5.1 Por tanto tenemos tres puntos, $(2,0), (7,0), (8,0) \in E$ junto con otros cuatro valores $1, 5, 6, 12$ que pueden proporcionar puntos de la curva. Por ejemplo, para $\alpha = 6$ debemos calcular las dos raíces cuadradas de $\beta = 8$. Como $17 \equiv 1 \mod 4$, nos encontramos en el segundo caso. Buscamos un elemento de $\mathbb{F}_{17}$ que no sea residuo cuadrático, por ejemplo $\gamma = 12$. Como $\frac{17-1}{2} = 2^3$,

Cuadro 5.1: Residuos cuadráticos en $E(1,7)$

α	$\beta = \alpha^3 + \alpha + 7$	β^8
0	7	16
1	9	1
2	0	0
3	3	16
4	7	16
5	1	1
6	8	1
7	0	0
8	0	0
9	14	16
10	14	16
11	6	16
12	13	1
13	7	16
14	11	16
15	14	16
16	5	16

tenemos que dar tres pasos empezando en la identidad $\beta^{2^3}\gamma^0 = 1$

$$\begin{gathered}
\beta^{2^3}\gamma^0 \equiv 1 \quad \text{mód } 17,\\
y_1 = \beta^{2^2}\gamma^0 \equiv -1 \quad \text{mód } 17,\\
\beta^{2^2}\gamma^8 \equiv 1 \quad \text{mód } 17,\\
y_2 = \beta^2\gamma^4 \equiv -1 \quad \text{mód } 17,\\
\beta^2\gamma^{4+8} = \beta^2\gamma^{12} \equiv 1 \quad \text{mód } 17,\\
y_3 = \beta\gamma^6 \equiv -1 \quad \text{mód } 17,\\
\beta\gamma^{6+8} = \beta\gamma^{14} \equiv 1 \quad \text{mód } 17,
\end{gathered}$$

de donde

$$\beta^2\gamma^{14} \equiv \beta \quad \text{mód } 17$$

y

$$\sqrt{\beta} = \pm\beta\gamma^7 \equiv \pm 5 \quad \text{mód } 17.$$

Esta identidad nos da dos nuevos puntos $(6,5), (6,12) \in E$. Los demás puntos se calculan de forma análoga.

Caso $\mathbb{F}_{2^\ell}$. Nuestra curva elíptica $E = E(a,b)$ viene dada por la ecuación (5.4). Si $(x_0, y_0) \in E$, y_0 es solución de la ecuación cuadrática

$$y^2 + x_0 y = x_0^3 + a x_0^2 + b.$$

Para ello necesitamos algunos resultados relativos a la resolución de ecuaciones cuadráticas en característica 2. Recordemos, que para todo $\alpha \in \mathbb{F}_{2^\ell}$, se define la traza como

$$\mathrm{Tr}(\alpha) = \textstyle\sum_{j=0}^{\ell-1} \alpha^{2^j}.$$

La raíz cuadrada en $\mathbb{F}_{2^\ell}$ siempre existe y es única, ya que la aplicación $\tau : \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$ definida por $\tau(\alpha) = \alpha^2$ es un automorfismo de álgebras, el automorfismo de Frobenius. De hecho τ es automorfismo de álgebras porque $(\alpha+\beta)^2 = \alpha^2 + \beta^2$ en característica 2. En particular

$$\mathrm{Tr}(\alpha) = \textstyle\sum_{j=0}^{\ell-1} \tau^j(\alpha),$$

lo que implica

$$\mathrm{Tr}(\alpha+\beta) = \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta).$$

Además,

$$(\alpha^{2^{\ell-1}})^2 = \alpha^{2^\ell} = \alpha\alpha^{2^\ell-1} = \alpha,$$

de donde

$$\tau^{-1}(\alpha) = \alpha^{2^{\ell-1}} = \tau^{\ell-1}(\alpha),$$

es decir, $\sqrt{\alpha} = \alpha^{2^{\ell-1}}$ en $\mathbb{F}_{2^\ell}$. La identidad

$$\begin{aligned}\left(\textstyle\sum_{j=0}^{\ell-1} \alpha^{2^j}\right)^2 &= \textstyle\sum_{j=0}^{\ell-1} \left(\alpha^{2^j}\right)^2 \\ &= \textstyle\sum_{j=0}^{\ell-1} \alpha^{2^{j+1}} = \sum_{i=1}^{\ell} \alpha^{2^i} = \sum_{i=0}^{\ell-1} \alpha^{2^i}\end{aligned}$$

implica

$$\mathrm{Tr}(\alpha)^2 = \mathrm{Tr}(\alpha^2) = \mathrm{Tr}(\alpha),$$

de donde $\mathrm{Tr}(\alpha)$ es raíz del polonomio $z^2 - z \in \mathbb{F}_{2^\ell}[z]$, es decir, $\mathrm{Tr}(\alpha) \in \mathbb{F}_2$.

Proposición 5.12. *Los conjuntos*

$$\{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 0\}, \quad \{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 1\}$$

tienen el mismo cardinal.

