Universidad Autónoma de Yucatán

Facultad de Matemáticas

Tesis:

El problema del logaritmo discreto. Caso de estudio: Certicom ECCp-79

Presentada por

Ramón Ariel Alonzo Martínez

para la obtención del grado de Maestro en Ciencias Matemáticas

Dirigida por

Dr. Víctor Manuel Bautista Ancona

Merida, Yucatan, Mexico
Junio 2015

Resumen

En 1997 la compañía Certicom lanzó una serie de retos los cuales tienen como objetivo incrementar y fomentar la apreciación e investigación acerca de la criptografía sobre curvas elípticas.

La criptografía sobre curvas elípticas consiste de los siguientes elementos: un campo finito, una curva elíptica sobre este campo, un punto sobre la curva que funciona como la llave publica y un entero entre cero y el orden de un punto base, este número será la llave privada. La seguridad de la criptografía sobre curvas elípticas está basada en la dificultad de resolver el problema del logaritmo discreto. Los retos planteado por Certicom consisten en resolver este problema en curvas de diferentes grados de dificultad.

Uno de los retos planteados, en la categoría de ejercicios, es ECCp-79, una curva sobre un campo primo de 79 bits de tamaño. En 1997 Robert Harley y Wayne Baisley, del INRIA ubicado en Francia y del Laboratorio Nacional Fermi en Estados Unidos, respectivamente, lograron resolver dicho reto.

El objetivo principal de este trabajo es resolver el reto ECCp-79 propuesto por Certicom.

Agradecimientos

Agradezco al *Consejo Nacional de Ciencia y Tecnología* (CONACYT) por su apoyo para la elaboración de este documento.

Al Dr. Javier Díaz Vargas por el conocimiento y tiempo compartido durante las clases cursadas en esta maestría.

De igual forma a la Facultad de Matemáticas de la *Universidad Autónoma de Yucatán* (UADY), por su apoyo con infraestructura y espacio de laboratorio, en especial a los doctores Gabriel Murrieta Hernández, Alejandro Lara Rodríguez y Ricardo Legarda Sáenz.

Finalmente quiero agradecer al Dr. Victor Bautista Ancona por el apoyo, paciencia y consejo brindado durante la realización de este trabajo de tesis.

Dedicado a mis padres Ariel y Teresa.

Índice general

Capítulo 1

Una Perspectiva Histórica

En este capítulo, se presenta la historia de la criptografía sobre curvas elípticas; desde sus orígenes, hasta su uso comercial y gubernamental. El objetivo, es dar el contexto acerca de como surgió la criptografía sobre curvas elípticas y que hechos llevaron a la compañía Certicom a crear los retos, entre los cuales se encuentra ECCp-79.

1.1. Certicom y la criptografía sobre curvas elípticas

La criptografía sobre curvas elípticas fue descubierta a mediados de la década de 1980 por el Dr. Victor Miller y el Dr. Neal Koblitz. Las curvas elípticas ya eran conocidas con anterioridad pero pocas personas habían considerado sus aplicaciones prácticas, quedando reservadas para el uso en matemáticas puras. La primera aplicación de curvas elípticas en el área de la criptografía fue en 1984 con el algoritmo de H.W. Lenstra para la factorización de números enteros y fue este la inspiración que permitió a Miller y a Koblitz preguntarse si las curvas elípticas podrían formar la base de un sistema de cifrado; y más tarde, a proponer de forma independiente el uso de los grupos de puntos sobre una curva elíptica definida en un campo finito para la creación de sistemas de cifrado basados en el problema del logaritmo discreto.

Varios años después de que la criptografía sobre curvas elípticas (CCE) fuera propuesta, la respuesta por parte de la comunidad científica fue de curiosidad y aprobación. Aunque la mayoría de los investigadores no había trabajado con las curvas elípticas, la idea de un sistema de cifrado basado en curvas algebraicas era bien recibida.

A finales de la década de los ochenta, el interés en la criptografía sobre curvas elípticas creció y atrajo la atención de varios matemáticos. Hasta ese momento, la criptografía sobre curvas elípticas no era considerada como una amenaza

comercial, aún no existían grandes rivalidades en el mundo de los negocios e incluso RSA no se había convertido aún en una gran fuerza comercial.

Fue en esta época cuando tres profesores de la universidad de Waterloo, Gordon Agnew, Ron Mullin y Scott Vanstone, formaron una compañía, ahora llamada Certicom, que se dedicó al desarrollo y promoción de la criptografía sobre curvas elípticas. Investigadores afiliados con Certicom, empezaron a asistir a conferencias y reuniones donde trataban de convencer a los cuerpos de estándares industriales que recomendaran a la criptografía sobre curvas elípticas. Por ejemplo, el algoritmo de firma digital basado en curvas elípticas se estaba estableciendo como una alternativa a ambas: las firmas digitales creadas por RSA y la firma digital basada en campos finitos desarrollada por la agencia de seguridad nacional.

A mediados de la década de los noventa, la Agencia de Seguridad Nacional (NSA, por sus siglas en inglés) que desde su fundación había trabajado bajo un velo de secrecía decidió salir a la luz para convertirse en una organización que participase abiertamente en las investigaciones sostenidas por la comunidad criptográfica. En esta época; Certicom, el mayor promotor de CCE, se encontraba en una ardua batalla contra RSA. Desde un punto comercial, RSA tenía ventaja sobre la compañía canadiense pues ya era una compañía conocida y establecida; sin embargo con el paso del tiempo y con la creación de nuevos métodos para la factorización de números enteros grandes, forzaban a RSA a usar llaves cada vez más largas. Para la gente que entendía la matemática detrás de ambos sistemas era cada vez más claro que, con el paso del tiempo, RSA sería inferior a CCE para sistemas donde el espacio de memoria fuese limitado.

En diciembre de 1995, en una conferencia del Instituto Americano Nacional de Estándares (ANSI, por sus siglas en inglés), la NSA decidió hacer pública su aprobación por la criptografía sobre curvas elípticas; y en 1997, en el primer artículo que se publicaba abiertamente por parte de un investigador de la NSA, se publicó un procedimiento que mejoraba enormemente la eficiencia de CCE sobre una familia de curvas definidas en un campo binario conocidas como curvas binarias anómalas o curvas de Koblitz.

En ese mismo año, Certicom presentó un concurso el cual tenía como propósito incrementar el conocimiento y la apreciación por parte de la industria acerca de la criptografía sobre curvas elípticas y de la dificultad del problema del logaritmo discreto sobre ellas. De igual forma, pretendía estimular la investigación y análisis de la seguridad sobre los sistemas de cifrado en curvas elípticas.
Con el paso del tiempo la preferencia de la NSA hacia la criptografía sobre curvas elípticas fue cada vez más evidente. En 2003, licenció 26 patentes de Certicom relacionadas con CCE por 25 millones de dólares. De esta forma, la criptografía sobre curvas elípticas se convirtió en la recomendación oficial de la NSA para la protección de documentos gubernamentales. En 2009, Certicom se volvió una subsidiaria de BlackBerry Limited.

Capítulo 2

Una Introducción a Campos Finitos

En este capítulo se introducen los campos finitos, estos tienen varias aplicaciones en áreas como la criptografía y la teoría de códigos. Primero se presentan algunas propiedades de los números enteros, para luego definir a los grupos, anillos y por último, a los campo finitos y sus propiedades.

2.1. Algunas propiedades de $\mathbb{Z}$

Definición 2.1. Sean $a, b \in \mathbb{Z}$ con $a \neq 0$. Se dice que a es divisor o factor de b, denotado $a \mid b$ si existe $t \in \mathbb{Z}$ tal que $b = at$. Si no existe $t \in \mathbb{Z}$ tal que $b = at$, se dice que a no es un divisor de b y se denota $a \nmid b$.

Definición 2.2. Un número entero $p \geq 2$ es **primo**, si sus únicos divisores positivos son 1 y él mismo. Si un número entero no es primo, entonces es **compuesto**.

Teorema 2.3 (Algoritmo de la división)**.** *Para cualesquiera $a, b \in \mathbb{Z}$ con $b > 0$, existen únicos $q, r \in \mathbb{Z}$ tales que*

$$a = bq + r, \quad 0 \leq r < b. \tag{2.1}$$

Demostración. Primero se demostrará que (2.1) tiene al menos una solución. Sea S el conjunto:

$$S = \{a - bk : k \in \mathbb{Z}\}.$$

Para el caso:

$$k_1 = \begin{cases} 0 & \text{si } a \geq 0 \\ a & \text{if } a < 0 \end{cases}$$

se tiene que $a - bk_1$ es no negativo. Por lo tanto S contiene elementos no negativos. De aquí se sigue que el subconjunto formado por los elementos no negativos

en S tiene un elemento mínimo. Sea r este elemento y q el valor k correspondiente; es decir, q es el entero más grande tal que $a-bq \geq 0$. Entonces $r = a-bq \geq 0$, mientras que $r - b = a - (q+1)b < 0$. Por lo que se satisface (2.1).

Ahora se demostrará la unicidad de q, r. Sean $q', r' \in \mathbb{Z}$ distintos a q, r tales que

$$bq + r = a = bq' + r', \quad 0 \leq r, r' < b.$$

Entonces $b(q-q') = r'-r$. Si $r' > r$, se tiene $q - q' > 0$. Por lo tanto $b(q-q') = r' - r \geq b$ y $r' \geq b + r \geq b$, lo cual es una contradicción. Si $r > r'$, entonces $q' - q > 0$ y nuevamente se obtiene una contradicción. De aquí, se sigue que $r' = r$ y $q = q'$

■

Definición 2.4 (Máximo común divisor)**.** Para dos números enteros a, b ambos distintos de 0. Se dice que $d \in \mathbb{N}$ es el **máximo común divisor** de a y b, denotado $mcd(a, b)$, si se cumplen las siguientes propiedades:

(i) d es un divisor común, es decir $d \mid a$ y $d \mid b$.

(ii) Si c es un entero tal que $c \mid a$ y $c \mid b$, entonces $c \mid d$.

Definición 2.5 (Primos relativos)**.** Se dice que a y b son **primos relativos** si $mcd(a, b) = 1$.

Proposición 2.6 (Identidad de Bézout)**.** *Sean a, b dos números enteros no ambos cero, y sea $d = mcd(a, b)$. Entonces existen enteros X y Y tales que*

$$aX + bY = d.$$

En particular, si a, b son primos relativos. Se tiene que:

$$aX + bY = 1.$$

Demostración. Sean $a, b \in \mathbb{Z}$ ambos no cero y sea S el conjunto:

$$S = \{aX + bY > 0 : X, Y \in \mathbb{Z}\}$$

con $d = \min(S)$.

Se demostrará que d es un divisor de a y b, y que todo divisor común de a y b debe dividir a d.

Si d no divide a a. Entonces

$$\begin{aligned} a &= qd + r, \quad 0 < r < d \\ qd &= a - r. \end{aligned}$$

Pero $d = aX' + bY', \quad X', Y' \in \mathbb{Z}$. Sigue que

$$\begin{aligned} q(aX' + bY') &= a - r \\ r &= -q(aX' + bY') + a \\ r &= -a(1 - qX') + b(-qY'). \end{aligned}$$

Se puede ver a r como una combinación lineal de a y b, por lo que $r \in S$ pero $r < d$ lo cual es una contradicción pues $d = \min(S)$. Por lo tanto $r = 0$ y $d \mid a$. De igual forma se puede hacer el mismo procedimiento para b y obtener que $d \mid b$.

Ahora sea c un divisor común de a y b se tiene que

$$\begin{aligned} &a = cu, b = cv, \quad u, v \in \mathbb{Z} \\ &d = aX' + bY' \\ &d = c(uX' + vY'). \end{aligned}$$

Por lo que $c \mid d$.

■

Lema 2.7. *Si $a = bq + r$ entonces $mcd(a, b) = mcd(b, r)$.*

Demostración. Sea $d = mcd(a, b)$ y $d' = mcd(b, r)$. Entonces $d \mid a$ y $d \mid b$, por lo tanto $d \mid (a - bq) = r$ y $mcd(b, r) \geq d$. De igual forma,$d' \mid b$ y $d' \mid r$, entonces existen $t_1, t_2 \in \mathbb{Z}$ tales que $b = t_1 d'$ y $r = (a - bq) = t_2 d'$. Sustituyendo en $r = a - bq$, se tiene que:

$$\begin{aligned} t_2 d' &= a - t_1 d' q \\ t_2 d' + t_1 d' q &= a \\ d'(t_2 + t_1 q) &= a. \end{aligned}$$

Entonces $d' \mid a$ y $mcd(a, b) \geq d'$. Por lo tanto $mcd(a, b) = mcd(b, r)$.

■

Teorema 2.8 (El algoritmo de Euclides)**.** *Para cualesquiera $a, b \in \mathbb{Z}$ con $b > 0$, es posible aplicar el algoritmo de la división sucesivamente para obtener las siguientes ecuaciones:*

$$\begin{aligned} a &= bq_1 + r_1 && 0 < r_1 < b, \\ b &= r_1 q_2 + r_2 && 0 < r_2 < r_1, \\ r_1 &= r_2 q_3 + r_3 && 0 < r_3 < r_2, \\ &\cdots && \cdots \\ r_{j-2} &= r_{j-1} q_j + r_j && 0 < r_j < r_{j-1}, \\ r_{j-1} &= r_j q_{j+1}. \end{aligned}$$

El máximo común divisor de a y b es r_j, el último residuo distinto de cero obtenido en el proceso de divisiones.

Demostración. Ya que $b > r_1 > r_2 > \cdots \geq 0$. El algoritmo llega a un residuo $r_{j+1} = 0$, después de un máximo de b pasos. Para demostrar que $r_j = mcd(a, b)$, se puede ver, por el lema 2.7, que:

$$mcd(a, b) = mcd(b, r_1) = mcd(r_1, r_2) = \cdots = mcd(r_{j-1}, r_j).$$

Como $r_j \mid r_{j-1}$, entonces $mcd(r_{j-1}, r_j) = r_j$.

■

Teorema 2.9 (El algoritmo extendido de Euclides)**.** *El algoritmo de Euclides encuentra el máximo común divisor entre dos números enteros, no ambos cero, a, b. Adicionalmente, es posible extender este algoritmo al calcular los siguientes valores además de los residuos r_i.*

$$\begin{aligned} s_0 &= 1, & t_0 &= 0 \\ s_1 &= 0, & t_1 &= 1 \\ &\ldots, & &\ldots \\ s_{i+1} &= s_{i-1} - q_i s_i, & t_{i+1} &= t_{i-1} - q_i t_i. \end{aligned}$$

El proceso termina cuando $r_{j+1} = 0$ y obtenemos la siguiente combinación lineal

$$r_j = as_j + bt_j.$$

A este proceso se le conoce como ***El algoritmo extendido de Euclides****.*

Demostración. Sea $a = r_0$ y $b = r_1$, se tiene $r_i = as_i + bt_i$ para $i \in \{0, 1\}$. Por inducción sobre $i > 1$, se tiene:

$$\begin{aligned} r_{i+1} &= r_{i-1} - r_i q_i \\ &= (as_{i-1} + bt_{i-1}) - (as_i + bt_i)q_i \\ &= (as_{i-1} - as_i q_i) + (bt_{i-1} - bt_i q_i) \\ &= as_{i+1} + bt_{i+1}. \end{aligned}$$

Por lo tanto, s_j y t_j son los coeficientes indicados en la proposición 2.6.

■

Teorema 2.10. *Los coeficientes binomiales $\binom{n}{k}$ satisfacen la siguiente propiedad:*

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

Demostración. Tenemos que

$$\binom{n-1}{k-1} = \frac{(n-1)!}{(k-1)!(n-k)!} = \left(\frac{(n-1)!}{(k-1)!(n-k-1)!}\right)\left(\frac{1}{n-k}\right) \tag{2.2}$$

$$\binom{n-1}{k} = \frac{(n-1)!}{k!(n-1-k)!} = \left(\frac{(n-1)!}{(k-1)!(n-k-1)!}\right)\left(\frac{1}{k}\right). \tag{2.3}$$

Sumando ambas 2.2, 2.3 obtenemos

$$\begin{aligned}\binom{n-1}{k-1}+\binom{n-1}{k} &= \left(\frac{(n-1)!}{(k-1)!(n-k-1)!}\right)\left(\frac{1}{n-k}+\frac{1}{k}\right)\\ &= \left(\frac{(n-1)!}{(k-1)!(n-k-1)!}\right)\left(\frac{n}{k(n-k)}\right)\\ &= \frac{n!}{k!(n-k)!} = \binom{n}{k}.\end{aligned}$$

■

Proposición 2.11. *Los coeficientes binomiales* $\binom{n}{k}$, $0 \leq k \leq n$ *son números enteros.*

Demostración. Utilizando inducción. Tenemos que

$$\binom{1}{0} = \binom{1}{1} = 1.$$

Ahora, supongamos que $\binom{n}{k}$, $0 \leq k \leq n$ son números enteros. Tenemos entonces para $n+1$ que

$$\binom{n+1}{0} = \binom{n+1}{n+1} = 1$$

y para $0 \leq k \leq n-1$, utilizando el teorema 2.10 tenemos que

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$$

lo cual es la suma de dos números enteros.

■

2.2. Congruencias módulo n

Definición 2.12. Para un conjunto S y R una relación binaria sobre S. Se dice que R es una **relación de equivalencia**, si satisface las siguientes propiedades:

(i) Es reflexiva; $a \sim a$ para toda $a \in S$.

(ii) Es simétrica; si $a \sim b$, entonces $b \sim a$ para toda $a, b \in S$.

(iii) Es transitiva; si $a \sim b$ y $b \sim c$, entonces $a \sim c$ para toda $a, b, c \in S$.

Una relación de equivalencia R sobre S determina una **partición** sobre S. Una **clase de equivalencia** de un elemento $a \in S$, son todos aquellos elementos de S equivalentes a a. Denotamos a una clase de equivalencia como sigue:

$$[a] = \{t \in S : a \sim t\}.$$

El conjunto de todas las clases de equivalencia, forma una partición sobre S, tenemos que $[a] = [b]$ si y solo si $a \sim b$.

Definición 2.13. Sean $a, b \in \mathbb{Z}$ y $n \in \mathbb{N}$. Se dice que a es congruente con b módulo n, denotado por $a \equiv b(\text{mód } n)$, si $n \mid a - b$.

Proposición 2.14. *La relación, congruencia módulo n, es una* ***relación de equivalencia*** *sobre $\mathbb{Z}$, las clases de equivalencia están determinadas de la siguiente forma.*

$$[a] = \{t \in \mathbb{Z} : t \equiv a(\textit{mód } n)\}.$$

Demostración. Para $a, b \in \mathbb{Z}$, $n \in \mathbb{N}$. Se tiene que

$$a \sim b \iff a \equiv b(\text{mód } n) \iff n \mid a - b$$

es una relación de equivalencia ya que:

(i) Es reflexiva, pues $n \mid 0 = a - a$, por lo tanto $a \sim a$.

(ii) Es simétrica, pues si $a \sim b$ se tiene que $n \mid a - b$, entonces $n \mid -(a - b)$, por lo tanto $n \mid b - a$ y $b \sim a$.

(iii) Es transitiva, pues si $a \sim b$ y $b \sim c$, se tiene que $n \mid a - b$ y $n \mid b - c$, entonces $n \mid (a - b) + (b - c)$ y $n \mid a - c$, por lo que $a \sim c$.

■

Ejemplo 2.15. Para $n = 5$ se tiene:

$$[0] = \{\ldots, -15, -10, -5, 0, 5, 10, 15, \ldots\}$$

$$[1] = \{\ldots, -14, -9, -4, 1, 6, 11, 16, \ldots\}$$

$$[2] = \{\ldots, -13, -8, -3, 2, 7, 12, 17, \ldots\}$$

$$[3] = \{\ldots, -12, -7, -2, 3, 8, 13, 18, \ldots\}$$

$$[4] = \{\ldots, -11, -6, -1, 4, 9, 14, 19, \ldots\}$$

Proposición 2.16. *Se pueden definir las operaciones de suma y producto sobre un conjunto de clases de equivalencia, usando la relación de equivalencia descrita en la proposición 2.14.*

Demostración. Se define la suma como:

$$[a] + [b] = [a + b].$$

Para ver que está bien definida se supone que $[a] = [a']$ y que $[b] = [b']$.

Entonces $n \mid a - a'$ y $n \mid b - b'$, así que $n \mid (a - a') + (b - b')$, es decir que $n \mid (a + b) - (a' + b')$ lo cual implica:

$$\begin{aligned} a + b &\equiv a' + b'(\text{mód } n) \\ [a + b] &= [a' + b'] \\ [a] + [b] &= [a'] + [b']. \end{aligned}$$

Se define al producto como:

$$[a][b] = [ab].$$

Para ver que está bien definida se supone que $[a] = [a']$ y que $[b] = [b']$.

Entonces $n \mid a - a'$ y $n \mid b - b'$, así que $n \mid b(a - a') \Rightarrow n \mid ab - a'b$, de igual forma $n \mid a'(b - b') \Rightarrow n \mid a'b - a'b'$. Entonces $n \mid (ab - a'b) + (a'b - a'b') \Rightarrow n \mid ab - a'b'$ lo cual implica:

$$\begin{aligned} ab &\equiv a'b' (\text{mód } n) \\ [ab] &= [a'b'] \\ [a][b] &= [a'][b']. \end{aligned}$$

■

Proposición 2.17. *Sea p un número primo y k un entero tal que $1 \leq k \leq p-1$, entonces*

$$\binom{p}{k} = \frac{p!}{k!(p-k)!} \equiv 0 (\textit{mód } p). \tag{2.4}$$

Demostración. Veamos que $p \mid \binom{p}{k}$. Por la proposición 2.11 sabemos que $\binom{p}{k}$ es un entero, p es primo por lo tanto es suficiente ver que p no divide al denominador en 2.4. Como $k < p$ entonces p no es un factor de $k!$, de igual forma $(p-k) < p$ por lo que p no es un factor de $(p-k)$. De aquí se sigue que $p \mid \binom{p}{k}$ y $\binom{p}{k} \equiv 0 (\text{mód } p)$.
■

Proposición 2.18. *Si $at \equiv bt (\textit{mód } n)$, donde t y n son primos relativos, entonces $a \equiv b (\textit{mód } n)$.*

Demostración. $at \equiv bt (\text{mód } n)$ implica que existe $q \in \mathbb{Z}$ tal que:

$$\begin{aligned} at - bt &= qn \\ t(a - b) &= qn. \end{aligned}$$

De aquí se puede ver que $t \mid qn$, como t y n son primos relativos entonces $mcd(t, n) = 1$, entonces $t \mid q$ y $q = tk$ para alguna $k \in \mathbb{Z}$. De aquí se sigue que

$$\begin{aligned} t(a - b) &= tkn \\ (a - b) &= kn. \end{aligned}$$

Por lo tanto $a \equiv b (\text{mód } n)$.

■

Teorema 2.19 (Teorema pequeño de Fermat)**.** *Sea $p \in \mathbb{Z}$ un número primo y $a \in \mathbb{Z}$ un número tal que $mcd(a, p) = 1$. Entonces $a^{p-1} \equiv 1 (\textit{mód } p)$.*

Demostración. Sea S el conjunto:

$$S = \{a : a^p \equiv a(\text{mód } p)\}.$$

Donde $a \in \mathbb{N}$. Entonces $0 \in S$, pues $0^p = 0$ para toda p, entonces $0^p \equiv 0(\text{mód } p)$. Por inducción, se asume que si $k \in S$, entonces $k^p \equiv k(\text{mód } p)$. Se quiere demostrar que $(k+1) \in S$, es decir $(k+1)^p \equiv (k+1)(\text{mód } p)$. Utilizando el Teorema del Binomio, se tiene que:

$$(k+1)^p = k^p + \binom{p}{1}k^{p-1} + \binom{p}{2}k^{p-2} + \cdots + 1^p.$$

Utilizando la proposición 2.17 tenemos que:

$$\binom{p}{i} = \frac{p!}{i!(p-i)!} \equiv 0(\text{mód } p) \quad i \in \{1, 2, \ldots, p-1\}.$$

Por lo tanto se tiene que:

$$(k+1)^p \equiv k^p + 1^p \equiv k + 1(\text{mód } p).$$

Si $mcd(p, a) = 1$ entonces usando la proposición 2.18 se tiene que $a^{p-1} \equiv 1(\text{mód } p)$. Si a es negativo, entonces $a \equiv r(\text{mód } p)$ para alguna $r \in \{0, 1, \ldots, p-1\}$ y por lo tanto $a^p \equiv r^p \equiv r \equiv a(\text{mód } p)$.

■

2.3. Grupos

Definición 2.20. Sea S un conjunto. Una **operación binaria** en S es un mapeo de $S \times S$ a S, es decir una asignación de parejas ordenadas en S a un elemento en el mismo, como el resultado de esta operación se encuentra de nuevo en el conjunto, la operación es **cerrada**.

Definición 2.21. Un **grupo** es un conjunto G junto con una operación binaria $*$ con las siguientes propiedades:

(i) La operación $*$ es asociativa.

(ii) Existe un elemento $1 \in G$ tal que $1 * g = g * 1 = g$ para toda $g \in G$. 1 es conocido como **identidad**.

(iii) Para toda $g \in G$ existe g^{-1} tal que $g * g^{-1} = g^{-1} * g = 1$. Este elemento es conocido como el **inverso** de g.

Se dice que un grupo es **conmutativo** o **abeliano** si cumple la propiedad adicional:

(iv) Para toda $g, h \in G$ se tiene que $g * h = h * g$.

Por simplicidad, se utiliza gh para referirse a $g * h$, esta notación es conocida como **multiplicativa**. A veces es conveniente utilizar $g + h$ en lugar de gh, esta notación es llamada **aditiva**. En este caso se representa a la identidad con el 0 y el inverso de un elemento g es $-g$.

Para expresar la composición de la operación n-veces, donde $n \in \mathbb{N}$, se utiliza:

$$g^n = gg \cdots g \quad (n\text{veces}).$$

Para el caso aditivo:

$$ng = g + g + \cdots + g \quad (n\text{veces}).$$

Definición 2.22. El orden de un grupo G, denotado por $|G|$, es el número de elementos en el grupo. Si $|G|$ es finito, entonces G es **finito** en caso contrario G es **infinito**.

Ejemplos 2.23.

1. $(\mathbb{Z}, +)$. Los números enteros junto a la adición, forman un grupo de orden infinito.

2. $(\mathbb{Q}\backslash\{0\}, \cdot)$. Los números racionales menos el cero, junto a la operación producto, forman un grupo de orden infinito.

Definición 2.24. Un **subgrupo** H de G que es un grupo bajo la misma operación binaria que G.

Definición 2.25. Sea G un grupo y $g \in G$. El **orden** de g es el entero positivo n más pequeño tal que $g^n = 1$, si no existe tal número, entonces decimos que el orden de g, denotado por $ord(g)$, es infinito.

Definición 2.26. Un grupo G es **cíclico** si existe un elemento $g \in G$ tal que para cualquier $h \in G$, existe un entero i tal que $h = g^i$. Este elemento es llamado el **generador** del grupo.

Ejemplo 2.27. Sea $n \in \mathbb{N}$. El conjunto $\{[0], [1], \ldots, [n-1]\}$ de clases de equivalencia, definidas por la relación, congruencia módulo n, y la operación suma, como fue descrita en la proposición 2.16. Es llamado el conjunto de enteros módulo n y es denotado por $\mathbb{Z}/n\mathbb{Z}$. Este es cíclico y tiene a [1] como generador.

Definición 2.28. Sea G un grupo y $g \in G$. El subgrupo que consiste de todas las potencias de g es llamado el subgrupo generado por g y es denotado por $\langle g \rangle$.

2.4. Anillos

Definición 2.29. Un **anillo** es un conjunto R junto con dos operaciones binarias denotadas por $+$ y $\cdot$ con las siguientes propiedades:

(i) $(R, +)$ forma un grupo abeliano. La identidad es representada por el 0.

(ii) La operación $\cdot$ es asociativa, es decir $a\cdot(b\cdot c) = a\cdot(b\cdot c)$ para toda $a, b, c \in R$.

(iii) Se cumplen las leyes distributivas, esto es $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(b + c) \cdot a = b \cdot a + c \cdot a$ para toda $a, b, c \in R$.

Adicionalmente,

(iv) Se dice que el anillo tiene **identidad** o es unitario si existe un elemento $1 \in R$ tal que $1\cdot a = a\cdot 1 = a$ para toda $a \in R$. El elemento 1 es denominado uno.

(v) Un anillo es llamado **conmutativo**, si la operación $\cdot$ es conmutativa.

(vi) Un anillo R es llamado un **dominio entero**, si es conmutativo con identidad distinta del cero y donde $ab = 0$ implica que $a = 0$ o $b = 0$.

Ejemplos 2.30.

1. $(\mathbb{Z}, +, \cdot)$. El conjunto de los números enteros junto con las operaciones suma y producto forma un anillo conmutativo con identidad.
2. $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$. El conjunto de los números enteros módulo n junto con las operaciones suma y producto forma un anillo conmutativo con identidad.

De aquí en adelante, por **anillo** se hace referencia a un anillo conmutativo con identidad.

Proposición 2.31. *Sea R un dominio entero y $a \neq 0 \in R$. Entonces si $ab = ac$ sigue que $b = c$.*

Demostración. Se tiene que $ab = ac$ por lo tanto:

$$ab - ac = 0$$
$$a(b - c) = 0.$$

Como R es un dominio entero y $a \neq 0$ se tiene que:

$$b - c = 0$$
$$b = c.$$

■

Definición 2.32. Sea R un anillo. Denotamos por $R^\times$ al conjunto de unidades en R, es decir a los elementos $a \in R$ tales que existe $b \in R$ con $1 = ab$. Un elemento $p \neq 0 \in R\backslash R^\times$ es llamado un elemento primo de R, si cuando $p \mid ab$, $a, b \in R$, entonces $p \mid a$ o $p \mid b$.

Definición 2.33. Sea $n \geq 1$ un entero. Definimos

$$\varphi(n) = |(\mathbb{Z}/n\mathbb{Z})^\times|.$$

Es decir, $\varphi(n)$ es el número de unidades de $\mathbb{Z}/n\mathbb{Z}$; o bien el número de elementos en $\mathbb{Z}/n\mathbb{Z}$ que poseen un inverso. Se denomina a φ como **función indicatriz de Euler** o **función φ de Euler**.

Ejemplo 2.34. $((\mathbb{Z}/n\mathbb{Z})^{\times}, \cdot)$ es un grupo finito de orden $\varphi(n)$.

Proposición 2.35. *Sea $p \geq 2$ un número entero. Entonces p es un número primo si y solo si p es un elemento primo en $\mathbb{Z}$, de acuerdo a la definición 2.32.*

Demostración.

($\Rightarrow$) Sean $a, b \in R$ tales que $p \mid ab$. Si $p \mid a$, no hay más que demostrar, por lo que se asume que $p \nmid a$. Como los únicos divisores positivos de p son 1 y el mismo, entonces como $p \nmid a$ se tiene que $mcd(p, a) = 1$. Por lo tanto de acuerdo a la propiedad 2.6 existen $s, t \in \mathbb{Z}$ tal que:

$$1 = as + pt.$$

Multiplicando la ecuación anterior por b, se obtiene:

$$b = bas + bpt.$$

Como $p \mid ba$ y $p \mid bp$, entonces $p|(bas + bpt) = b$.

($\Leftarrow$) Sea un elemento primo en $\mathbb{Z}$ entonces si $p = ab$, para $a, b \in \mathbb{Z}$, se tiene que $p \mid a$ o $p \mid b$. De aquí sigue que $a \geq p$ o $b \geq p$, entonces $a = p$ o $b = p$ y p es un número primo.

■

2.5. Campos finitos

Definición 2.36. Un anillo en el cual los elementos distintos del cero forman un grupo abeliano bajo el producto, es llamado un **campo**. En otras palabras, un campo es un anillo conmutativo en el cual todos los elementos distintos del 0 poseen un inverso bajo la operación $\cdot$.

Ejemplo 2.37. $(\mathbb{Q}, +, \cdot)$. Los números racionales junto con las operaciones suma y producto, forman un campo.

Definición 2.38. Un campo con una cantidad finita de elementos es llamado **campo finito**.

Teorema 2.39. *Todos los campos son dominios enteros.*

Demostración. Sea F un campo y $a, b \in F$ tal que $ab = 0$. Entonces si $b \neq 0$ tenemos que

$$a = 1a = (b^{-1}b)a = b^{-1}(ba) = b^{-1}0 = 0.$$

Por lo tanto $a = 0$, de igual forma si suponemos que $a \neq 0$ entonces $b = 0$ y F es un dominio entero.

■

A continuación, se mostrará cómo formar un campo finito, partiendo del anillo de enteros módulo n, $\mathbb{Z}/n\mathbb{Z}$. Si $n = 1$ entonces el anillo no es un campo pues $\mathbb{Z}/1\mathbb{Z}$ esta formado por un solo elemento, cuando al menos necesitamos dos, el 0 y el 1, para que sea considerado un campo.

Pero si n es un número compuesto, como en el siguiente ejemplo:

Ejemplo 2.40. El anillo $\mathbb{Z}/4\mathbb{Z}$. Tabla para el producto.

$\cdot$	0	1	**2**	3
0	0	0	0	0
1	0	1	2	3
2	0	2	**0**	2
3	0	3	2	1

Se puede observar que $2 \cdot 2 = 0$. Es decir $\mathbb{Z}/4\mathbb{Z}$ no es un dominio entero y por lo tanto no puede ser un campo.

Se demostrará que esto ocurre siempre que n es un número compuesto.

Proposición 2.41. *Sea n un número compuesto. Existen al menos dos números en $\mathbb{Z}/n\mathbb{Z}$ ambos distintos de cero tal que su producto es cero.*

Demostración. Sea $n > 1$ un número compuesto. Como n es compuesto entonces $n = ab$ para dos números a,b mayores que 1 y menores que n, pero tenemos que su producto en $\mathbb{Z}/n\mathbb{Z}$ es 0.

■

Por lo tanto $\mathbb{Z}/n\mathbb{Z}$ no es un dominio entero siempre y cuando n sea compuesto.

Teorema 2.42. *Todo dominio entero con una cantidad finita de elementos es un campo.*

Demostración. Sea R un dominio entero con n elementos y $a \in R\backslash\{0\}$ y sean $d_0, d_1 \dots d_{n-1}$ los elementos de R. De acuerdo a la proposición 2.31, se sabe que si $i \neq j$ entonces $ad_i \neq ad_j$. Existen n elementos distintos en R por lo que $\{ad_0, ad_1 \dots ad_{n-1}\}$ es una permutación de los elementos de R y existe una i tal que $ad_i = 1$, sigue entonces que $d_i = a^{-1}$.