Demostración. El polinomio $\sum_{j=0}^{\ell-1} z^{2^j} \in \mathbb{F}_{2^\ell}[z]$ tiene grado $2^{\ell-1}$, luego hay elementos en $\mathbb{F}_{2^\ell}$ que no son raíces del mismo, lo que implica que existe $\alpha_0 \in \mathbb{F}_{2^\ell}$ tal que $\mathrm{Tr}(\alpha_0) = 1$. Dado que

$$\alpha_0 + \{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 0\} \subseteq \{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 1\}$$

y

$$\alpha_0 + \{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 1\} \subseteq \{\alpha \in \mathbb{F}_{2^\ell} \mid \mathrm{Tr}(\alpha) = 0\},$$

tenemos el resultado. □

Dado que

$$z^2 + \beta = \left(z + \tau^{-1}(\beta)\right)^2 \in \mathbb{F}_{2^\ell}[z],$$

las ecuaciones cuadráticas de tipo $z^2 + \beta$ tienen solución siempre. Si $\alpha \neq 0$, el polinomio

$$z^2 + \alpha z + \beta \in \mathbb{F}_{2^\ell}[z]$$

se transforma en

$$\alpha^2 t^2 + \alpha^2 t + \alpha^2 \beta \alpha^{-2} \in \mathbb{F}_{2^\ell}[t]$$

mediante el cambio de variable $z = \alpha t$, por lo que es suficiente con saber buscar las raíces de un polinomio de la forma

$$t^2 + t + \beta \in \mathbb{F}_{2^\ell}[t].$$

Proposición 5.13. *La ecuación*

$$t^2 + t + \beta = 0$$

tiene solución si y sólo si $\mathrm{Tr}(\beta) = 0$*, en cuyo caso, si* t_0 *es una solución entonces* $t_0 + 1$ *es la otra solución.*

Demostración. Si $t_0^2 + t_0 + \beta = 0$, entonces

$$0 = \mathrm{Tr}(t_0^2 + t_0 + \beta) = \mathrm{Tr}(t_0^2) + \mathrm{Tr}(t_0) + \mathrm{Tr}(\beta) \\ = 2\,\mathrm{Tr}(t_0) + \mathrm{Tr}(\beta) = \mathrm{Tr}(\beta).$$

Además

$$(t_0 + 1)^2 + (t_0 + 1) + \beta = t_0^2 + 1^2 + t_0 + 1 + \beta = t_0^2 + t_0 + \beta = 0,$$

por tanto sólo nos queda demostrar la existencia de solución si $\mathrm{Tr}(\beta) = 0$.
Supongamos que ℓ es impar. Sea

$$f(\alpha) = \sum_{j=0}^{\frac{\ell-1}{2}} \alpha^{2^{2j}}.$$

Dado que

$$f(\alpha)^2 + f(\alpha) = \sum_{j=0}^{\frac{\ell-1}{2}} \alpha^{2^{2j+1}} + \sum_{j=0}^{\frac{\ell-1}{2}} \alpha^{2^{2j}} = \mathrm{Tr}(\alpha) + \alpha,$$

para cualquier $\alpha \in \mathbb{F}_{2^\ell}$, tenemos que

$$f(\beta)^2 + f(\beta) + \beta = \mathrm{Tr}(\beta) = 0,$$

por lo que $t_0 = f(\beta)$ es una de las dos soluciones. Finalmente, si ℓ es par, sea $\delta \in \mathbb{F}_{2^\ell}$ tal que $\mathrm{Tr}(\delta) = 1$ (que puede ser fácilmente encontrado mediante una búsqueda aleatoria) y sea

$$t_0 = \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i}.$$

Tenemos que

$$\begin{aligned}
t_0^2 + t_0 &= \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^{j+1}} \right) \beta^{2^{i+1}} \\
&\quad + \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i} \\
&= \sum_{i=1}^{\ell-1} \left(\sum_{j=i}^{\ell-1} \delta^{2^{j+1}} \right) \beta^{2^i} \\
&\quad + \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i} \\
&= \sum_{i=1}^{\ell-1} \left(\sum_{j=i+1}^{\ell} \delta^{2^j} \right) \beta^{2^i} \\
&\quad + \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i} \\
&= \sum_{i=1}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i} + \delta \sum_{i=1}^{\ell-1} \beta^{2^i} \\
&\quad + \sum_{i=0}^{\ell-2} \left(\sum_{j=i+1}^{\ell-1} \delta^{2^j} \right) \beta^{2^i} \\
&= \delta \sum_{i=1}^{\ell-1} \beta^{2^i} + \left(\sum_{j=0}^{\ell-1} \delta^{2^j} \right) \beta \\
&= \delta(\mathrm{Tr}(\beta) + \beta) + (\mathrm{Tr}(\delta) + \delta)\beta \\
&= \delta\,\mathrm{Tr}(\beta) + \beta,
\end{aligned}$$

por lo que $t_0^2 + t_0 + \beta = 0$ ya que $\mathrm{Tr}(\beta) = 0$. □

Ejemplo 5.14. Sean de nuevo $\mathbb{F}_8 = \mathbb{F}_2[\xi]_{\xi^3+\xi+1}$ y la curva $E = E(\xi+1, \xi)$ dada por la ecuación

$$y^2 + xy = x^3 + (\xi + 1)x^2 + \xi.$$

Los primeros dos puntos de E son $\mathcal{O} \in E$ y $(0, \sqrt{\xi}) = (0, \xi^4) = (0, \xi^2 + \xi) \in E$. Para los demás necesitamos conocer $\mathrm{Tr}(\alpha^3 + (\xi +$