■

Ejemplos 2.43.

1. $\mathbb{Z}/p\mathbb{Z}$ con p primo es un campo finito pues es un dominio entero con un número finito de elementos.

2. $\mathbb{Z}/n\mathbb{Z}$ con n compuesto, no es un campo pues al ser n compuesto entonces $\mathbb{Z}/n\mathbb{Z}$ no es un dominio entero.

Definición 2.44. La **característica** de un campo es igual a 0 si

$$1 + 1 + \ldots + 1 \quad \text{(n veces)}$$

nunca es igual a 0 para alguna $n \geq 1$. De otra forma, la característica del campo es el entero positivo m más pequeño tal que $\sum_{i=1}^{m} 1 = 0$.

Ejemplo 2.45. $(\mathbb{Z}/2\mathbb{Z}, +, \cdot)$ es un campo finito con 2 elementos el 0 y el 1, y las operaciones definidas como sigue:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

$(\mathbb{Z}/2\mathbb{Z}, +, \cdot)$ es un campo finito con **característica** 2.

Teorema 2.46. *Sea F un campo. La característica de F es 0 o un número primo.*

Demostración. Si la característica de F es un número compuesto $n = ab$, con a, b mayores que uno y menores que n entonces se tiene que:

$0 = 1{+}1{+}\ldots{+}1 \quad \text{(n veces)} = (1{+}1{+}\ldots{+}1 \quad \text{(a veces)})(1{+}1{+}\ldots{+}1 \quad \text{(b veces)})$.

Pero F es un dominio entero por lo tanto

$$1 + 1 + \ldots + 1 = 0 \quad \text{(a veces)}$$

o

$$1 + 1 + \ldots + 1 = 0 \quad \text{(b veces)}.$$

Esto contradice que n es el número mínimo con esta propiedad, por lo que n es primo.

■

Se sabe que en un campo todos los elementos distintos del cero, tienen un inverso bajo el producto. Se puede utilizar el **algoritmo extendido de Euclides**, teorema 2.9, para hallar los elementos inversos en un campo, $\mathbb{Z}/p\mathbb{Z}$. Se puede ver que al ser p un número primo, entonces será primo relativo para cada elemento en $\mathbb{Z}/p\mathbb{Z}^{\times}$. Así, si $a \in \mathbb{Z}/p\mathbb{Z}^{\times}$ tenemos

$$aX + pY = 1,$$

y al reducir esta ecuación módulo p, tenemos que

$$aX = 1$$

y por lo tanto, $X = a^{-1}$.

Definición 2.47. Sea D un dominio entero. Una función $N : D \to \mathbb{N} \cup \{0\}$ con la propiedad que $N(0) = 0$ es una **norma** para D.

Definición 2.48. Un dominio euclidiano es una pareja (D, N), donde D es un dominio entero, N una norma y se cumple la siguiente propiedad: Dados $a, b \in D, b \neq 0$ se tiene que $b \mid a$ o existen $q, r \in D$ tales que $a = bq + r$ con $N(r) < N(b)$.

Esto nos da una versión más general del algoritmo de la división, teorema 2.3, utilizado en $\mathbb{Z}$.

2.5.1. Construyendo nuevos campos

Definición 2.49. Sea F un campo; un **polinomio** p con indeterminada X sobre F se define como sigue

$$p(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0.$$

Donde cada $a_i \in F$ y $n \geq 0$. El elemento a_i se denomina como el **coeficiente** de X^i en $p(X)$. La n más grande tal que $a_n \neq 0$ recibe el nombre de **grado** del polinomio $p(X)$ y se denota por $grado(p(X))$, a_n recibe el nombre de **coeficiente líder**. Si todos los coeficientes del polinomio son 0, decimos que el polinomio es el **polinomio cero** y tiene grado igual a infinito. Un polinomio cuyo coeficiente líder es 1, recibe el nombre de **polinomio mónico**.

Definición 2.50. Sea F un campo. Denominamos como $F[X]$ al anillo formado por el conjunto de polinomios con coeficientes en F y que cumplen con las siguientes propiedades:

1. **Igualdad**: Dados dos polinomios $p(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ y $q(X) = b_n X^n + b_{n-1} X^{n-1} + \cdots + b_1 X + b_0$ son iguales si sus coeficientes son iguales es decir $a_i = b_i$ para toda i.

2. **Adición**: Sea $p(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ y $q(X) = b_m X^m + b_{m-1} X^{m-1} + \cdots + b_1 X + b_0$ definimos la suma como sigue $p(X) + q(X) = c_s X^s + c_{s-1} X^{s-1} + \cdots + c_1 X + c_0$. Donde $c_i = a_i + b_i$ para cada i y $s = \max(n, m)$. Si $n < m$ agregamos coeficientes $a_{n+1}, a_{n+2}, \cdots, a_s$ a $p(X)$ todos igual a 0, de la misma forma en el caso contrario, si $m < n$, con los coeficientes de $q(X)$.

3. **Producto**: Sea $p(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ y $q(X) = b_m X^m + b_{m-1} X^{m-1} + \cdots + b_1 X + b_0$ definimos el producto como sigue $p(X)q(X) = c_t X^t + c_{t-1} X^{t-1} + \cdots + c_1 X + c_0$ donde $c_i = a_i b_0 + a_{i-1} b_1 + \cdots + a_1 b_{i-1} + a_0 b_i$ para cada i y $t = n + m$.

Teorema 2.51. *Dado un campo F y $F[X]$ su respectivo anillo de polinomios. $F[X]$ es un dominio entero.*

Demostración. Sean $F, G \in F[X] \backslash \{0\}$ definidos como sigue:

$$F = f_n X^n + f_{n-1} X^{n-1} + \ldots + f_1 X + f_0, \quad f_n \neq 0$$
$$G = g_m X^m + g_{m-1} X^{m-1} + \ldots + g_1 X + g_0 \quad g_m \neq 0,$$

f_n y g_m son los coeficientes no cero que acompañan a las potencias de X más altas en cada polinomio respectivamente. Ambas deben existir pues F, G son polinomios distintos de 0. De aquí se sigue que $FG \neq 0$ pues al menos tendrá al término $f_n g_m X^{n+m}$ el cual tiene un coeficiente distinto de cero pues F es un dominio entero.

■

Proposición 2.52. *Sea F un campo. Las únicas unidades en $F[X]$ son las constantes distintas de 0.*

Demostración. Sean $p, q \in F[X]\backslash\{0\}$ definidos de la siguiente forma:

$$p = \alpha_n X^n + \alpha_{n-1} X^{n-1} + \cdots + \alpha_0, \quad \alpha_n \neq 0$$
$$q = \beta_m X^m + \beta_{m-1} X^{m-1} + \cdots + \beta_0, \quad \beta_m \neq 0.$$

Donde q es el inverso de p. En el producto pq tendremos al menos al monomio $\alpha_n \beta_m X^{m+n}$. Como $F[X]$ es un dominio entero tenemos que el producto es distinto de cero. Pero $pq = 1$ por lo que necesitamos que $m + n = 0$ sigue que $m = n = 0$ y ambos p y q son polinomios constantes.

■

Teorema 2.53 (Algoritmo de la división polinomial)**.** *Sea F un campo. $F[X]$ su respectivo anillo de polinomios y $g \neq 0$ un polinomio en $F[X]$. Entonces para cualquier $f \in F[X]$ existen polinomios $q, r \in F[X]$ tal que*

$$f = gq + r,$$

donde $grado(r) < grado(g)$.

Demostración. Sea S el conjunto:

$$S = \{f - gq : q \in F[X]\}.$$

Se tiene que el conjunto $\{grado(s) : s \in S\}$ esta conformado por enteros no negativos, por lo tanto debe tener un elemento mínimo. Sea $r \in S$ este elemento, tenemos que $r = f - gq$ con $q \in F[X]$. Ahora si $r = 0$ entonces $grado(r) < grado(g)$ pues $g \neq 0$. Si $r \neq 0$ entonces se pueden ver a r y a g como polinomios de grado n y m respectivamente:

$$r = r_n X^n + \cdots + r_1 X + r_0, \quad r_n \neq 0,$$
$$g = g_m X^m + \cdots + g_1 X + g_0, \quad g_m \neq 0.$$

Supongamos que $n \geq m$ y consideremos el siguiente polinomio

$$r - \frac{r_n}{g_m} X^{n-m} g = r_n X^n + \cdots + r_1 X + r_0 - \left(r_n X^n + \frac{r_n g_{m-1}}{g_m} X^{n-1} + \cdots \right).$$

Se puede observar que tiene grado menor que n pues los términos de grado n se cancelan. Sin embargo, arreglando los términos de la siguiente forma

$$\begin{aligned} r - \frac{r_n}{g_m} X^{n-m} g &= f - gq - \frac{r_n}{g_m} X^{n-m} g \\ &= f - g\left(q + \frac{r_n}{g_m} X^{n-m}\right), \end{aligned}$$

podemos ver que se trata de un elemento de S. Esto es una contradicción pues posee un grado menor al del elemento mínimo r. De aquí, se sigue que $grado(r) < grado(g)$.

■

Este teorema es una generalización del **algoritmo de la división**, teorema 2.3, para polinomios y nos permite utilizar el **algoritmo extendido de Euclides**, teorema 2.9, en anillos de polinomios.

En $\mathbb{Z}$, no todos los elementos tienen inversos bajo el producto. Sin embargo es posible construir campos como $\mathbb{Z}/p\mathbb{Z}$ cuando p es un número primo. Ahora se mostrara un procedimiento similar utilizando al anillo de polinomios $F[X]$ como punto de partida.

Definición 2.54. Sea M un polinomio en $F[X]$. Definimos $F[X]/(M)$ como el conjunto de clases de equivalencia dado por cada elemento p en $F[X]$. Denotamos a la clase de equivalencia que lo contiene como $[p]$ entonces tenemos que

$$[p] = \{q \in F[X] : q \equiv p \pmod{M}\}.$$

Esta construcción es análoga a la de $\mathbb{Z}/n\mathbb{Z}$, proposición 2.14, y así se puede definir la operación de suma y producto, proposición 2.16, para elementos de $F[X]/(M)$ como sigue:

$$[p] + [q] = [p + q], \qquad \text{y } [p] \cdot [q] = [p \cdot q].$$

Al igual que en $\mathbb{Z}/n\mathbb{Z}$, $F[X]/(M)$ no siempre es un dominio entero. Si M es un polinomio que puede factorizarse como el producto de otros dos polinomios, entonces se tiene que $F[X]/(M)$ no es un dominio entero. Por lo que es necesario que M sea un elemento irreducible en $F[X]$.

Definición 2.55. Sea D un dominio entero. Se dice que $p \in D$ es irreducible si:

(i) No es cero ni unidad.

(ii) Si $p = ab$ entonces a es unidad o b es unidad.

Teorema 2.56. *Si F es un campo y $M \in F[X]$ es distinto de cero, entonces $F[X]/(M)$ es un campo si y solo si M es irreducible.*

Demostración.

($\Leftarrow$) Si M es irreducible, sea k cualquier elemento distinto de 0 en $F[X]/(M)$ se tiene que $k = K + (M)$ con $K \in F[X]$, como M es irreducible y no divide a K, se sigue que $mcd(K, M) = 1$. Se puede usar el **algoritmo extendido de Euclides**, teorema 2.9, para encontrar u y v tal que

$$Mu + Kv = 1.$$

Pero en $F[X]/(M)$, $Mu = 0$ y se tiene que $kv = 1$ por lo que k es invertible y por lo tanto $F[X]/(M)$ es un campo.

($\Rightarrow$) Si M es reducible. Se tienen los siguientes casos

a) Si $M = 0$. Entonces $F[X]/(M) = F[X]$ el cual claramente no es un campo.

b) Si M es una constante distinta de cero. Se tiene que M es una unidad y por lo tanto $(M) = F[X]$ y $F[X]/(M)$ es el anillo cero, el cual no es un campo, pues un campo tiene al menos dos elementos: 0 y 1.

c) Si $M = fg$ con f, g polinomios no constantes y se tiene que:

$$grado(f), grado(g) < grado(M),$$

entonces ambos son distintos de cero en $F[X]/(M)$ pero $fg = M = 0$ en $F[X]/(M)$ por lo tanto $F[X]/(M)$ no es un dominio entero y no puede ser un campo.

■

Definición 2.57. Sean F y F' campos y sea $\pi : F \to F'$ una función. Se dice que π es un homomorfismo de campos, si y solo si:

(i) $\pi(a + a') = \pi(a) + \pi(a')$, para toda $a, a' \in F$.

(iii) $\pi(aa') = \pi(a)\pi(a')$, para toda $a, a' \in F$.

Si $\pi : F \to F'$ es biyectiva y un homomorfismo de campos, se dice que π es un isomorfismo de campos.

Proposición 2.58. *Sea F un campo y $F[X]$ su respectivo anillo de polinomios. Si M es un elemento irreducible en $F[X]$, entonces M es un elemento primo en $F[X]$.*

Demostración. Como M es irreducible, entonces $F[X]/(M)$ es un campo. Sea $\pi : F[X] \to F[X]/(M)$ un homomorfismo de anillos que mapea elementos en $F[X]$ a sus respectivas clases de equivalencia. Se tiene que $M \mid ab$ con $a, b \in F[X]$, entonces $[0] = \pi(ab) = \pi(a)\pi(b) = [a][b]$. Como $F[X]/(M)$ es un dominio entero, entonces $[a] = 0$ o $[b] = 0$, luego $M \mid a$ o $M \mid b$.

■

2.5.2. Propiedades de los campos finitos

Teorema 2.59. *Sea F un campo finito con q elementos, entonces $a^q = a$ para toda $a \in F$.*

Demostración. Se puede suponer que $a \neq 0$, pues $0^q = 0$. Considérese al polinomio xa, se puede ver a este polinomio como una función de F a F, esta función es inyectiva pues si $ba = b'a$ entonces multiplicando ambos lados por a^{-1} se tiene que $b = b'$. Ya que F es una función uno a uno y va de F a si mismo, entonces es sobre. Esta función envía 0 a 0, por lo tanto si se remueve al 0, la función sigue siendo una biyección. Se tiene que

$$\prod_{b \in F^\times} b = \prod_{b \in F^\times} (ab)$$

haciendo $\gamma = \prod_{b \in F^\times} b$, tenemos que $\gamma = a^{q-1}\gamma$, pues

$$\prod_{b \in F^\times} (ab) = a^{q-1} \prod_{b \in F^\times} b. \tag{2.5}$$

Al multiplicar (2.5) por γ^{-1}, se tiene que $a^{q-1} = 1$. De aquí se sigue el resultado. ■

Teorema 2.60. *Sea F un campo finito, con q elementos, entonces $F^\times$ es un grupo cíclico*

Demostración. Se puede asumir $q \geq 3$, ya que el caso $q = 2$ es trivial. Sea h el orden de $F^\times$, tenemos $h = q - 1$. Sea $h = p_1^{r_1} p_2^{r_2} \cdots p_m^{r_m}$ su descomposición en factores primos. Tenemos que para cada $i \in \{1, 2, \ldots, m\}$,$X^{h/p_i} - 1$ tiene a lo más h/p_i raíces en $\mathbb{F}_q$. h/p_i es menor que h, por lo tanto hay elementos distintos de cero en $\mathbb{F}_q$ que no son raíces de este polinomio. Sea a_i uno de estos elementos y

$$b_i = a_i^{h/p_i^{r_i}}.$$

Al elevar b_i a la potencia $p_i^{r_i}$, se tiene que es igual a 1. Por lo tanto el orden de b_i divide a $p_i^{r_i}$ y tiene forma $p_i^{s_i}$ para alguna $1 \leq s_i \leq r_i$. Pero

$$b_i^{p_i^{r_i - 1}} = a_i^{h/p_i} \neq 1.$$

De aquí, el orden de b_i es $p_i^{r_i}$.

Sea $b = b_1 b_2 \cdots b_m$. Si el orden de b es $h = q - 1$, entonces b es un generador del grupo. En caso contrario si el orden de b es un divisor propio de h, entonces es un divisor de al menos uno de los h/p_i, $i \in \{1, \ldots, m\}$. Sin perdida de generalidad, digamos que es h/p_1. Entonces:

$$1 = b^{h/p_1} = b_1^{h/p_1} b_2^{h/p_1} \cdots b_m^{h/p_1}.$$

Pero, si $i \in \{2, \ldots, m\}$, entonces $p_i^{r_i}$ divide a h/p_1 y por lo tanto $b_i^{h/p_1} = 1$. Esto hace que $b_1^{h/p_1} = 1$ y que el orden de b_1 divida a h/p_1, lo cual es imposible pues el orden de b_1 es $p_1^{r_1}$. Por lo tanto $F^\times$ es un grupo cíclico con generador b. ■

2.5.3. Número de elementos en un campo finito

Teorema 2.61. *Sea F un campo finito con q elementos y característica p, entonces $q = p^j$ para algún entero positivo j.*

Demostración. Notemos que $\mathbb{Z}/p\mathbb{Z}$ es un subcampo de F. Podemos ver a F como un espacio vectorial sobre $\mathbb{Z}/p\mathbb{Z}$. Como el campo es finito, entonces el espacio vectorial es finito también. Sea j la dimension de este espacio. Entonces existe una base $\alpha_1, \ldots, \alpha_j$; por lo tanto, cualquier elemento en $b \in F$ puede expresarse de manera única como

$$b = s_1\alpha_1 + \ldots + s_j\alpha_j$$

donde las $s_i \in \mathbb{Z}/p\mathbb{Z}$. Esto nos permite contar el número de elementos en F. Tenemos p opciones para cada s_i por lo que hay p^j elementos en F.

■

Teorema 2.62. *Sea F un campo con característica p tenemos que*

$$(a+b)^p = a^p + b^p$$

para cualesquiera $a, b \in F$

Demostración. Expandiendo $(a+b)^p$ usando el Teorema del Binomio se tiene:

$$(a+b)^p = a^p + \binom{p}{1}a^{p-1}b + \binom{p}{2}a^{p-2}b^2 + \ldots + b^p.$$

Utilizando la proposición 2.17 tenemos que:

$$\binom{p}{i} = \frac{p!}{i!(p-i)!} \equiv 0(\text{mód } p) \quad i \in \{1, 2, \ldots, p-1\}.$$

Por lo tanto, cada uno de los coeficientes $\binom{p}{i}$ son cero en F y se tiene que:

$$(a+b)^p = a^p + b^p.$$

■

2.5.4. Existencia y unicidad de campos finitos

Teorema 2.63. *Si un campo con $q = p^m$ existe, entonces este es único salvo isomorfismos. Se denota a este campo $\mathbb{F}_q$.*

Demostración. Por el teorema 2.61, cualquier campo finito debe tener p^j elementos y que cualquier elemento distinto de cero es una raíz del polinomio $X^{q-1} - 1$ el cual tiene $q-1$ raíces. Por lo que los elementos del campo son precisamente las $q-1$ raíces del polinomio junto al 0. Ahora, sea a un elemento generador en F y sea $M(X)$ el polinomio de grado mínimo con coeficientes en $\mathbb{F}_p$ tal que $M(a) = 0$,

se quiere demostrar que todas las raíces de $M(X)$ satisfacen a $X^{q-1}-1$. Se utiliza el teorema 2.53 para obtener que $X^{q-1}-1 = M(X)G(X)+R(X)$ donde $grado(R) < grado(M)$. Sigue que

$$0 = a^{q-1}-1 = M(a)G(a)+R(a)$$

por lo que $R=0$, pues $M(X)$ es el polinomio de menor grado tal que evaluado en a nos da cero. Por lo que todas las raíces de $M(X)$ satisfacen

$$X^{q-1}-1.$$

Sea F' otro campo finito con q elementos y sea b un generador de F', todos los elementos distintos de cero en F' son raíces del polinomio $X^{q-1}-1$. Como $M(X) \mid X^{q-1}-1$ alguna b^i debe ser una raíz de $M(X)$ por lo que el mapeo

$$\pi(a) \to b^i$$

induce un isomorfismo entre F y F'. ■

Teorema 2.64. *Para cualquier primo p y un entero positivo j, existe un único, salvo isomorfismos, campo de p^j elementos, $\mathbb{F}_{p^j}$.*

Demostración. Se ha demostrado que si el campo existe, este es único salvo isomorfismos. La existencia de polinomios irreducibles de grado j sobre $\mathbb{F}_p$ hace posible construir el campo.

■

Capítulo 3

Criptografía y el problema del logaritmo discreto

En este capitulo se definirá el problema del logaritmo discreto y se presentarán dos sistemas de cifrado que basan su seguridad en la dificultad de resolver este problema. Finalmente, se introducirán los algoritmos más utilizados para resolver el problema del logaritmo discreto al igual que sus diferencias y propiedades.

3.1. Criptografía simétrica y asimétrica

La criptografía es el estudio de sistemas matemáticos para resolver dos tipos de problemas de seguridad: privacidad y autenticación. Uno de estos problemas de privacidad es el intercambio de información o mensajes entre dos personas.

3.1.1. Criptografía simétrica

Si Alice quiere enviar un mensaje a Bob y ambos tienen un secreto en común entonces es posible realizar el intercambio utilizando un **sistema criptográfico simétrico**. Esto es un sistema que utiliza la misma **llave** para **encriptar** el **mensaje** o **texto plano** y para **desencriptar** el **criptograma** o **texto cifrado**.

Una forma de entender como funciona un sistema criptográfico simétrico es como sigue: Imagine que Alice tiene una caja fuerte, la cual solo se abre con una llave. Alice obtiene una copia de esta llave y se la da a Bob. Cuando Alice desea enviar un mensaje a Bob lo deposita dentro de la caja fuerte y se la envía a Bob. Cuando Bob recibe la caja fuerte la abre utilizando su copia de la llave y lee el mensaje. En este esquema solo Alice y Bob pueden abrir la caja fuerte pues son los únicos que tienen la llave. Este ejemplo es ilustrado en la figura 3.1.

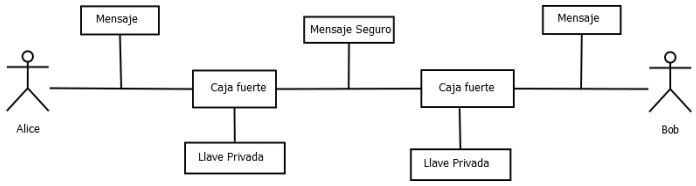

Figura 3.1: Intercambio de mensajes utilizando una caja fuerte.

Definición 3.1. Un **sistema criptográfico simétrico** consiste de un conjunto

$$(\mathcal{K}, \mathcal{M}, \mathcal{C}, e, d)$$

donde $\mathcal{K}, \mathcal{M}, \mathcal{C}$ representan a todas las posibles llaves, mensajes y criptogramas respectivamente. e, es la función de encriptación, y d, la función de desencriptación, definidas como sigue:

$$e : \mathcal{K} \times \mathcal{M} \to \mathcal{C} \qquad d : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$$

y cumplen que

$$d(k, e(k, m)) = m$$

para toda $k \in \mathcal{K}$ y $m \in \mathcal{M}$.

En algunos casos se usará la notación e_k y d_k para referirse a las funciones de encriptación y desencriptación que utilizan la llave k. Se tiene que para cada llave $k \in \mathcal{K}$, d_k es la función inversa de e_k. Esta función debe ser inyectiva, pues de esta forma a cada mensaje le corresponde un único criptograma bajo una llave k y es posible desencriptar este para obtener de nuevo, de forma única, el mensaje.

Algunas propiedades de un sistema criptográfico exitoso son las siguientes:

1. Para toda $k \in \mathcal{K}$ y $m \in \mathcal{M}$, debe ser fácil calcular el criptograma $e(k, m)$.
2. Para toda $k \in \mathcal{K}$ y $c \in \mathcal{C}$, debe ser fácil calcular el mensaje $d(k, c)$.
3. Dado un conjunto de criptogramas $c_1, c_2, \ldots, c_n \in \mathcal{C}$, los cuales fueron encriptados con una llave $k \in \mathcal{K}$, debe ser difícil de obtener cualquiera de los mensajes $d(k, c_1), d(k, c_2), \ldots, d(k, c_n)$ sin conocer k.

Se debe asumir que un intruso que trata de obtener un mensaje encriptado, conoce el mecanismo por el cual funciona la pareja e, d. La seguridad del sistema debe depender únicamente de la llave escogida. A esta propiedad se le conoce como el **principio de Kerckhoff**.

Ejemplo 3.2. El cifrado afín es un tipo de cifrado por sustitución en el que a cada símbolo del alfabeto en claro se le asigna uno del alfabeto cifrado. La llave

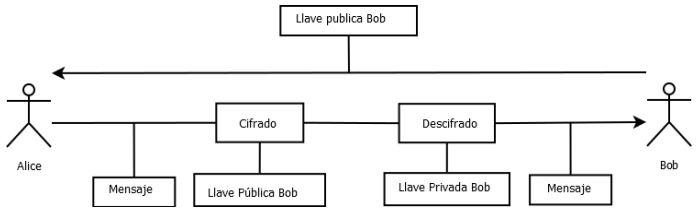

Figura 3.2: Sistema criptográfico de llave pública.

en el cifrado afín consiste de dos enteros $k = (k_1, k_2)$ y funciones de encriptación y desencriptación definidas como sigue:

$$e_k(m) = k_1 m + k_2 (\text{mód } p)$$
$$d_k(c) = k_1^{-1}(c - k_2)(\text{mód } p),$$

donde p es un número primo y k_1^{-1} es el inverso de k_1 módulo p. Antes de poder usar el cifrado afín debe ser posible asignar a cada símbolo del alfabeto en claro un valor numérico.

3.1.2. El intercambio de llaves

Uno de los problemas de los sistemas criptográficos simétricos, es el **intercambio de llaves**. Si Alice quiere enviarle un mensaje a Bob, pero no hay forma segura de reunirse con él para el intercambio de llaves, entonces no hay forma en que puedan intercambiar mensajes encriptados. Los **sistemas criptográficos asimétricos** tienen entre sus objetivos dar solución a este problema.

3.1.3. Criptografía asimétrica

Una llave en un sistema criptográfico asimétrico o de **llave pública** es en realidad dos llaves, una llave pública, la cual es visible para cualquiera que quiera utilizarla y una llave privada, la cual es secreta. Si Alice quiere enviar un mensaje a Bob, ella lo encripta utilizando la llave pública de Bob, una vez recibido el mensaje Bob utiliza su llave privada para desencriptar el mensaje. Este esquema esta ilustrado en la figura 3.2.

Utilizando una analogía similar a la de la caja fuerte, se puede ver a un sistema criptográfico de llave pública como un buzón de correos, cualquier persona puede depositar una carta, pero una vez dentro del buzón solo la persona que tenga la llave de este, puede recuperar los mensajes.

Definición 3.3. Un **sistema criptográfico de llave pública** consiste de un conjunto

$$(\mathcal{K}, \mathcal{M}, \mathcal{C}, e, d)$$

donde $\mathcal{K}, \mathcal{M}, \mathcal{C}$ representan a todas las posibles llaves, mensajes y criptogramas respectivamente, pero donde un elemento $k \in \mathcal{K}$ consiste de un par

$$k = (k_{priv}, k_{pub})$$

donde k_{priv} es la llave privada y k_{pub} es la llave pública. Las funciones de encriptación, e, y desencriptación, d, están definidas como sigue:

$$e_{k_{pub}} : \mathcal{M} \to \mathcal{C} \qquad d_{k_{priv}} : \mathcal{C} \to \mathcal{M}$$

y cumplen que

$$d_{k_{priv}}(e_{k_{pub}}(m)) = m$$

para toda $m \in \mathcal{M}$.

Para que un sistema criptográfico asimétrico pueda ser considerado seguro, debe ser difícil para una intrusa Eve obtener el mensaje a partir de un criptograma, aun si ella conoce los detalles acerca de la llave pública.

3.2. Intercambio de llaves de Diffie-Hellman

En 1976 con un revolucionario artículo de investigación titulado *New Directions in Cryptography*. Whitfield Diffie y Martin Hellman mostraron una forma de obtener una llave privada compartida entre dos partes. Posteriormente esta llave podría ser usada en un sistema criptográfico de llave privada para el intercambio seguro de mensajes. Cabe aclarar que el intercambio de llaves de Diffie-Hellman no permite la encriptación y desencriptación de mensajes, sino una forma de compartir una llave privada.

Como se muestra en la tabla 3.1, el sistema de intercambio de llaves Diffie-Hellman consta de las siguientes partes, la parte preliminar, donde se determinan los parámetros a utilizar en el algoritmo, la creación de los parámetros públicos y privados, aquí ambas partes calculan las llaves privadas, k_{priv}, y públicas, k_{pub}, la parte del intercambio de llaves públicas, donde se realiza el intercambio de llaves entre ambas partes y finalmente la creación del secreto compartido, el cual funcionaría como la llave privada.

La dificultad para Eve, consiste en encontrar los valores a, b a partir de los valores públicos p, g, A, B. En realidad Eve no necesita conocer a, b, es decir podría encontrar el secreto compartido si los obtiene, pero en realidad el problema que enfrenta es el siguiente:

Definición 3.4. Sea p un primo y g un entero. El **Problema Diffie-Hellman** consiste en encontrar el valor de $g^{ab} (\text{mód } p)$, si se conocen los valores $g^a (\text{mód } p)$ y $g^b (\text{mód } p)$.

Ejemplo 3.5. Alice y Bob acuerdan utilizar los parámetros $p = 179$ y $g = 30$.

Parte preliminar	
Un tercero en confianza escoge un número primo grande p y un entero $g \in \{2, 3, \ldots, p-2\}$. Se publican p y g.	
Creación de parámetros públicos y privados	
Alice	Bob
Calcula $k_{privA} = a \in \{1, 2, \ldots, p-1\}$ Calcula $k_{pubA} = A = g^a (\text{mod } p)$	Calcula $k_{privB} = b \in \{1, 2, \ldots, p-1\}$ Calcula $k_{pubB} = B = g^b (\text{mod } p)$
Intercambio de llaves	
Alice	Bob
Envia el parametro A a Bob	Envía el parámetro B a Alice
Creación del secreto compartido	
Alice	Bob
Calcula $B^a (\text{mod } p)$.	Calcula $A^b (\text{mod } p)$.
El secreto compartido es $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b (\text{mod } p)$.	

Tabla 3.1: Intercambio de llaves Diffie-Hellman.

Cada uno de ellos calcula sus respectivas llaves públicas y privadas. Para Alice: $k_{privA} = 101, k_{pubA} = 30^{101} (\text{mód } p) = 127$ y para Bob: $k_{privB} = 84, k_{pubB} = 30^{84} (\text{mód } p) = 100$.

Observe que después de realizar el intercambio de las llaves públicas, k_{pubA}, k_{pubB}, ambos Alice y Bob pueden calcular el secreto compartido

$$B^a \equiv g^{ab} \equiv A^b \equiv 70 (\text{mód } p)$$

3.3. Sistema de cifrado ElGamal

El sistema de cifrado ElGamal, presentado en la tabla 3.2, fue publicado en 1985 por Taher ElGamal y esta basado en las ideas presentadas en el intercambio de llaves Diffie-Hellman. ElGamal es un sistema de encriptación de llave pública que nos permite intercambiar mensajes.

La dificultad para Eve, es calcular la llave privada de Alice, a, conociendo los valores p, g y la llave pública $A = g^a$.

Ejemplo 3.6. Alice utiliza el primo $p = 503$ y el elemento generador $g = 5$. Alice escoge $a = 231$ como su llave privada y calcula su llave pública $A \equiv g^a \equiv 5^{231} \equiv 425 (\text{mód } 503)$.

Bob decide enviar el mensaje $m = 155$. Escoge una llave de sesión $k = 334$ y calcula $c_1 \equiv g^k \equiv 5^{334} \equiv 177 (\text{mód } 503)$ y $c_2 \equiv mA^k \equiv 155 \cdot 425^{334} \equiv 118 (\text{mód } 503)$. Bob envía (c_1, c_2) a Alice.

Alice calcula $x \equiv c_1^a \equiv 177^{231} \equiv 176$ y luego $x^{-1} \equiv 483 (\text{mód } 503)$. Una vez que

Parte preliminar	
Un tercero en confianza escoge un número primo grande p un elemento g, generador en $(\mathbb{Z}/p\mathbb{Z})^{\times}$. Se publican p y g.	
Creación de parámetros públicos y privados	
Alice	Bob
Calcula $k_{privA} = a \in \{1, 2, \ldots, p-1\}$ Calcula $k_{pubA} = A = g^a (\text{mod } p)$ pública A	
Encriptación del mensaje	
Alice	Bob
	Escoge el mensaje m Calcula una llave de sesión k Utiliza A para calcular $c_1 = g^k (\text{mód } p)$ y $c_2 = mA^k (\text{mód } p)$ Envía (c_1, c_2) a Alice.
Desencriptación del mensaje	
Alice	Bob
Calcula $(c_1^a)^{-1} c_2 (\text{mód } p) = m$.	

Tabla 3.2: Sistema de cifrado ElGamal.

tiene estos valores puede recuperar el mensaje m calculando $c_2 x^{-1} \equiv 118 \cdot 483 \equiv 155 (\text{mód } 503)$.

En el sistema de cifrado ElGamal el mensaje es un entero entre 2 y $p-1$, mientras que la cifra consiste de dos valores c_1 y c_2 en el mismo rango. En general toma el doble de espacio escribir la cifra que el mensaje. Decimos que ElGamal tiene una expansión de mensaje 2 a 1.

La seguridad de ambos ElGamal y el intercambio de llaves Diffie-Hellman, consiste en resolver una o más ecuaciones de tipo $A = g^a (\text{mód } p)$ donde todos los valores menos a son conocidos. Esto se conoce como el **problema del logaritmo discreto** que será discutido en la sección 3.4, a continuación.

3.4. El problema del logaritmo discreto

Definición 3.7. El problema del logaritmo discreto (PLD) sobre un grupo finito cíclico $\mathbb{F}_p^{\times}$ esta basado en dos números g y h y un número secreto x el cual resuelve la siguiente congruencia:

$$h \equiv g^x \pmod{\text{p}}.$$

Resolver el PLD consiste en encontrar el número secreto x, cuando se conocen g y h.

El entero más pequeño m que satisfaga $h = g^m$ es llamado el logaritmo de h con respecto a g, y es denotado por:

$$m = \log_g(h).$$

Si existe una solución para el problema del logaritmo discreto, entonces existen infinitas de ellas esto debido al **teorema pequeño de Fermat**, teorema 2.19, que nos dice que $g^{p-1} \equiv 1 (\text{mód } p)$. Por lo tanto si x es solución, sigue que $x + k(p-1)$ es una solución para cualquier valor de k, esto pues

$$g^{x+k(p-1)} = g^x (g^{(p-1)})^k \equiv h \cdot 1^k \equiv h (\text{mód } p).$$

Por lo tanto se define el logaritmo módulo $p-1$.

$$\log_g : \mathbb{F}_p^\times \to \mathbb{Z}/(p-1)\mathbb{Z}.$$

A pesar de que en la definición 3.7 el **problema del logaritmo discreto** está definido para $\mathbb{F}_p^\times$ es posible dar una definición más general.