Cuadro 5.2: Trazas en $E(\xi+1,\xi)$

α	$\beta=\alpha+(\xi+1)+\xi\alpha^{-2}$	$\mathrm{Tr}(\beta)$
ξ	ξ^2	0
ξ^2	1	1
$\xi+1$	ξ^2	0
$\xi^2+\xi$	ξ^2	0
$\xi^2+\xi+1$	$\xi+1$	1
ξ^2+1	ξ^2+1	1
1	0	0

$1)\alpha^2+\xi)$ para $\alpha\in\mathbb{F}_8$. El cuadro 5.2 contiene dichos valores. Los valores $\xi,\xi+1,\xi^2+\xi,1$ deben dar nuevos puntos de la curva. Por ejemplo, para $x_0=\xi^2+\xi$, transformamos la ecuación

$$y^2+x_0y=x_0^3+(\xi+1)x_0^2+\xi$$

en la ecuación

$$t^2+t=x_0+(\xi+1)+\xi x_0^{-2}=\xi^2+\xi+\xi+1+\xi(\xi^2+1)=\xi^2$$

mediante el cambio $t=yx_0^{-1}$. Como $\ell=3$ es impar, las raíces vienen dadas por

$$f(\xi^2)=\sum_{j=0}^{1}(\xi^2)^{2^{2j}}=(\xi^2)^{2^0}+(\xi^2)^{2^2}=\xi^2+\xi,$$

luego $y_0=f(\xi^2)x_0=(\xi^2+\xi)(\xi^2+\xi)=\xi$ nos da los puntos $(\xi^2+\xi,\xi),(\xi^2+\xi,\xi^2)\in E$.

Ejemplo 5.15. Os dejo como ejercicio calcular los puntos de la curva $E(\xi, \xi + 1)$ sobre $\mathbb{F}_4$.

Como consecuencia de la resolución de ecuaciones cuadráticas en característica 2, observemos que para cualquier $x_0 \in \mathbb{F}_{2^\ell}^*$ la ecuación

$$y^2 + x_0 y = x_0^3 + a x_0^2 + b$$

tiene solución si y solo si $\mathrm{Tr}(x_0 + a + b x_0^{-2}) = 0$, en cuyo caso, si y_0 es solución, $y_0 + x_0$ también lo es. Como $\mathcal{O} \in E(a, b)$ y $(0, \sqrt{\beta}) \in E(a, b)$, tenemos que $|E(a, b)|$ es par. Si asumimos que los elementos de la forma $x_0 + a + b x_0^{-2}$ están uniformemente distribuidos entre aquellos con traza 0 y traza 1, podemos concluir que $|E| \approx 2^\ell$.

Selección del punto base. Sea $E = E(a, b)$ una curva elíptica tal que $|E| = hn$ con n primo y h pequeño. Para encontrar un punto de orden n seleccionamos aleatoriamente $P \in E$, calculamos $Q = hP$ y comprobamos si $Q \neq \mathcal{O}$. Como n es primo y $nQ = \mathcal{O}$, Q será un generador de E_n. Si $Q = \mathcal{O}$ tomamos empezamos con un nuevo P.

5.5 Protocolo ECDH

Fijamos una curva elíptica $E = E(a, b)$ tal que $|E| = hn$ con n primo y h pequeño. Fijamos también Q un elemento de orden n. La estructura de grupo de una curva elíptica permite establecer un protocolo de intercambio de claves análogo al protocolo de Diffie-Hellman. La conjetura de Diffie y Hellman para curvas elípticas puede presentarse como sigue

Conjetura 5.16 (Diffie-Hellman)**.** *Conocidos* $P_A = aQ$ *y* $P_B = bQ$ *para ciertos* $1 \leq a, b \leq n$*, calcular* abQ *es computacionamente equivalente a calcular* $a = \log_Q(P_A)$ *o* $b = \log_Q(P_B)$.

El protocolo es el siguiente:

- Alicia y Bruno se ponen de acuerdo en la curva elíptica E y el punto $Q \in E$.
- Alicia elige aleatoriamente $2 \leq a \leq n-1$ y envía a Bruno $P_A = aQ$.
- Bruno elige aleatoriamente $2 \leq b \leq n-1$, envía a Alicia $P_B = bQ$ y calcula $b(P_A)$.
- Alicia calcula $a(P_B)$.
- La clave compartida es $(ab)Q = a(P_B) = b(P_A)$.