Definición 3.8. Sea $(G, \cdot)$ un grupo multiplicativo y $g \in G$ un elemento de orden n entonces para un elemento $h \in \langle g \rangle$ el **problema del logaritmo discreto** consiste en encontrar el único entero $x, 0 \leq x \leq n-1$ tal que

$$h = g^x.$$

Esta x es llamada el logaritmo de h con respecto a g y es denotada por

$$\log_g(h).$$

3.5. Resolviendo el problema del logaritmo discreto

En esta sección se asume que $(G, \cdot)$ es un grupo multiplicativo y $g \in G$ es un elemento de orden n. Por lo tanto el problema del logaritmo discreto consiste en encontrar el único entero $x, 0 \leq x \leq n-1$ tal que $h = g^x$.

3.5.1. Fuerza Bruta o Búsqueda exhaustiva

El algoritmo más sencillo para resolver el problema del logaritmo discreto, es simplemente calcular de manera secuencial $g, g^2, g^3, \ldots$ hasta dar con la solución. Cada paso es calculado multiplicando el elemento anterior de la sucesión por g. Este método requiere de a lo más $n-2$ pasos y n comparaciones.

3.5.2. Paso de bebé, paso de gigante

También conocido como el **algoritmo de Shanks**, este algoritmo busca una colisión realizando comparaciones con una lista de valores pre-calculados.

1. Sea $m = 1 + \lfloor \sqrt{n} \rfloor$, en particular $m > \sqrt{n}$.
2. Creamos dos listas.

$$\text{Lista 1: } e, g, g^2, g^3, \ldots, g^m$$
$$\text{Lista 2: } h, hg^{-m}, hg^{-2m}, hg^{-3m}, \ldots, hg^{-m^2}$$

3. Hallamos una coincidencia entre las dos listas, digamos $g^i = hg^{-jm}$.
4. Entonces $x = i + jm$ es una solución de $g^x = h$.

Teorema 3.9. *El **algoritmo de Shanks** resuelve el problema del logaritmo discreto en $\mathcal{O}(\sqrt{n} \cdot \log n)$ pasos.*

Demostración. Crear ambas listas toma aproximadamente $2m$ multiplicaciones, pues para calcular el elemento que sigue en la lista, multiplicamos el anterior por g o g^{-m}. Asumiendo que existe una colisión entonces es posible hallarla utilizando un algoritmo de ordenamiento y búsqueda en $\log m$ pasos. Por lo que el tiempo total del algoritmo es de $\mathcal{O}(m \log m) = \mathcal{O}(\sqrt{n} \log n)$ esto pues

$$m \log m \approx \sqrt{n} \log \sqrt{n} = \frac{1}{2}\sqrt{n} \log n.$$

Veamos ahora que siempre es posible encontrar una colisión en las listas. Sea x la solución para $g^x = h$, podemos escribir a x como $x = mq + r$ con $0 \leq r < m$. sabemos que $1 \leq x < n$, por lo tanto sigue que

$$q = \frac{x - r}{m} < \frac{n}{m} < m.$$

Esto pues $m > \sqrt{n}$. De aquí sigue que se puede escribir $g^x = h$ como sigue

$$g^r = h \cdot g^{-qm},$$

con $0 \leq r < m$ y $0 \leq q < m$. Esto significa que g^r se encuentra en la lista 2. Lo que demuestra que existe un elemento en común entre ambas listas.

■

3.6. La paradoja del cumpleaños

El nombre de **paradoja del cumpleaños** viene del hecho de que mucha gente se sorprende de que en un grupo de más de 23 personas, la probabilidad de que 2 de ellas tengan la misma fecha de cumpleaños es mayor que el 50 %.

Teorema 3.10. *Sea S un conjunto de N elementos. Si estos elementos son escogidos uniformemente al azar, esto es, con la misma probabilidad para cada elemento, entonces el número de muestras a escoger antes de que un elemento aparezca dos veces es menor a $\sqrt{\frac{\pi N}{2}} + 2 \approx 1.253\sqrt{N}$. El elemento que aparece dos veces es conocido como una **colisión**.*

Demostración. Sea X una variable aleatoria que nos da el número de elementos seleccionados uniformemente al azar de S antes de que una colisión aparezca. Después de que l elementos han sido seleccionados la probabilidad de que el siguiente elemento sea distinto de los anteriores es $(1 - l/N)$. Por lo tanto la probabilidad $\Pr(X > l)$, para un conjunto N de elementos, esta dada por:

$$P_{N,l} = \left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\cdots\left(1 - \frac{(l-1)}{N}\right).$$

Ahora como $1 - x \leq e^{-x}$ para $x \geq 0$ tenemos que

$$\begin{aligned} P_{N,l} = e^{-\frac{1}{N}} e^{-\frac{2}{N}} \cdots e^{-\frac{(l-1)}{N}} &= e^{-\sum_{j=0}^{l-1} \frac{j}{N}} \\ &= e^{\frac{-\frac{1}{2}(l-1)l}{N}} \\ &= e^{\frac{-(l-1)^2}{2N}}. \end{aligned}$$

Por definición, el valor esperado de X es

$$\begin{aligned} \sum_{l=1}^{\infty} l\Pr(X = l) &= \sum_{l=1}^{\infty} l(\Pr(X > l-1) - \Pr(X > l)) \\ &= \sum_{l=0}^{\infty} (l + 1 - l)\Pr(X > l) \\ &= \sum_{l=0}^{\infty} \Pr(X > l) \\ &\leq 1 + \sum_{l=1}^{\infty} e^{\frac{-(l-1)^2}{2N}}. \end{aligned}$$

Podemos estimar esta suma utilizando la siguiente integral

$$1 + \int_0^{\infty} e^{-x^2/2N} dx.$$

Ahora como $e^{-x^2/2N}$ es monótona decreciente y toma valores entre 0 y 1 entonces la diferencia entre el valor de la suma y el valor de la integral es a lo más 1. Haciendo un cambio de variable $u = x/\sqrt{2N}$ nos da

$$\sqrt{2N} \int_0^{\infty} e^{-u^2} du$$

esta integral tiene como resultado $\sqrt{\pi/2}$. Por lo tanto el valor esperado para X es menor o igual a $\sqrt{\pi N/2} + 2$.

■

Ejemplo 3.11. ¿Si tenemos 23 personas juntas en una habitación cuál sería la probabilidad de que dos de ellas compartan la misma fecha de cumpleaños?

La probabilidad de que 23 personas tengan cumpleaños distintos esta dada por

$$\begin{aligned} P_{365,23} &= \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\cdots\left(1 - \frac{(23-1)}{365}\right) \\ &= \frac{365!}{365^{23}(365-23)!} \\ &\approx 0.4927. \end{aligned}$$

Entonces tenemos que la probabilidad de que haya una colisión, es decir de que dos personas compartan la misma fecha de cumpleaños es de $1-P_{365,23} \approx 0.5072$. Notemos que 23 es menor a $\sqrt{\frac{\pi 365}{2}}+2 = 25.9445\ldots$, esto, de acuerdo al teorema 3.10.

3.7. La ρ de Pollard

Sea S un conjunto finito. Sea:

$$f : S \to S.$$

Una función que revuelve los elementos de S de forma similar a como lo haría una función aleatoria. Utilizando f podemos construir la siguiente sucesión:

$$x_0 = x, x_1 = f(x_0), x_2 = f(x_1), x_3 = f(x_2)\ldots$$

En otras palabras

$$x_i = \underbrace{(f \circ f \circ f \ldots \circ f)}_{i \text{ veces}}(x).$$

A esta sucesión se le conoce como la **órbita** hacia adelante de x con el mapeo f y es denotada como $O_f^+(x)$.

Como el conjunto S es finito, tarde o temprano aparecerá un valor repetido. Denominamos a T como el mayor entero tal que x_{T-1} aparezca una sola vez en $O_f^+(x)$ y como M al entero más pequeño tal que $x_{T+M} = x_T$. Si vemos a los valores x_i como puntos en una gráfica, observaremos que hay un ciclo a partir del valor x_T, el primer valor repetido, esto da origen a un patrón parecido a la letra griega ρ como muestra la figura 3.3, donde $T = 4$ y $M = 7$.

Suponga que S contiene N elementos. La cantidad $T+M$ usualmente no es más que algún múltiplo pequeño de $\sqrt{N}$. Como $x_T = x_T + M$ esto quiere decir que obtenemos una colisión en $\mathcal{O}(\sqrt{N})$ pasos.

Como no se conocen los valores de T parecería que tenemos que guardar todos los valores para poder detectar una colisión, sin embargo es posible encontrar la colisión usando algoritmos de detección de ciclos, sin necesidad de guardar todos los valores. La idea es calcular otra secuencia adicional y_i como sigue

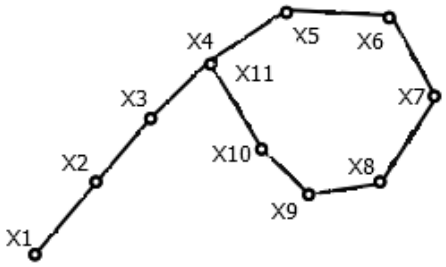

Figura 3.3: Una órbita con la forma de la letra griega ρ.

$$y_0 = x_0, \quad \text{y} \quad y_{i+1} = f(f(y_i)).$$

Esto para toda i. En otras palabras $y_i = x_{2i}$. Ahora ¿cuánto tiempo toma encontrar i tal que $x_{2i} = x_i$?

En general tenemos que para $j > i$

$$x_j = x_i \iff i \geq T \text{ y } j \equiv i(\text{mód } M),$$

obtenemos $x_j = x_i$ cuando ya nos pasamos de x_T y $j - i$ es un múltiplo de M.

Por lo tanto $x_{2i} = x_i$ si y solo si $i \geq T$ y $2i \equiv i(\text{mód } M)$. Se sigue que $2i \equiv i(\text{mód } M) = M \mid 2i - i$ y de aquí $M|i$. Es decir, obtenemos la colisión exactamente cuando i es igual al primer múltiplo de M. Como alguno de los números $T, T+1, \ldots, T+M-1$ es divisible por M esto prueba que $x_{2i} = x_i$ para alguna $1 \leq i < T + M$. Esta es la base del **algoritmo de detección de ciclos de Floyd**, también conocido como la tortuga y la liebre, y está descrito en el algoritmo 1.

El algoritmo de detección de ciclos de Floyd es uno de los algoritmos más conocidos para la detección de ciclos pero no es el único, un ejemplo de un algoritmo que utiliza una idea similar, pero con un mejor desempeño, es el algoritmo de **Brent**, creado por Richard P. Brent. Este algoritmo hace que la liebre camine una cantidad de pasos la cual se va incrementando en cada intento en el que no encuentre a la tortuga, cambiando la posición de esta a la inicial de la liebre se garantiza que si hay un ciclo se encuentre cuando el número de pasos que la liebre camina sea mayor a ambos M y T. Una ventaja de este algoritmo sobre el de Floyd es que encuentra el tamaño del ciclo de forma directa en vez de tener que realizar una cuenta posterior, ademas de detectar ciclos con una mayor velocidad.

Para un grupo $G = \langle g \rangle$ con orden primo n. El problema del logaritmo discreto dado por $h = g^a, h \in \langle g \rangle$, puede ser resuelto utilizando sucesiones pseudoaleatorias. Primero consideremos que si hallamos $a_i, b_i, a_j, b_j \in \mathbb{Z}/n\mathbb{Z}$ tales que

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$$

Algoritmo 1 Algoritmo de detección de ciclos de Floyd

Input: f, x_0
Output: T, M

```
tortuga = f(x_0).
liebre = f(f(x_0)).
while tortuga distinto a liebre do
    tortuga = f(tortuga).
    liebre = f(f(liebre)).
end while
T = 0.
tortuga = x_0
while tortuga distinto a liebre do
    tortuga = f(tortuga).
    liebre = f(liebre).
    T = T + 1
end while
M = 1
liebre = f(tortuga)
while tortuga distinto a liebre do
    liebre = f(liebre).
    M = M + 1
end while
return T, M
```

con $b_i \not\equiv b_j (\text{mód } n)$ Entonces podemos resolver el problema del logaritmo discreto como sigue

$$h = g^{(a_i - a_j)(b_j - b_i)^{-1} (\text{mód } n)}.$$

La idea es calcular una sucesión de elementos $x_i = g^{a_i} h^{b_i} \in G$ utilizando una función f, como fue descrita al inicio de esta sección. Esta sucesión recibe el nombre de **caminata determinista pseudoaleatoria**. Utilizando la paradoja del cumpleaños, teorema 3.10, es posible dar una cota superior al valor esperado por una colisión el cual es de $\sqrt{\pi n/2} + 2$.

3.7.1. Caminatas pseudoaleatorias

El algoritmo ρ de Pollard simula una función aleatoria de G a G como sigue:

1. Particiona G en n_s subconjuntos disjuntos, usualmente del mismo tamaño.
2. $G = S_0 \cup S_1 \cup \ldots \cup S_{n_s - 1}$.

El subconjunto S_i es definido por una función que selecciona a los elementos $S : G \to \{0, 1, \ldots, n_s - 1\}$ de aquí se tiene que $S_i = \{g \in G \mid S(g) = i\}$.

Definición 3.12. Las **caminatas** ρ están definidas como sigue: primero se precalculan los valores $g_j = g^{u_j} h^{v_j}$ para $0 \leq j \leq n_s - 1$ donde $0 \leq u_j, v_j < n$

son escogidos uniformemente al azar. Sea $x_1 = g$. La caminata original ρ esta definida:

$$x_{i+1} = f(x_i) = \begin{cases} x_i^2 & \text{si } S(x_i) = 0 \\ x_i g_j & \text{si } S(x_i) = j, j \in \{1, \ldots, n_s - 1\} \end{cases}$$

y la caminata aditiva esta definida como sigue:

$$x_{i+1} = f(x_i) = x_i g_{s(x_i)}.$$

Una característica importante de estas caminatas es que cada paso requiere una única operación, una vez que los valores iniciales han sido seleccionados la caminata es determinista y cada valor depende únicamente del anterior.

Para poder encontrar la colisión es necesario poder realizar la descomposición

$$x_i = g^{a_i} h^{b_i}.$$

Los valores $a_i, b_i \in \mathbb{Z}/n\mathbb{Z}$ son obtenidos al poner como valores iniciales $a_1 = 1, b_0 = 0$ y hallar los valores siguientes como sigue

$$a_{i+1} = \begin{cases} 2a_i (\text{mód } n) & \text{si } S(x_i) = 0 \\ a_i + u_{s(x_i)} (\text{mód } n) & \text{si } S(x_i) > 0 \end{cases}$$

$$b_{i+1} = \begin{cases} 2b_i (\text{mód } n) & \text{si } S(x_i) = 0 \\ b_i + v_{s(x_i)} (\text{mód } n) & \text{si } S(x_i) > 0 \end{cases}$$

De esta forma tenemos que

$$(x_{i+1}, a_{i+1}, b_{i+1}) = \text{walk}(x_i, a_i, b_i).$$

Pero es importante recalcar que x_{i+1} depende únicamente de x_i.

3.7.2. ρ de Pollard utilizando el algoritmo de Floyd

Es posible utilizar caminatas ρ junto al algoritmo de Floyd para encontrar una colisión. A diferencia del algoritmo 1, en vez de buscar los valores T, M nos interesa el momento en que se encuentra la colisión para poder encontrar la solución dados los valores encontrados por la caminata ρ. Este proceso está descrito en el algoritmo 2.

3.7.3. Puntos distinguidos

Definición 3.13. Un elemento $g \in G$ es un punto distinguido si su representación binaria $b(g)$ satisface alguna propiedad fácilmente verificable. Denotamos por $D \subset G$ al conjunto de elementos distinguidos. La probabilidad $\#D/\#G$ de que un elemento del grupo, escogido uniformemente al azar, sea distinguido es denotada por θ.

Algoritmo 2 Algoritmo de detección de ciclos de Floyd

Input: $g, h \in G$, n orden del grupo.
Output: a tal que $h = g^a$ o $\perp$

```
Escoger una caminata ρ la que se denominara walk.
x_1 = g, a_1 = 1, b_1 = 0
(x_2, a_2, b_2) = walk(x_1, a_1, b_1)
while x_1 distinto a x_2 do
    (x_1, a_1, b_1) = walk(x_1, a_1, b_1)
    (x_2, a_2, b_2) = walk(walk(x_1, a_1, b_1))
end while
if b_1 ≡ b_2(mód n) then
    return ⊥
end if
return (a_2 − a_1)(b_2 − b_1)^{-1}(mód n)
```

Ejemplo 3.14. Los puntos distinguidos en una curva elíptica definidos como sigue:

$$D = \{P = (x_p, y_p) \in E(\mathbb{F}_p) \mid H(x_p) \equiv 0(\text{mód } 2^{n_d}), 0 \leq x_p < p\},$$

donde H es una función hash cuya salida es un entero mayor o igual a 0. Esto es, los puntos cuyo resultado de aplicar la función H a la coordenada x tienen los últimos n_d dígitos iguales a 0 cuando se ven en su representación binaria. Más información acerca de curvas elípticas se presenta en el capitulo 4.