5.6 Criptosistema ElGamal en EC

Parámetros y clave pública. Alicia elige una curva elíptica E de orden hn próximo a primo sobre un cuerpo finito y un punto $Q \in E$ de orden primo n. A continuación elige $2 \leq a \leq n-1$ y hace público (E, Q, aQ), manteniendo en privado el valor de a.

Cifrado. Para enviar un mensaje cifrado a Alicia, Bruno codifica dicho mensaje como $M \in E$, elige aleatoriamente $2 \leq k \leq n-1$, y transmite a Alicia $E(M) = (kQ, M + k(aQ))$.

Descifrado. Alicia calcula $D(C_1, C_2) = C_2 - aC_1$. Efectivamente,

$$D(kQ, M + k(aQ)) = M + k(aQ) - a(kQ) = M.$$

En esta construcción hemos dejado de lado el problema de convertir un mensaje en un punto de la curva. Si empleamos este criptosistema como parte de un criptosistema híbrido, podemos transmitir una clave de sesión como $K = lQ$ para cierto $1 \leq l \leq n$ aleatorio. En cualquier caso, nuestro mensaje puede siempre identificarse con un número dentro de un cierto rango, ¿cómo asociamos unívocamente un número con un punto de nuestra curva?.

5.7 ECDSA

De manera análoga a los protocolos basados en el problema del logaritmo discreto en $\mathbb{Z}_p^*$, las curvas elípticas son la base de un estándar de firma digital basado en la dificultad de resolver el logaritmo discreto en ellas.

Parámetros. Una curva elíptica $E = E(a, b)$ sobre un cuerpo finito $\mathbb{F}_q$, de la que conocemos su orden hn y en la que hemos seleccionado un punto $Q \in E$ de orden primo n.

Generación de claves. Alicia genera de forma aleatoria $1 \leq d_A \leq n - 1$, que mantiene como clave secreta. Calcula $P_A = d_A Q$, que distribuye como clave pública.

Algoritmo de firma. Los mensajes son elementos $m \in \mathbb{Z}_n$, normalmente obtenido como el hash de un mensaje de tamaño arbitrario.

1. Genera aleatoriamente un valor $1 \leq k \leq n-1$.
2. Calcula $kQ = (x, y)$, convierte la coordenada x a un entero[1], y calcula $r = x \bmod n$. Si $r = 0$ volvemos al paso 1.
3. Calcula $s = k^{-1}(m + rd_A) \bmod n$. Si $s = 0$, se vuelve al paso 1.
4. (r, s) es la firma del mensaje m.

Algoritmo de verificación El mensaje firmado es una tripleta (m, r, s) de elementos en $\mathbb{Z}_n$. Disponemos de P_A, además de los parámetros de la curva, para verificar la firma.

1. Se comprueba que $(r, s) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$, rechazando la firma en caso contrario.
2. Se calcula $v = s^{-1} \bmod n$, $w_1 = mv \bmod n$, $w_2 = rv \bmod n$, y $R = w_1 Q + w_2 P_A \in E$. Si $R = \mathcal{O}$, se rechaza la firma.
3. Si $R = (x', y')$, convertimos x a entero. Si $x' \equiv r \mod n$, la firma se acepta, en caso contrario se rechaza

Proposición 5.17. *Si* (r, s) *es la firma generada para el mensaje* m*, entonces* $x' \equiv r \mod n$.

[1] Si $\mathbb{F}_q = \mathbb{Z}_p$, nuestra coordenada ya es un número entero, si $\mathbb{F}_q = \mathbb{F}_{2^\ell}$, identificamos sus elementos con números en binario.

Demostración. Dado que

$$k \equiv s^{-1}(m + rd_A) \equiv s^{-1}m + s^{-1}rd_A \equiv w_1 + w_2 d_A \mod n,$$

tenemos que

$$kQ = w_1 Q + w_2 d_A Q = w_1 Q + w_2 P_A = R,$$

luego $r \equiv x' \mod n$. □

Seguridad

- Si permitiésemos $r = 0$, la firma no dependería de la clave privada d_A.
- Dados m y r, encontrar s tal que $R = s^{-1}(mQ + rP_A)$ es equivalente a calcular un logaritmo discreto.
- Dado m, encontrar (r, s) tales que r es la primera coordenada de $R = s^{-1}(mQ + rP_A)$ no se sabe que sea equivalente a calcular un logaritmo discreto, sin embargo no hay evidencias de que pueda hacerse de forma eficiente.

5.8 Codificación de mensajes

Sea E una curva elíptica con orden próximo a primo sobre un cuerpo $\mathbb{F}_q$. Es natural que nuestros mensajes puedan verse como elementos $m \in \mathbb{F}_q$. Una forma de codificar en un punto de la curva sería la siguiente:

1. Elegimos aleatoriamente $r \in \mathbb{F}_q^*$ y llamamos $x_0 = rm$.
2. Si existe $y_0 \in \mathbb{F}_q$ tal que $(x_0, y_0) \in E$, asociamos a m el punto $M = (x_0, y_0)$. En caso contrario volvemos al paso anterior.

Como ya hemos analizado, sobre la mitad de los elementos de $\mathbb{F}_q$ aparecen como primera coordenada de un punto en E, por lo que un par de intentos deberían bastar para encontrar un punto. Podemos tratar de que $(x_0, y_0) \in E_n$, lo que baja la probabilidad a $\frac{1}{2n}$. El valor r debe ser transmitido para que el receptor pueda recuperar m a partir de r y x_0.

5.9 Criptosistema de Menezes-Vanstone

La generación de claves es idéntica a ECDH y a ElGamal sobre EC. Los mensajes van a ser parejas $(m_1, m_2) \in \mathbb{F}_q^2$. Para cifrar un mensaje, seleccionamos aleatoriamente $2 \leq k \leq n-1$, calculamos kQ y $(x_0, y_0) = k(aQ)$. Si $x_0 y_0 = 0$ tomamos un nuevo k. El criptograma es

$$E(m_1, m_2) = (kQ, x_0 m_1, y_0 m_2).$$

Para descifrar un criptograma (C_1, c_2, c_3), Alicia calcula $a(C_1) = a(kQ) = k(aQ) = (x_0, y_0)$ y

$$D(C_1, c_2, c_3) = (x_0^{-1} c_2, y_0^{-1} c_3).$$

Los valores x_0, y_0 no son independientes, satisfacen la ecuación de la curva elíptica. Un atacante que averigüe cualquiera de las dos mitades del mensaje, puede calcular la otra mitad. Por este motivo, se suele tomar m_1 como el mensaje y m_2 como valor aleatorio.

5.10 Curvas en OpenSSL

OpenSSL es un proyecto de código abierto que proporciona un conjunto de herramientas robustas, completas y de nivel comercial para los protocolos TLS y SSL. Incluye una biblioteca criptográfica de propósito general. La mejor fuente de información sobre **OpenSSL** podéis encontrarla en `https://www.openssl.org/`.
OpenSSL dispone de varias curvas elípticas implementadas. Podemos acceder al listado de aquellas disponibles en nuestra implementación mediante la orden

```
openssl ecparam -list_curves
```