3.7.4. El algoritmo ρ de Pollard distribuido

En este apartado se presentará un modelo basado en cómputo distribuido para resolver el problema del logaritmo discreto utilizando el algoritmo ρ de Pollard con puntos distinguidos. Sea S el servidor donde se guardan los puntos distinguidos encontrados y N_p clientes los cuales pueden intercambiar mensajes con el servidor y viceversa.

Para resolver el problema del logaritmo discreto el servidor asignará los datos iniciales a los clientes y estos correrán la caminata ρ y enviarán información de vuelta al servidor cuando un punto distinguido sea encontrado, eventualmente el servidor tendrá la información necesaria para poder resolver el PLD. En otras palabras habrá encontrado una colisión. La funcionalidad del servidor y de los clientes se presenta en los algoritmos 3 y 4 respectivamente.

Lo más que se puede esperar con el uso de este modelo distribuido comparado con el caso no distribuido es un incremento lineal. En otras palabras con N_p clientes se espera alcanzar un tiempo de ejecución proporcional a $\sqrt{\frac{n}{N_p}}$, donde n es el orden del grupo sobre el cual se trabaja.

Algoritmo 3 Servidor

Input: $g, h \in G$, n orden del grupo.
Output: x tal que $h = g^x$

```
Selecciona una función walk(x, a, b).
Inicializa una base de datos L.
Inicializa todos los clientes con la función walk(x, a, b).
while DLP no resuelto do
    Recibe la terna (x, a, b) de los clientes y la inserta en L.
    if La primera coordenada (x, a, b) coincide con alguna preexistente
(x, a', b') then
        if b' ≠ b(mód n) then
            Termina todos los clientes.
            return (a − a')(b − b')^{−1}(mód n)
        end if
    end if
end while
```

Algoritmo 4 Cliente

Input: $g, h \in G$, n orden del grupo, función walk(x, a, b).

```
while DLP no resuelto do
    Escoge al azar 0 ≤ a, b < n
    Sea x = g^a h^b
    while x no sea un punto distinguido do
        (x, a, b) = walk(x, a, b)
    end while
    Envía (x, a, b) al servidor.
end while
```

Capítulo 4

Curvas elípticas y el problema del logaritmo discreto

A continuación, hacemos un breve repaso a los principales resultados y definiciones sobre curvas elípticas

4.1. Curvas Elípticas

Sea K un campo con característica distinta de 2 o 3, decimos de manera informal que una **curva elíptica** sobre K es el conjunto de soluciones a la siguiente ecuación:

$$Y^2 = X^3 + AX + B, \quad A, B \in K. \tag{4.1}$$

Ejemplo 4.1. La figura 4.1 muestra la gráfica para la curva elíptica $Y^2 = X^3 - X$ sobre el campo de los números racionales, mientras que la figura 4.2 muestra la gráfica para la curva $Y^2 = X^3 + X$ sobre el campo finito $\mathbb{F}_{347}$.

Una característica de estas curvas es que es posible asociarles un grupo abeliano utilizando los puntos sobre ellas.

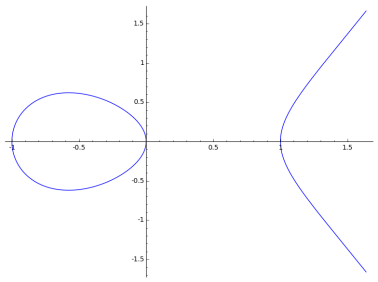

Figura 4.1: Curva elíptica $Y^2 = X^3 - X$ sobre $\mathbb{Q}$.

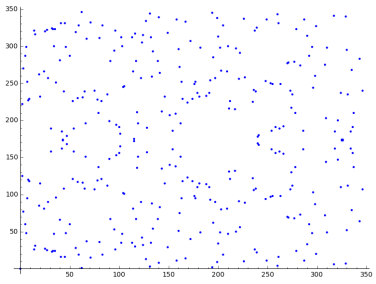

Figura 4.2: Curva elíptica $Y^2 = X^3 + X$ sobre $\mathbb{F}_{347}$.

A simple vista esto no es completamente claro. Podemos hacer uso de la geometría para ver cómo los puntos sobre una curva forman un grupo. Para esto sean P, Q puntos distintos sobre la curva elíptica E, como se muestra en la figura 4.3. Realizamos el procedimiento de sumar ambos puntos de la forma siguiente:

1. Hallamos la línea L que pasa por P y Q.
2. Hallamos el punto R en el cual la línea interseca a la curva en un tercer punto.
3. El punto $P + Q$ es la reflexión de R sobre el eje x.

Este caso esta representado en la figura 4.3. Pero ¿cómo sumamos un punto P a si mismo, dado que hay muchas líneas distintas que pasan por P? La respuesta es utilizar la recta tangente a la curva elíptica en P. Luego continuamos con los pasos 2 y 3 del procedimiento anterior. Este caso se ilustra en la figura 4.4. Sea P un punto sobre una curva elíptica. Llamamos a la reflexión de este sobre el eje x como $-P$. Podemos observar que al intentar realizar la suma $P + (-P)$ tenemos un problema pues la línea vertical sobre estos dos puntos no corta a la curva en algún otro punto.

Ya que no existe un punto en el plano que funcione para realizar la suma, se crea uno adicional. El punto **infinito** $\mathcal{O}$, con la propiedad de estar en toda línea vertical. Llegamos a la siguiente definición formal de una curva elíptica:

Definición 4.2. Sea K un campo con característica distinta de 2 o 3 una curva elíptica se puede definir como el conjunto de soluciones a una ecuación de la forma:

$$Y^2 = X^3 + AX + B, \quad A, B \in K,$$

junto con un punto **infinito** $\mathcal{O}$ y donde se requiere que el discriminante, Δ, sea:

$$\Delta = 4A^3 + 27B^2 \neq 0.$$

Lo cual nos dice que el polinomio $X^3 + AX + B$ tiene raíces distintas y por lo tanto, que la curva es no singular.

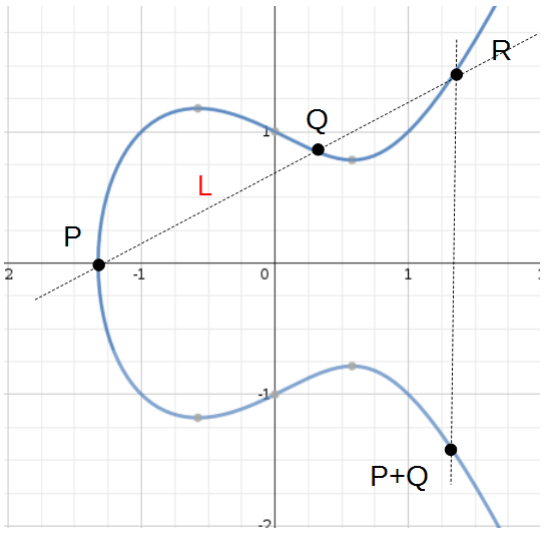

Figura 4.3: Suma de dos puntos.

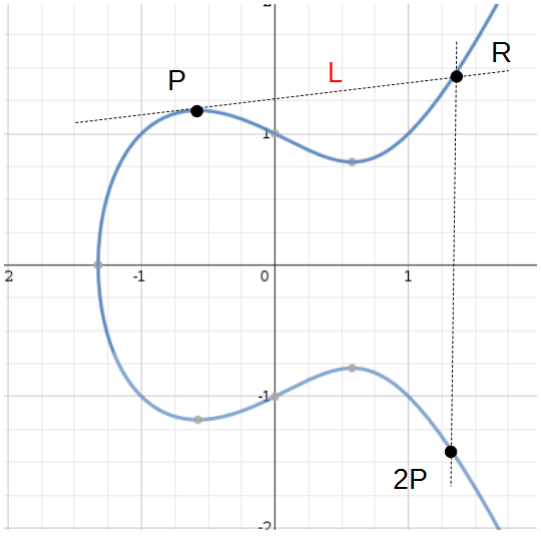

Figura 4.4: Duplicación de un punto.

Si el campo K tiene característica igual a 2 entonces una curva elíptica es el conjunto de soluciones a una ecuación de la forma:

$$Y^2 + CY = X^3 + AX + B$$

o a una ecuación de la forma:

$$Y^2 + XY = X^3 + AX^2 + B,$$

junto con el punto infinito $\mathcal{O}$, en este caso no se requiere que la cúbica al lado derecho de la ecuación tenga raíces distintas.
Si el campo K tiene característica igual a 3 entonces una curva elíptica es el conjunto de soluciones a una ecuación de la forma:

$$Y^2 = X^3 + AX^2 + BX + C,$$

junto con el punto infinito $\mathcal{O}$ y donde se requiere que el polinomio $X^3 + AX^2 + BX + C$ no tenga raíces repetidas.

A continuación presentamos el algoritmo formal para la suma y duplicación de puntos sobre una curva elíptica cuando el campo tiene característica distinta a 2 o 3.

Definición 4.3. Sea E una curva elíptica:

$$E : Y^2 = X^3 + AX + B.$$

Y sean P_1 y P_2 puntos sobre E. Entonces:

1. Si $P_1 = \mathcal{O}$, entonces $P_1 + P_2 = P_2$.
2. Si $P_2 = \mathcal{O}$, entonces $P_1 + P_2 = P_1$.
3. De otra forma podemos ver a los puntos como: $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$.
4. Si $x_1 = x_2$ y $y_1 = -y_2$ entonces $P_1 + P_2 = \mathcal{O}$.
5. En cualquier otro caso $P_1 + P_2 = (x_3, y_3)$ donde:

$$x_3 = \lambda^2 - x_1 - x_2 \quad \text{y} \quad y_3 = \lambda(x_1 - x_3) - y_1$$

con

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & P_1 \neq P_2 \\ \frac{3x_1^2 + A}{2y_1} & P_1 = P_2 \end{cases}$$

Teorema 4.4. *Sea E una curva elíptica, entonces la regla de la adición en E cumple con las siguientes propiedades:*

1. *Identidad:* $P + \mathcal{O} = \mathcal{O} + P = P \quad \forall\ P \in E.$
2. *Inverso:* $P + (-P) = \mathcal{O} \quad \forall\ P \in E.$
3. *Asociatividad:* $(P + Q) + R = P + (Q + R) \quad \forall\ P, Q, R \in E.$
4. *Conmutatividad:* $P + Q = Q + P \quad \forall\ P, Q \in E.$

El grupo de puntos sobre la curva elíptica, junto con la operación dada en la definición 4.3 y el teorema 4.4 nos permite asociar formalmente un grupo abeliano a una curva elíptica.

4.1.1. Ecuación de Weierstrass generalizada e isomorfismos

La ecuación 4.1 es conocida como la ecuación de Weierstrass de una curva elíptica. También existe una ecuación mas general de la forma:

$$Y^2 + a_1XY + a_3 = X^3 + a_2X^2 + a_4X + a_6$$

donde $a_1, \ldots, a_6$ son constantes sobre un campo K. Esta forma más general es conocida como la ecuación de Weierstrass generalizada y es útil cuando se trabaja en campos de característica 2 y 3.

Definición 4.5. Sean E_1, E_2 dos curvas elípticas definidas sobre K con las siguientes ecuaciones de Weierstrass generalizada:

$$E_1 : Y^2 + a_1XY + a_3 = X^3 + a_2X^2 + a_4X + a_6,$$
$$E_1 : Y^2 + b_1XY + b_3 = X^3 + b_2X^2 + b_4X + b_6$$

decimos que son isomorfas sobre el campo K, si existen $u, r, s, t \in K, u \neq 0$, tal que el cambio de variables

$$(X, Y) \to (u^2X + r, u^3Y + u^2sX + t)$$

transforma E_1 en E_2.

Sea E una curva elíptica definida por una ecuación de Weierstrass generalizada en un campo, K, de característica distinta a 2 y 3 es posible realizar el siguiente cambio de variables:

$$(X, Y) \to \left(\frac{X - 3a_1^2 - 12a_2}{36}, \frac{Y - 3a_1X}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24}\right). \tag{4.2}$$

El cambio de variables 4.2 transforma a E a una curva de la forma

$$Y^2 = X^3 + AX + B, \quad A, B \in K,$$

esto es, una ecuación de Weierstrass.

4.2. Curvas Elípticas sobre campos finitos

Para que podamos aplicar la teoría sobre curvas elípticas a la criptografía, necesitamos trabajar sobre aquellas curvas que tengan coordenadas en un campo finito $\mathbb{F}_p$.

Denotamos una curva elíptica sobre un campo finito $\mathbb{F}_p$, $p \neq 2, 3$ de la siguiente forma:

$$E : Y^2 = X^3 + AX + B \quad A, B \in \mathbb{F}_p \quad \text{y} \quad 4A^3 + 27B^2 \not\equiv 0 \pmod{p}.$$

Todos los puntos en E con coordenadas en $\mathbb{F}_p$ están definidos como sigue:

$$E(\mathbb{F}_p) = \{(x, y) : x, y \in \mathbb{F}_p \mid y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

El siguiente teorema, nos permite ver que es posible utilizar el algoritmo de la suma de puntos sobre una curva elíptica como en la definición 4.3.

Teorema 4.6. *Sea E una curva elíptica sobre un campo $\mathbb{F}_p$ y sean P y Q puntos en $E(\mathbb{F}_p)$. Entonces:*

(a) El algoritmo de adición de la definición 4.3, aplicado a P y Q da como resultado un punto en $E(\mathbb{F}_p)$. A este punto lo denotamos como $P + Q$.

(b) La regla de adición en $E(\mathbb{F}_p)$ satisface todas las propiedades listadas en el teorema 4.4. En otras palabras la regla de adición hace de $E(\mathbb{F}_p)$ un grupo abeliano finito.

Sea E una curva elíptica sobre $\mathbb{F}_p$. El número de puntos en $E(\mathbb{F}_p)$, es llamado el **orden** de E sobre $\mathbb{F}_p$. Como la ecuación 4.1 tiene a lo más dos soluciones para cada $x \in \mathbb{F}_p$ el orden es a lo más $2q + 1$, dos soluciones por cada x y el punto $\mathcal{O}$. El teorema 4.7 nos da una mejor cota para el orden de una curva elíptica.

Teorema 4.7 (Teorema de Hasse)**.** *Sea N el número de puntos en una curva elíptica definida sobre $\mathbb{F}_p$. Entonces:*

$$|N - (q + 1)| \leq 2\sqrt{q}.$$

4.3. El problema del logaritmo discreto sobre curvas elípticas

Si P es un punto sobre una curva elíptica y k un entero positivo entonces denotamos a kP como:

$$kP = \underbrace{P + P + P + \cdots + P}_{k \text{ veces}}.$$

Si k es un número menor que 0 tenemos que:

$$kP = \underbrace{(-P) + (-P) + (-P) + \cdots + (-P)}_{k \text{ veces}}.$$

Algoritmo 5 Método para la multiplicación de un punto P por un escalar k.

Input: k entero positivo, $P \in E(\mathbb{F}_p)$.
Output: kP

```
a ⇐ k
B ⇐ O
C ⇐ P
while a ≠ 0 do
    if a es par then
        a ⇐ a/2
        B ⇐ B
        C ⇐ 2C
    end if
    if a es impar then
        a ⇐ a − 1
        B ⇐ B + C
        C ⇐ C
    end if
end while
return B
```

De esta forma es posible definir el problema del logaritmo discreto (Definición 3.7), sobre un grupo de puntos en una curva elíptica $E(\mathbb{F}_p)$.

Definición 4.8. El **orden** N de un punto P sobre una curva elíptica, es el entero positivo más pequeño tal que $NP = \mathcal{O}$.

Definición 4.9. Sea E una curva elíptica sobre el campo finito $\mathbb{F}_p$ y $P \in E(\mathbb{F}_p)$ un elemento de orden n entonces para un elemento $Q \in \langle P \rangle$ el **problema del logaritmo discreto elíptico** (PLDCE) consiste en encontrar el único entero $k, 0 \leq k \leq n-1$ tal que $Q = kP$. Denotamos a k como:

$$k = log_P(Q)$$

y lo llamamos el logaritmo discreto elíptico de Q con respecto a P.

Es posible utilizar las técnicas descritas en la sección 3.5 para resolver el problema del logaritmo discreto sobre curvas elípticas.

4.4. Criptografía sobre curvas elípticas

4.4.1. El método de duplicación sucesiva

Calcular kP utilizando sumas sucesivas es ineficiente especialmente si k es grande. El método de duplicación sucesiva, descrito en el algoritmo 5, nos permite calcular estos valores de forma más eficiente.

Veamos que el algoritmo 5 es correcto y que realmente calcula kP.

Proposición 4.10. *Sea k un número entero positivo y P un punto en una curva elíptica. La siguiente expresión*

$$I = aC + B = kP,$$

para a, C y B definidos en el algoritmo se cumple durante todo el procedimiento. A I la denominaremos con el nombre de ***invariante****.*

Demostración.