En mi implementación, la salida a la orden anterior es

```
secp112r1 : SECG/WTLS curve over a 112 bit prime field
secp112r2 : SECG curve over a 112 bit prime field
secp128r1 : SECG curve over a 128 bit prime field
secp128r2 : SECG curve over a 128 bit prime field
secp160k1 : SECG curve over a 160 bit prime field
secp160r1 : SECG curve over a 160 bit prime field
secp160r2 : SECG/WTLS curve over a 160 bit prime field
secp192k1 : SECG curve over a 192 bit prime field
secp224k1 : SECG curve over a 224 bit prime field
secp224r1 : NIST/SECG curve over a 224 bit prime field
secp256k1 : SECG curve over a 256 bit prime field
secp384r1 : NIST/SECG curve over a 384 bit prime field
secp521r1 : NIST/SECG curve over a 521 bit prime field
prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field
prime192v2: X9.62 curve over a 192 bit prime field
prime192v3: X9.62 curve over a 192 bit prime field
prime239v1: X9.62 curve over a 239 bit prime field
prime239v2: X9.62 curve over a 239 bit prime field
prime239v3: X9.62 curve over a 239 bit prime field
prime256v1: X9.62/SECG curve over a 256 bit prime field
sect113r1 : SECG curve over a 113 bit binary field
sect113r2 : SECG curve over a 113 bit binary field
sect131r1 : SECG/WTLS curve over a 131 bit binary field
sect131r2 : SECG curve over a 131 bit binary field
sect163k1 : NIST/SECG/WTLS curve over a 163 bit binary field
sect163r1 : SECG curve over a 163 bit binary field
```

```
  sect163r2 : NIST/SECG curve over a 163 bit binary field
  sect193r1 : SECG curve over a 193 bit binary field
  sect193r2 : SECG curve over a 193 bit binary field
  sect233k1 : NIST/SECG/WTLS curve over a 233 bit binary field
  sect233r1 : NIST/SECG/WTLS curve over a 233 bit binary field
  sect239k1 : SECG curve over a 239 bit binary field
  sect283k1 : NIST/SECG curve over a 283 bit binary field
  sect283r1 : NIST/SECG curve over a 283 bit binary field
  sect409k1 : NIST/SECG curve over a 409 bit binary field
  sect409r1 : NIST/SECG curve over a 409 bit binary field
  sect571k1 : NIST/SECG curve over a 571 bit binary field
  sect571r1 : NIST/SECG curve over a 571 bit binary field
  c2pnb163v1: X9.62 curve over a 163 bit binary field
  c2pnb163v2: X9.62 curve over a 163 bit binary field
  c2pnb163v3: X9.62 curve over a 163 bit binary field
  c2pnb176v1: X9.62 curve over a 176 bit binary field
  c2tnb191v1: X9.62 curve over a 191 bit binary field
  c2tnb191v2: X9.62 curve over a 191 bit binary field
  c2tnb191v3: X9.62 curve over a 191 bit binary field
  c2pnb208w1: X9.62 curve over a 208 bit binary field
  c2tnb239v1: X9.62 curve over a 239 bit binary field
  c2tnb239v2: X9.62 curve over a 239 bit binary field
  c2tnb239v3: X9.62 curve over a 239 bit binary field
  c2pnb272w1: X9.62 curve over a 272 bit binary field
  c2pnb304w1: X9.62 curve over a 304 bit binary field
  c2tnb359v1: X9.62 curve over a 359 bit binary field
  c2pnb368w1: X9.62 curve over a 368 bit binary field
  c2tnb431r1: X9.62 curve over a 431 bit binary field
  wap-wsg-idm-ecid-wtls1: WTLS curve over a 113 bit binary field
  wap-wsg-idm-ecid-wtls3: NIST/SECG/WTLS curve over a 163 bit binary field
  wap-wsg-idm-ecid-wtls4: SECG curve over a 113 bit binary field
  wap-wsg-idm-ecid-wtls5: X9.62 curve over a 163 bit binary field
  wap-wsg-idm-ecid-wtls6: SECG/WTLS curve over a 112 bit prime field
  wap-wsg-idm-ecid-wtls7: SECG/WTLS curve over a 160 bit prime field
  wap-wsg-idm-ecid-wtls8: WTLS curve over a 112 bit prime field
  wap-wsg-idm-ecid-wtls9: WTLS curve over a 160 bit prime field
  wap-wsg-idm-ecid-wtls10: NIST/SECG/WTLS curve over a 233 bit binary field
  wap-wsg-idm-ecid-wtls11: NIST/SECG/WTLS curve over a 233 bit binary field
  wap-wsg-idm-ecid-wtls12: WTLS curve over a 224 bit prime field
  Oakley-EC2N-3:
IPSec/IKE/Oakley curve #3 over a 155 bit binary field.
Not suitable for ECDSA.
Questionable extension field!
  Oakley-EC2N-4:
IPSec/IKE/Oakley curve #4 over a 185 bit binary field.
Not suitable for ECDSA.
Questionable extension field!
  brainpoolP160r1: RFC 5639 curve over a 160 bit prime field
  brainpoolP160t1: RFC 5639 curve over a 160 bit prime field
  brainpoolP192r1: RFC 5639 curve over a 192 bit prime field
  brainpoolP192t1: RFC 5639 curve over a 192 bit prime field
  brainpoolP224r1: RFC 5639 curve over a 224 bit prime field
  brainpoolP224t1: RFC 5639 curve over a 224 bit prime field
  brainpoolP256r1: RFC 5639 curve over a 256 bit prime field
  brainpoolP256t1: RFC 5639 curve over a 256 bit prime field
  brainpoolP320r1: RFC 5639 curve over a 320 bit prime field
```

```
  brainpoolP320t1: RFC 5639 curve over a 320 bit prime field
  brainpoolP384r1: RFC 5639 curve over a 384 bit prime field
  brainpoolP384t1: RFC 5639 curve over a 384 bit prime field
  brainpoolP512r1: RFC 5639 curve over a 512 bit prime field
  brainpoolP512t1: RFC 5639 curve over a 512 bit prime field
  FRP256v1  : FRP256v1
  id-GostR3410-2001-TestParamSet: GOST R 34.10-2001 Test Curve
  id-GostR3410-2001-CryptoPro-A-ParamSet: GOST R 34.10-2001 CryptoPro-A
  id-GostR3410-2001-CryptoPro-B-ParamSet: GOST R 34.10-2001 CryptoPro-B
  id-GostR3410-2001-CryptoPro-C-ParamSet: GOST R 34.10-2001 CryptoPro-C
  id-GostR3410-2001-CryptoPro-XchA-ParamSet: GOST R 34.10-2001 CryptoPro-XchA
  id-GostR3410-2001-CryptoPro-XchB-ParamSet: GOST R 34.10-2001 CryptoPro-XchB
  id-tc26-gost-3410-2012-512-paramSetA: GOST R 34.10-2012 TC26-A
  id-tc26-gost-3410-2012-512-paramSetB: GOST R 34.10-2012 TC26-B
```

Podemos acceder a los parámetros de cada una de dichas curvas. Por ejemplo, la orden

```
openssl ecparam -name sect233r1 -param_enc
explicit -text -noout
```

da como salida

```
Field Type: characteristic-two-field
Basis Type: tpBasis
Polynomial:
    02:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:04:00:00:00:00:00:00:00:01
A:    1 (0x1)
B:
    66:64:7e:de:6c:33:2c:7f:8c:09:23:bb:58:21:3b:
    33:3b:20:e9:ce:42:81:fe:11:5f:7d:8f:90:ad
Generator (uncompressed):
    04:00:fa:c9:df:cb:ac:83:13:bb:21:39:f1:bb:75:
    5f:ef:65:bc:39:1f:8b:36:f8:f8:eb:73:71:fd:55:
    8b:01:00:6a:08:a4:19:03:35:06:78:e5:85:28:be:
    bf:8a:0b:ef:f8:67:a7:ca:36:71:6f:7e:01:f8:10:
    52
Order:
    01:00:00:00:00:00:00:00:00:00:00:00:00:00:
    13:e9:74:e7:2f:8a:69:22:03:1d:26:03:cf:e0:d7
Cofactor:  2 (0x2)
Seed:
    74:d5:9f:f0:7f:6b:41:3d:0e:a1:4b:34:4b:20:a2:
    db:04:9b:50:c3
```

Esta salida se interpreta de la siguiente forma:

- La descripción de la curva nos indica que el polinomio con respecto al que hacemos reducción modular tiene coeficiente 1 en los grados 233, 74 y 0, luego el cuerpo es $\mathbb{F}_{2^{233}} = \mathbb{F}_2[\xi]_{\xi^{233}+\xi^{74}+1}$.

- La curva es $y^2 + xy = x^3 + ax^2 + b$, donde $a = 1$ y b es el polinomio cuyos coeficientes escritos como lista binaria (escrita en hexadecimal) son

```
b = 0x66647ede6c332c7f8c0923bb58213b333b20e9ce4281fe115f7d8f90ad,
```

  es decir

$$\begin{aligned} b = {} & \xi^{230} + \xi^{229} + \xi^{226} + \xi^{225} + \xi^{222} + \xi^{221} + \xi^{218} \\ & + \xi^{214} + \xi^{213} + \xi^{212} + \xi^{211} + \xi^{210} + \xi^{209} + \xi^{207} \\ & + \xi^{206} + \xi^{204} + \xi^{203} + \xi^{202} + \xi^{201} + \xi^{198} + \xi^{197} \\ & + \xi^{195} + \xi^{194} + \xi^{189} + \xi^{188} + \xi^{185} + \xi^{184} + \xi^{181} \\ & + \xi^{179} + \xi^{178} + \xi^{174} + \xi^{173} + \xi^{172} + \xi^{171} + \xi^{170} \\ & + \xi^{169} + \xi^{168} + \xi^{167} + \xi^{163} + \xi^{162} + \xi^{155} + \xi^{152} \\ & + \xi^{149} + \xi^{145} + \xi^{144} + \xi^{143} + \xi^{141} + \xi^{140} + \xi^{139} \\ & + \xi^{137} + \xi^{136} + \xi^{134} + \xi^{132} + \xi^{131} + \xi^{125} + \xi^{120} \\ & + \xi^{117} + \xi^{116} + \xi^{115} + \xi^{113} + \xi^{112} + \xi^{109} + \xi^{108} \\ & + \xi^{105} + \xi^{104} + \xi^{101} + \xi^{100} + \xi^{99} + \xi^{97} + \xi^{96} \\ & + \xi^{93} + \xi^{87} + \xi^{86} + \xi^{85} + \xi^{83} + \xi^{80} + \xi^{79} \\ & + \xi^{78} + \xi^{75} + \xi^{74} + \xi^{73} + \xi^{70} + \xi^{65} + \xi^{63} \\ & + \xi^{56} + \xi^{55} + \xi^{54} + \xi^{53} + \xi^{52} + \xi^{51} + \xi^{50} \\ & + \xi^{49} + \xi^{44} + \xi^{40} + \xi^{38} + \xi^{36} + \xi^{35} + \xi^{34} \\ & + \xi^{33} + \xi^{32} + \xi^{30} + \xi^{29} + \xi^{28} + \xi^{27} + \xi^{26} \\ & + \xi^{24} + \xi^{23} + \xi^{19} + \xi^{18} + \xi^{17} + \xi^{16} + \xi^{15} \\ & + \xi^{12} + \xi^{7} + \xi^{5} + \xi^{3} + \xi^{2} + 1. \end{aligned}$$

- El punto base tiene por coordenadas los polinomios correspondientes a las listas de bits que en hexadecimal se representan co-

mo

$$Q = (0x00fac9dfcbac8313bb2139f1bb755fef65bc391f8b36f8f8eb7371fd558b,$$
$$0x01006a08a41903350678e58528bebf8a0beff867a7ca36716f7e01f81052)$$

- El orden de Q es

$$n = 0x0100000000000000000000000013e974e72f8a6922031d2603cfe0d7$$

- El cofactor es $h = 2$.

La orden

```
openssl ecparam -name prime239v3 -param_enc
explicit -text -noout
```

da como salida

```
Field Type: prime-field
Prime:
    7f:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:7f:ff:ff:
    ff:ff:ff:80:00:00:00:00:00:7f:ff:ff:ff:ff:ff
A:
    7f:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:7f:ff:ff:
    ff:ff:ff:80:00:00:00:00:00:7f:ff:ff:ff:ff:fc
B:
    25:57:05:fa:2a:30:66:54:b1:f4:cb:03:d6:a7:50:
    a3:0c:25:01:02:d4:98:87:17:d9:ba:15:ab:6d:3e
Generator (uncompressed):
    04:67:68:ae:8e:18:bb:92:cf:cf:00:5c:94:9a:a2:
    c6:d9:48:53:d0:e6:60:bb:f8:54:b1:c9:50:5f:e9:
    5a:16:07:e6:89:8f:39:0c:06:bc:1d:55:2b:ad:22:
    6f:3b:6f:cf:e4:8b:6e:81:84:99:af:18:e3:ed:6c:
    f3
Order:
    7f:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:7f:ff:ff:
    97:5d:eb:41:b3:a6:05:7c:3c:43:21:46:52:65:51
Cofactor:  1 (0x1)
Seed:
    7d:73:74:16:8f:fe:34:71:b6:0a:85:76:86:a1:94:
    75:d3:bf:a2:ff
```

Esta salida es más fácil de interpretar pues los valores son enteros escritos en hexadecimal.