1. **Caso Base:** Al principio del algoritmo tenemos lo siguiente:

$$\begin{aligned} a &= k \\ C &= P \\ B &= \mathcal{O}. \end{aligned}$$

   Es claro ver que la invariante se cumple pues

$$I = aC + B = kP + \mathcal{O} = kP.$$

2. **Paso inductivo** Veamos que la invariante se mantiene durante cada paso del algoritmo.

   a) a **es par:**

$$\begin{aligned} a' &= \frac{a}{2} \\ B' &= B \\ C' &= 2C. \end{aligned}$$

   De aquí tenemos

$$\begin{aligned} I &= a'C' + B' = \frac{a}{2} \cdot 2C + \mathcal{O} \\ I &= aC + \mathcal{O} \\ I &= kP + \mathcal{O} = kP. \end{aligned}$$

   De aquí se sigue que la condición se cumple para el caso a par.

   b) a **es impar:**

$$\begin{aligned} a' &= a - 1 \\ B' &= B + C \\ C' &= C, \end{aligned}$$

   obteniendo

$$\begin{aligned} I &= a'C' + B' \\ I &= (a-1)C + (B + C) \\ I &= aC - C + B + C \\ I &= aC + B = kP + \mathcal{O} = kP. \end{aligned}$$

   Por lo que la condición se cumple para el caso a impar.

Algoritmo 6 Método binario de derecha a izquierda para la multiplicación de un punto P por un escalar k.

Input: $k = (k_{t-1}, \ldots, k_1, k_0)_2, P \in E(\mathbb{F}_p)$.
Output: kP
  $Q \Leftarrow \mathcal{O}$
  while $i < t$ **do**
    if $k_i = 1$ **then**
      $Q \Leftarrow Q + P$
    end if
    $P \Leftarrow 2P$
  end while
  return Q

Tenemos entonces que la condición dada por I se mantiene a lo largo de cada paso del algoritmo. Eventualmente se llegara a un punto de terminación dado por la condición $a = 0$. Esta condición siempre se cumple pues k es un número positivo y en cada paso del algoritmo a decrece de forma discreta (siempre se manejan números enteros). Tenemos que a no puede ser menor que 0 pues dividir un número positivo par entre 2 siempre obtenemos un número entero positivo y restar 1 solo puede ocurrir cuando a es impar. El entero positivo impar más pequeño es 1 al cual si le restamos 1 obtenemos 0. Garantizando que el número de pasos del algoritmo es finito.

Al momento de terminación tenemos que

$$\begin{aligned} I &= a'C' + B' = kP \\ I &= 0C' + B' = kP \\ I &= \mathcal{O} + B' = kP \\ I &= B' = kP. \end{aligned}$$

El valor B es regresado por el algoritmo.

■

Es posible utilizar la representación binaria del número k para obtener los mismos resultados pero evitando realizar la operación división, la cual no es eficiente para una computadora. Este procedimiento esta descrito en el algoritmo 6.

A continuación describimos el intercambio de llaves Diffie-Hellman elíptico y el sistema de cifrado ElGamal elíptico. La seguridad de ambos esta basada en el problema del logaritmo discreto elíptico.

4.4.2. Intercambio de llaves de Diffie-Hellman elíptico

El Intercambio de llaves de Diffie-Hellman elíptico, descrito en la tabla 4.1, consta de las mismas partes descritas en el intercambio de llaves de Diffie-

Parte preliminar	
Un tercero en confianza escoge un número primo grande p Una curva elíptica E sobre $\mathbb{F}_p$. y un punto $P \in E(\mathbb{F}_p)$. Se publican p, E y P.	
Creación de parámetros públicos y privados	
Alice	Bob
Escoge un entero secreto $k_{privA} = a$ Calcula $k_{pubA} = A = aP$	Escoge un entero secreto $k_{privB} = b$ Calcula $k_{pubB} = B = bP$
Intercambio de llaves	
Alice	Bob
Envía el parámetro A a Bob	Envía el parámetro B a Alice
Creación del secreto compartido	
Alice	Bob
Calcula aB.	Calcula bA.
El secreto compartido es $aB = a(bP) = (ab)P = bA = b(aP)$.	

Tabla 4.1: Intercambio de llaves Diffie-Hellman elíptico.

Hellman (sección 3.2), con la diferencia de que se utiliza al grupo de puntos sobre una curva elíptica.

Ejemplo 4.11. Alice y Bob deciden usar el intercambio de llaves de Diffie-Hellman elíptico para crear un secreto compartido y acuerdan usar los parámetros $p = 4357$, $E : y^2 = x^3 + 345x + 27$ y $P = (3169, 661) \in E(\mathbb{F}_p)$.

Cada uno de ellos calcula sus respectivas llaves públicas y privadas. Para Alice: $k_{privA} = a = 201, k_{pubA} = A = 201P = (1782, 970)$ y para Bob: $k_{privB} = b = 1721, k_{pubB} = B = 1721P = (2747, 873)$.

Observe que después de realizar el intercambio de las llaves públicas, A, B, ambos Alice y Bob pueden calcular el secreto compartido. Para Alice se tiene que $aB = 201(1721P) = (156, 1932)$ y para Bob $bA = 1721(201P) = (156, 1932)$.

4.4.3. Sistema de cifrado ElGamal elíptico

El sistema de cifrado ElGamal elíptico, descrito en la tabla 4.2, consta de las mismas partes descritas en el sistema de cifrado ElGamal (sección 3.3), con las siguientes diferencias:

1. Se utiliza al grupo de puntos sobre una curva elíptica.

2. ElGamal elíptico posee una expansión de mensaje de 4 a 1, a diferencia de la expansión de 2 a 1 para el caso descrito en la sección 3.3. Esto pues cada punto sobre la curva elíptica consiste de 2 coordenadas.

3. No existe una forma natural de codificar mensajes en puntos sobre una curva elíptica.

Ejemplo 4.12. Alice utiliza el primo $p = 5351$, la curva elíptica $y^2 = x^3 + 721x + 456$ y el punto $P = (4301, 2827)$. Alice escoge $k_{privA} = a = 128$ como su llave privada y calcula su llave pública $k_{pubA} = A = 128P = (1086, 884)$.

Bob decide enviar el mensaje $m = (3769, 2754)$. Escoge una llave de sesión $k = 1009$ y calcula $c_1 = kP = (4972, 1926)$ y $c_2 = m + kA = (1292, 1148)$. Bob envía (c_1, c_2) a Alice.

Alice calcula $c_2 - ac_1 = (3769, 2754)$.

Parte preliminar	
Un tercero en confianza escoge un número primo grande p Una curva elíptica E sobre $\mathbb{F}_p$. y un punto $P \in E(\mathbb{F}_p)$. Se publican p, E y P.	
Creación de parámetros públicos y privados	
Alice	Bob
Escoge un entero secreto $k_{privA} = a$ Calcula $k_{pubA} = A = aP$ publica A	
Encriptación del mensaje	
Alice	Bob
	Escoge el mensaje $m \in E(\mathbb{F}_p)$ Calcula una llave de sesión k Utiliza A para calcular $c_1 = kP \in E(\mathbb{F}_p)$ y $c_2 = m + kA \in E(\mathbb{F}_p)$ Envía (c_1, c_2) a Alice.
Desencriptación del mensaje	
Alice	Bob
Calcula $c_2 - ac_1 \in E(\mathbb{F}_p) = m$.	

Tabla 4.2: Sistema de cifrado ElGamal elíptico.

Capítulo 5

Resultados obtenidos Eccp-79

En este capítulo se presentan los resultados obtenidos al encontrar la solución al problema del logaritmo discreto en el reto **Eccp-79**.

5.1. Eccp-79

Los retos propuestos por Certicom están clasificados en tres categorías: curvas elípticas sobre $\mathbb{F}_{2^m}$ con m un entero positivo, curvas elípticas sobre $\mathbb{F}_p$, donde p es un número primo, y curvas de Koblitz [1], estas categorías a su vez están divididas en ejercicios, retos tipo I y retos tipo II. Cada uno de estos retos o ejercicios, consiste en encontrar la llave privada, dada una llave pública Q, un punto base P de orden n y otros parámetros asociados al sistema de cifrado. Es decir, para cada reto se debe calcular el único número $l, 0 \leq l \leq n-1$ tal que $Q = lP$, por lo tanto cada uno de estos retos consiste en resolver el PLDCE. Con la excepción de los basados en curvas de Koblitz, todos los retos fueron generados utilizando parámetros aleatorios, esto significa que las llaves privadas l son desconocidas incluso para Certicom; sin embargo, es posible ver que una respuesta hallada sea la correcta, verificando que $Q = lP$. Entre las metas planteadas por Certicom se encuentran:

- Aumentar el conocimiento acerca de CCE y sobre la dificultad de PLDCE.
- Comparar CCE con otros sistemas de cifrado, específicamente con RSA.
- Ayudar a los usuarios a utilizar tamaños de llave apropiados.
- Comparar la dificultad de PLDCE para $\mathbb{F}_{2^m}$ y $\mathbb{F}_p$.
- Comparar la dificultad de PLDCE para $\mathbb{F}_{2^m}$, en curvas elegidas de forma aleatoria y curvas de Koblitz.
- Fomentar la investigación en algoritmos para el área de teoría computacional de los números.

[1] Curvas elípticas sobre $\mathbb{F}_{2^m}$ con exactamente dos puntos en $E(\mathbb{F}_2)$.

Para las curvas sobre un campo finito $\mathbb{F}_p$ se crearon tres ejercicios: Eccp-79, Eccp-89 y Eccp-97.

Para **Eccp-79** los parámetros asociados, al sistema de cifrado, en hexadecimal, son los siguientes:

- $p = 0x62CE5177412ACA899CF5$.
- $\texttt{seedE} = 0x3409C5C7E50FF4D696E67687561517552DF2B489$.
- $r = 0x1CE4AF36EED8DE22B99D$.
- $a = 0x39C95E6DDDB1BC45733C$.
- $b = 0x1F16D880E89D5A1C0ED1$.
- $n = 0x62CE5177407B7258DC31$.
- $h = 01$.
- $\texttt{seedP} = 0x1FD36F3C978D0398EAB24D696E67687561517593$.
- $P_x = 0x315D4B201C208475057D$.
- $P_y = 0x035F3DF5AB370252450A$.
- $\texttt{seedQ} = 0x37A5B90A4D696E676875615175D52727640CFCA6$.
- $Q_x = 0x0679834CEFB7215DC365$.
- $Q_y = 0x4084BC50388C4E6FDFAB$.

Donde cada parámetro se interpreta como sigue:

- p, un número primo, representa el tamaño del grupo asociado a la curva elíptica.
- seedE, representa la semilla que fue utilizada para generar los parámetros a, b y puede ser utilizada para verificar que a, b fueron generados de manera aleatoria.
- r, un parámetro utilizado por el algoritmo que genera a la curva.
- a, b, los elementos que determinan a la curva $E : Y^2 = X^3 + aX + b$.
- n, el orden del punto P. El entero n es un número primo.
- h, el número de puntos en la curva $E(\mathbb{F}_p)$ entre n.
- seedP, representa la semilla que fue utilizada para generar los parámetros P_x, P_y y puede ser utilizada para verificar que P_x, P_y fueron generados de manera aleatoria.

- P_x, P_y, las coordenadas x, y del punto P. Este es el punto que representan al punto base.
- seedQ, representa la semilla que fue utilizada para generar los parámetros Q_x, Q_y y puede ser utilizada para verificar que Q_x, Q_y fueron generados de manera aleatoria.
- Q_x, Q_y, las coordenadas x, y del punto P. Este es el punto que representa la llave pública.

De acuerdo con Certicom este ejercicio podría ser resuelto en unas horas utilizando una red de al menos 3000 computadoras de tipo Pentium 100.

5.2. Antecedentes

A continuación, se presenta información acerca de las veces que se resolvió **Eccp-79**.

5.2.1. Robery Harley y Wayne Baisley

En 1997, el mismo año en que Certicom publico sus retos, Robert Harley del Instituto Nacional de Investigación en Informática y Automática (INRIA, por sus siglas en francés) junto con Wayne Baisley lograron resolver **Eccp-79** utilizando un algoritmo ρ **de Pollard** junto con un algoritmo de detección de ciclos tipo Brent, ambos en paralelo. Un total de 1400 billones de pasos fueron realizados. Esta fue la primera vez que se resolvió el Eccp-79. En la figura 5.1 se muestra el correo enviado a Certicom anunciando que se había resuelto el reto.

5.2.2. Laboratorios BT

Integrantes de los Laboratorios BT, utilizando una arquitectura cliente-servidor, y con la participación de más de 1200 máquinas de diferentes partes del mundo, lograron resolver **Eccp-79** en 53 días. Se utilizó un algoritmo ρ de Pollard distribuido con puntos distinguidos, donde el criterio para estos fue que la coordenada x tenga los últimos 30 bits cero. Se hallaron $186,364$ puntos distinguidos y se realizaron alrededor de $200,000,000,000,000$ operaciones sobre el grupo de la curva elíptica [ESST99].

5.3. Trabajo previo a resolver Eccp-79

A continuación se presentan las pruebas y ejercicios que se realizaron antes de resolver el ejercicio Eccp-79.

To: certicom-ecc-challenge@certicom.com

Robert J. Harley,
Se‘vres, France,
6th of December, 1997.

Dear Anonymous,

Certicom's professed aim in setting its ECC challenge is to encourage research into secure cryptosystems based on elliptic curve discrete logarithms. Yet Certicom is a member of the Key Recovery Alliance, a lobby group whose purpose is to promote the use of back-doors allowing supposedly secure communications to be intercepted. How are these contradictory positions reconciled?

The solution to your ECCp-79 problem is the residue class of 9222150721970534568535O modulo 466597814831947642887217. It was found by Wayne Baisley and myself using several Digital Alpha workstations running Linux and Digital Unix at the Institut National de Recherche en Informatique et Automatique (INRIA), at Fermi National Accelerator Laboratory and at the California Institute of Technology C.S. Department.

The method used was a "birthday paradox" algorithm iterating from a random initial point (one per machine) with a random function (the same on all machines) until a collision was detected at 17:58 today at INRIA, Rocquencourt, France by a 500MHz Linux machine. This machine did 25 billion elliptic curve operations per day. The peak rate of all machines was approximately 6 six times as much. A total of about 1400 billion iterations were performed.

If this is the first correct submission, please send the prize (a copy of "Handbook of Applied Cryptography" and Maple software) to the following address:

Robert Harley,
c/o Sylvie Loubressac,
Projet CRISTAL,
INRIA,
Domaine de Voluceau - Rocquencourt,
78153 Le Chesnay,
France.

Thank you,
Rob.

Robert.Harley@inria.fr

‘-’ Linux + 500MHz Alpha + 256MB SDRAM = heaven ‘-’

Figura 5.1: Correo electrónico enviado a Certicom por Robert Harley.

5.3.1. Algoritmo ρ de Pollard con detección de ciclos de Floyd

Se realizaron pruebas utilizando el algoritmo ρ de Pollard con el algoritmo de detección de ciclos de Floyd para resolver el problema del logaritmo discreto sobre grupos basados en curvas elípticas de tamaño pequeño. Las pruebas se realizaron utilizando un programa creado con el lenguaje de programación **Python** y las librerías **gmpy2** y **xxhash**, las cuales permiten el uso de aritmética para números enteros grandes y el uso de la función hash xxhash respectivamente, el código se presenta en el Apéndice A.
De igual forma se utilizo el sistema algebraico computacional **Sage**, el código se presenta en el Apéndice B.

Las pruebas fueron realizadas en dos tipos de equipos de cómputo:

1. Una computadora portátil con procesador Intel Pentium Core Duo P6100 de 2.00GHz.
2. Una computadora de escritorio con un procesador Intel Pentium Quad Core i5 3.4GHz.

Resultados

1. Curva: $y^2 = x^3 + 116889722x + 127814978$ sobre campo finito de tamaño 883550527.
   - Bits: 30.
   - Punto base: $(793086996, 41970795)$.
   - Punto público: $(487577379, 561579834)$.
   - Solución: 205059194.
   - Computadora portátil utilizando Sage.
     - Tiempo: 4.39566421509 segundos.
   - Computadora portátil utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 0.431364059448 segundos.
     - Sumas: 29074.
     - Duplicaciones: 2721.
   - Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 0.186 segundos.
     - Sumas: 29074.
     - Duplicaciones: 2721.
2. Curva $y^2 = x^3 + 306090781x + 983239922$ sobre campo finito de tamaño 1053929377.
   - Bits: 30.

- Punto base: $(190134140, 865470343)$.
- Punto público: $(314257857, 789147536)$.
- Solución: 280369298.
- Computadora portátil utilizando Sage.
  - Tiempo: 5.00844502449 segundos.
- Computadora portátil utilizando las librerías gmpy2 y xxhash.
  - Tiempo: 1.26011896133 segundos.
  - Sumas: 84804.
  - Duplicaciones: 4549.
- Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
  - Tiempo: 0.538 segundos.
  - Sumas: 84804.
  - Duplicaciones: 4549.

3. Curva $y^2 = x^3 + 578623350x + 41049144$ sobre campo finito de tamaño 1039412951.

   - Bits: 30.
   - Punto base: $(415203460, 490822219)$.
   - Punto público: $(449820043, 926169471)$.
   - Solución: 488327655.
   - Computadora portátil utilizando Sage.
     - Tiempo: 4.20551896095 segundos.
   - Computadora portátil utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 1.4520740509 segundos.
     - Sumas: 99135.
     - Duplicaciones: 5225.
   - Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 0.642112970352 segundos.
     - Sumas: 99135.
     - Duplicaciones: 5225.

4. Curva $y^2 = x^3 + 171278553262x + 784961005892$ sobre campo finito de tamaño 1047124306831.

   - Bits: 40.
   - Punto base: $(262981436772, 693630898592)$.
   - Punto público: $(896484642207, 914625175156)$.
   - Solución: 684364444347.