- El cuerpo es

$$\mathbb{F}_{0x7fffffffffffffffffffffff7fffffffffff8000000000007fffffffff}.$$

- La curva es $y^2 = x^3 + ax + b$, donde

$$\begin{aligned} a &= 0x7fffffffffffffffffffffff7fffffffffff8000000000007ffffffffc \\ b &= 0x255705fa2a306654b1f4cb03d6a750a30c250102d4988717d9ba15ab6d3e, \end{aligned}$$

- El punto base tiene por coordenadas

$$\begin{aligned} Q = (&0x6768ae8e18bb92cfcf005c949aa2c6d94853d0e660bbf854b1c9505fe95a, \\ &0x1607e6898f390c06bc1d552bad226f3b6fcfe48b6e818499af18e3ed6cf3) \end{aligned}$$

- El orden de Q es

$$n = 0x7fffffffffffffffffffffff7fffff975deb41b3a6057c3c432146526551$$

- El cofactor es $h = 1$

Ejercicio 5.1. Completa la demostración de la Proposición 5.1. Es decir, comprueba que los cambios de variable indicados transforman la ecuación (5.1) en las ecuaciones (5.2), (5.3), (5.4) y (5.5).

Ejercicio 5.2. Sea $\mathbb{F}_8 = \mathbb{F}_2[\xi]_{\xi^3+\xi+1}$. Sea $E = E(\xi+1, \xi)$, es decir, la curva dada por la ecuación

$$y^2 + xy = x^3 + (\xi+1)x^2 + \xi.$$

1. Encuentra todos los $\alpha \in \mathbb{F}_8^*$ tales que

$$\mathrm{Tr}(\alpha + \xi + 1 + \xi\alpha^{-2}) = 0.$$

2. Usa los valores anteriores para calcular todos los puntos de E.

3. Demuestra, usando el Teorema de Cassel, que E es un grupo cíclico.

4. Calcula un generador P de E.

5. Calcula $2P, 3P, 4P$, este último de las dos formas posibles, es decir, $3P + P$ y $2(2P)$.

Ejercicio 5.3. Sea $\mathbb{F}_{16} = \mathbb{F}_2[\xi]_{\xi^4+\xi+1}$. Sea $E = E(\xi^3, \xi^2)$, es decir, la curva dada por la ecuación

$$y^2 + xy = x^3 + \xi^3 x^2 + \xi^2.$$

1. Encuentra todos los $\alpha \in \mathbb{F}_{16}^*$ tales que
$$\mathrm{Tr}(\alpha + \xi^3 + \xi^2\alpha^{-2}) = 0.$$
2. Usa los valores anteriores para calcular $|E|$.
3. Usa el Teorema de Cassel para obtener las posibles estructuras de E como grupo abeliano.
4. Sea $P = (\xi^2, \xi^3 + \xi + 1)$. Comprueba que $P \in E$.
5. Calcula $6P, 9P$. Deduce que E es un grupo cíclico.

Ejercicio 5.4. Sea $\mathbb{F}_{16} = \mathbb{F}_2[\xi]_{\xi^4+\xi+1}$. Sea $E = E(\xi + 1, \xi^2 + \xi)$, es decir, la curva dada por la ecuación
$$y^2 + xy = x^3 + (\xi + 1)x^2 + (\xi^2 + \xi).$$
Toma elementos aleatorios $Q \in E$ y calcula, con el algoritmo de Shanks, $\log_Q \mathcal{O}$, es decir, encuentra n tal que $nQ = \mathcal{O}$. Toma tantos elementos aleatorios como sea necesario para, con la ayuda del Teorema de Hasse, calcular $|E|$.

Bibliografía

[1] Hans Delfs and Helmut Knebl. *Introduction to Cryptography*. Information Security and Cryptography. Springer-Verlag Berlin Heidelberg, 2015.

[2] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, fourth edition, 1960.

[3] Neal Koblitz. *A Course in Number Theory and Cryptography*, volume 114 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 2 edition, 1994.

[4] National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS)*, July 2013.

[5] Harald Niederreiter and Arne Winterhof. *Applied Number Theory*. Springer International Publishing, 2015.

[6] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer International Publishing, 2016.

[7] Joachim von zur Gathen. *CryptoSchool*. Springer-Verlag Berlin Heidelberg, 2015.