- Computadora portátil utilizando Sage.
  - Tiempo: 361.386116982 segundos o 6.0231019497 minutos.
- Computadora portátil utilizando las librerías gmpy2 y xxhash.
  - Tiempo: 84.7567358017 segundos o 1.4126122633616667 minutos.
  - Sumas: 5731686.
  - Duplicaciones: 188781.
- Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
  - Tiempo: 37.3783500195 segundos.
  - Sumas: 5731686.
  - Duplicaciones: 188781.

5. Curva $y^2 = x^3 + 420504793882x + 503525268986$ sobre campo finito de tamaño 652415617283.
   - Bits: 40.
   - Punto base: $(157207871760, 308147219821)$.
   - Punto público: $(590848203042, 356260927146)$.
   - Solución: 290722406271.
   - Computadora portátil utilizando Sage.
     - Tiempo: 106.455410957 segundos, 1.7742568492833333 minutos.
   - Computadora portátil utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 37.2667558193 segundos.
     - Sumas: 2601907.
     - Duplicaciones: 86114.
   - Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 16.5737009048 segundos.
     - Sumas: 2601907.
     - Duplicaciones: 86114.
6. Curva $y^2 = x^3 + 636193494687943x + 649457870618682$ sobre campo finito de tamaño 718224257705687.
   - Bits: 50.
   - Punto base: $(119615542059394, 458559941571622)$.
   - Punto público: $(548110361706393, 153678407138095)$.
   - Solución: 250728265360524.
   - Computadora portátil utilizando Sage.
     - Tiempo: 3433.23071504 segundos o 57.22051191733333 minutos.
   - Computadora portátil utilizando las librerías gmpy2 y xxhash.

- Tiempo: 957.843756914 segundos o 15.964062615233333 minutos.
- Sumas: 63124777.
- Duplicaciones: 2037369.

- Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
  - Tiempo: 417.905923128 segundos o 6.965098718799999 minutos.
  - Sumas: 63124777.
  - Duplicaciones: 2037369.

7. Curva $y^2 = x^3 + 2803979435699277612x + 13649164907490030661$ sobre campo finito de tamaño 14216042240398994003.
   - Bits: 64.
   - Punto base: $(13381460426529769002L, 1082937423467519620)$.
   - Punto público: $(6007767484326270812, 3032269530778218781)$.
   - Solución: 12473522145187916992.
   - Computadora de escritorio utilizando las librerías gmpy2 y xxhash.
     - Tiempo: 21632.5831749 segundos o 6.009050881916666 horas.
     - Sumas: 3277905168.
     - Duplicaciones: 105752723.

5.3.2. ρ de Pollard con puntos distinguidos

Como se puede observar en los resultados anteriores, utilizando el algoritmo ρ de Pollard de forma secuencial toma alrededor de seis horas resolver el problema del logaritmo discreto sobre un curva de un tamaño aproximado de 64 bits. Para poder resolver el ejercicio de forma más rápida, se realizaron las siguientes pruebas en cuatro curvas de 50 y una de 64 bits, curva 5, estas fueron generadas utilizando Sage. P representa al punto base y Q, un múltiplo de P, es el punto publico. El objetivo de estas pruebas era observar qué parámetros eran los que ofrecían mejor tiempo utilizando el algoritmo de ρ de Pollard distribuido con puntos distinguidos.

En las siguientes pruebas el criterio utilizado para los puntos distinguidos, D, es considerar a aquellos puntos en la curva, E, tal que sus últimos n_D bits del resultado de aplicar una función hash a la primera coordenada del punto, son cero. En otras palabras

$$D = \{P = (x_P, y_P) \in E(\mathbb{F}_p) \mid H(x_P) \equiv 0 (\mathrm{mod} 2^{n_D})\},$$

donde $0 \leq x_P < p$.

Curvas

- Curva 1

  Ecuación $y^2 = x^3 + 293493353247694x + 771825103345424$.
  Orden F_q 1077040394527781.
  Orden $E(\mathbb{F}_q)$ 1077040344223619.
  P $(604150508091039, 910867655268449)$.
  Q $(96022630066051, 132846410458235)$.
  Solución 35923591975345.

- Curva 2

  Ecuación $y^2 = x^3 + 880750377387087x + 636090295938329$.
  Orden F_q 1016750710940339.
  Orden $E(\mathbb{F}_q)$ 1016750704033459.
  P $(207933417508077, 682854270281968)$.
  Q $(875934491745948, 779036929470134)$.
  Solución 526624276831221.

- Curva 3

  Ecuación $y^2 = x^3 + 404296406395548x + 417633452293915$.
  Orden F_q 1098525996401653.
  Orden $E(\mathbb{F}_q)$ 1098525988592861.
  P $(539798961312422, 634731194709983)$.
  Q $(361431338089231, 108409084871638)$.
  Solución 321696492599830.

- Curva 4

  Ecuación $y^2 = x^3 + 293493353247694x + 771825103345424$.
  Orden F_q 1077040394527781.
  Orden $E(\mathbb{F}_q)$ 1077040344223619.
  P $(604150508091039, 910867655268449)$.
  Q $(96022630066051, 132846410458235)$.
  Solución 35923591975345.

- Curva 5

  Ecuación $y^2 = x^3 + 9149401130010625903x + 5841903620978180006$.
  Orden F_q 14641239623330541439.
  Orden $E(\mathbb{F}_q)$ 14641239621854511439.
  P $(13689987213991244736, 11539019713003473579)$.
  Q $(1374766878626570264, 518389615513573853)$.
  Solución 946066052259388628.

Notación

Para las pruebas realizadas en esta sección se utiliza la siguiente notación:

(a) N_s: Número de particiones para el algoritmo ρ **de Pollard**.

(b) n_D: Criterio para los puntos distinguidos, esto es que los bits menos significativos de $H(x_p)$ sean cero donde H es la función xxhash.

(c) D: Número de puntos distinguidos. $D \approx 2^{n_D}$.

(d) Puntos distinguidos: Cantidad de puntos distinguidos encontrados.

(e) θ: Probabilidad que un punto uniformemente escogido al azar sea un punto distinguido. Es dada por $\theta \approx \frac{1}{2^{n_D}}$.

Resultados de las pruebas

Curva 1

N_s	n_D	θ	D	Puntos distinguidos	tiempo
50	16	0.0000152587890625000	2^{34}	332	22 segundos
64	16	0.0000152587890625000	2^{34}	690	48 segundos
32	16	0.0000152587890625000	2^{34}	453	30 segundos
50	19	$1.90734863281250e-6$	2^{31}	117	60 segundos
64	19	$1.90734863281250e-6$	2^{31}	37	20 segundos
32	19	$1.90734863281250e-6$	2^{31}	97	50 segundos
50	8	0.00390625000000000	2^{42}	72871	23 segundos
64	8	0.00390625000000000	2^{42}	213468	67 segundos
32	8	0.00390625000000000	2^{42}	99764	32 segundos

Curva 2

N_s	n_D	θ	D	Puntos distinguidos	tiempo
50	16	0.0000152587890625000	2^{34}	459	31 segundos
64	16	0.0000152587890625000	2^{34}	554	34 segundos
32	16	0.0000152587890625000	2^{34}	2246	140 segundos
50	19	$1.90734863281250e-6$	2^{31}	59	29 segundos
64	19	$1.90734863281250e-6$	2^{31}	42	26 segundos
32	19	$1.90734863281250e-6$	2^{31}	33	15 segundos
50	8	0.00390625000000000	2^{42}	238695	76 segundos
64	8	0.00390625000000000	2^{42}	231990	74 segundos
32	8	0.00390625000000000	2^{42}	136701	44 segundos

Curva 3

N_s	n_D	θ	D	Puntos distinguidos	tiempo
50	16	0.0000152587890625000	2^{34}	1648	109 segundos
64	16	0.0000152587890625000	2^{34}	411	26 segundos
32	16	0.0000152587890625000	2^{34}	394	25 segundos
50	19	$1.90734863281250e-6$	2^{31}	130	63 segundos
64	19	$1.90734863281250e-6$	2^{31}	100	58 segundos
32	19	$1.90734863281250e-6$	2^{31}	63	35 segundos
50	8	0.00390625000000000	2^{42}	61251	20 segundos
64	8	0.00390625000000000	2^{42}	338971	106 segundos
32	8	0.00390625000000000	2^{42}	126063	39 segundos

Pruebas realizadas utilizando la misma semilla para el generador pseudoaleatorio

Para disminuir los efectos causados por la aleatoriedad en el algoritmo ρ de Pollard, las siguientes pruebas se realizaron utilizando la misma semilla para la función aleatoria en cada caso, esto permitía que se realizara la misma caminata cada vez que se repetía el experimento.

Curva 4 y 5

Curva	n_D	θ	D	Puntos distinguidos	tiempo
4	16	0.0000152587890625000	2^{34}	634	28 segundos
4	18	$3.81469726562500e-6$	2^{32}	300	53 segundos
4	19	$1.90734863281250e-6$	2^{31}	183	64 segundos
5	16	0.0000152587890625000	2^{48}	58343	46.6 minutos
5	21	$4.76837158203125e-7$	2^{43}	1388	35.15 minutos
5	26	$1.49011611938477e-8$	2^{38}	93	72.5 minutos

Conclusiones

Para las pruebas realizadas en las curvas 1,2 y 3 se puede notar que un criterio para los puntos distinguidos con un valor de θ elevado tiene como consecuencia que el algoritmo necesitará de más espacio de almacenamiento, esto pues una mayor cantidad de puntos sobre la curva cumplirán con el criterio utilizado. Es más difícil llegar a una conclusión con respecto al número de particiones utilizado pues en las tres curvas se obtuvieron los mejores tiempos con diferentes tamaños, 64 para la curva 1, 32 para la curva 2 y 50 para la curva 3.

Para las pruebas realizadas en las curvas 4 y 5 se utilizo el mismo número de particiones $N_s = 64$. El mejor rendimiento fue con $n_D = 16$. En los otros casos se puede notar una disminución en los puntos encontrados al igual que un aumento en el tiempo. El código utilizado en estas pruebas se encuentra en el Apéndice C.

5.4. Resolviendo el reto Eccp-79

Para resolver el ejercicio Eccp-79, Se utilizó un tamaño de particiones de 64 y el criterio para los puntos distinguidos fue que los últimos 29 bits del resultado de aplicar la función hash xxhash a la coordenada x sean ceros. El código utilizado para Eccp-79 se presenta en el Apéndice C.

5.5. Resultados

Se llegó al siguiente resultado después de aproximadamente siete días de operación ininterrumpida:

$$92221507219705345685350 \bmod 466597814831947642887217$$

Este resultado concuerda con el hallado por Robery Harley y Wayne Baisley, al igual que el encontrado por los Laboratorios BT.

Se encontraron un total de 1472 puntos distinguidos y se realizaron $697,928,191,270$ pasos de la caminata ρ.

5.6. Variación en el tiempo

En el proceso de resolver el reto Eccp-79, se observó una gran variación en el tiempo que tomaba en resolver el problema del logaritmo discreto sobre la curva de 79 bits. Debido al largo tiempo que toma cada intento, no es posible realizar muchas pruebas. Este experimento realiza el mismo proceso pero en una curva más pequeña, de esta forma es posible tener más puntos de comparación.

5.6.1. Curva utilizada

Se utilizó la siguiente curva:

$$y^2 = x^3 + 171278553262x + 784961005892$$

sobre campo finito de orden 1047124306831. Para resolver el problema del logaritmo discreto se utilizaron los siguientes puntos de la curva:

- Punto base: $P = (262981436772, 693630898592)$.
- Punto público: $Q = (896484642207, 914625175156)$.
- Orden punto base: 1047124182043.

La solución al problema del logaritmo discreto encontrada por el algoritmo es 684364444347.

5.6.2. El experimento

Se realizaron 100 pruebas en los cuales se resolvía el problema del logaritmo discreto sobre la curva, el resultado correcto se obtuvo en todos los intentos. El código utilizado realiza el algoritmo de ρ de Pollard, en un único procesador, utilizando una caminata aditiva, con un tamaño de partición de 32, y puntos distinguidos. El criterio para verificar que un punto es distinguido es que los últimos 16 bits del resultado de aplicar la función xxhash a la coordenada x del punto sean iguales a cero.

5.6.3. Valor esperado

El número de pasos de la caminata ρ esperados esta dado por

$$\sqrt{\frac{\pi N}{2}} \approx 1282504.90012758,$$

donde N es el orden del punto base.

5.6.4. Resultados obtenidos

Se midieron 3 criterios. Primero el tiempo que le tomaba al algoritmo terminar, segundo el número de puntos distinguidos encontrados y finalmente el número de pasos realizados por la caminata ρ. Los resultados se presentan a continuación:

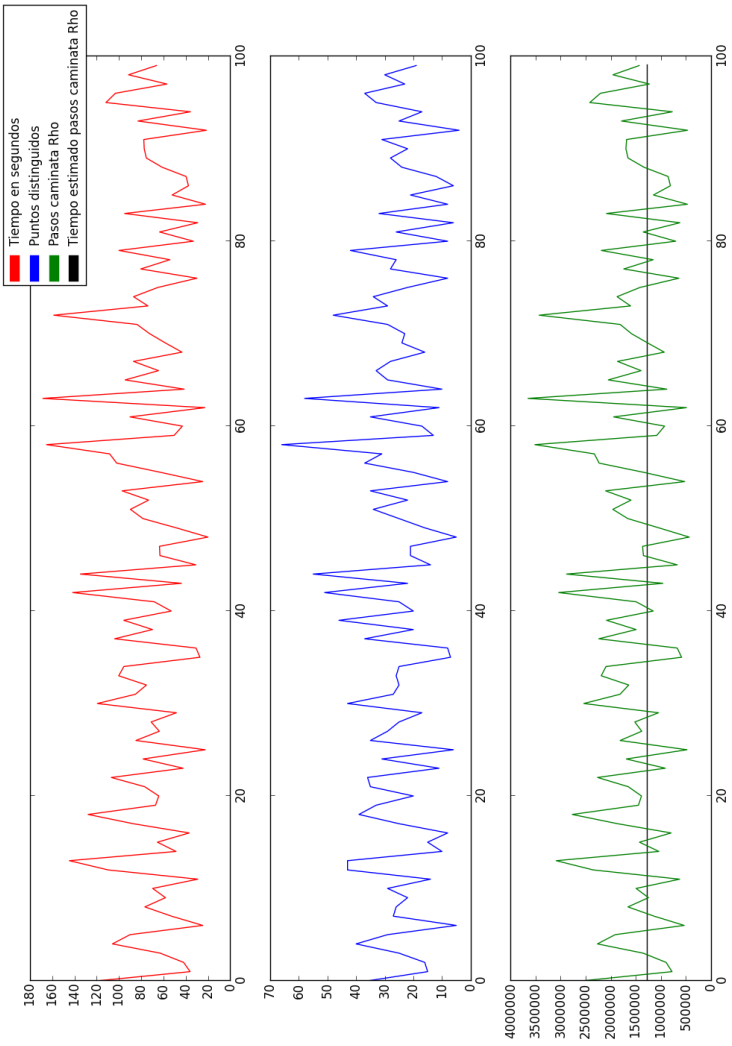

Figura 5.2: Datos obtenidos

Se puede observar que efectivamente existe disparidad en el tiempo para resolver el problema del logaritmo discreto utilizando el algoritmo de ρ de Pollard con puntos distinguidos.

5.7. Curvas Isomorfas

El objetivo de este experimento es ver la variación en el tiempo que toma resolver el problema del logaritmo discreto entre curvas isomorfas.

5.7.1. Curva utilizada

Se utilizo la siguiente curva:

$$y^2 = x^3 + 293493353247694x + 771825103345424$$

sobre campo finito de orden 1077040394527781. Para resolver el problema del logaritmo discreto se utilizaron los siguientes puntos de la curva:

- Punto base: $P = (604150508091039, 910867655268449)$.
- Punto público: $Q = (96022630066051, 132846410458235)$.
- Orden punto base: 1077040344223619.

La solución al problema del logaritmo discreto encontrada por el algoritmo es 35923591975345.

5.7.2. El experimento

Se realizaron 25 experimentos en los cuales se resolvía el problema del logaritmo discreto sobre curvas isomorfas, obtenidas realizando un cambio de variables como se describe en la sección 4.1.1, a la curva dada. El resultado correcto se obtuvo en todos los intentos. El código utilizado realiza el algoritmo de ρ de Pollard, utilizando la misma semilla para la función pseudoaleatoria en cada caso, esto para prevenir la variación causada al escoger diferentes puntos sobre la curva. Se utilizó una caminata aditiva con un tamaño de partición de 32, en un único procesador y puntos distinguidos. El criterio para verificar que un punto es distinguido es que los últimos 19 bits del resultado de aplicar la función xxhash a la coordenada x del punto sean iguales a cero.

5.7.3. Valor esperado

El número de pasos de la caminata ρ esperados esta dado por

$$\sqrt{\frac{\pi N}{2}} \approx 41131630.3654058,$$

donde N es el orden del punto base. El valor esperado no cambia cuando se utilizan curvas isomorfas pues el orden del punto base es el mismo. De igual forma la solución al problema del logaritmo discreto es la misma en cada caso.

5.7.4. Resultados obtenidos

Las curvas isomorfas fueron obtenidas utilizando valores aleatorios para u, r, s, t con $u \neq 0$ en un campo finito de orden 1077040394527781 y realizando un cambio de variables, en la ecuación original. Una vez realizando este cambio se reduce la ecuación a su forma corta de Weierstrass antes de aplicar el algoritmo.

```
Reporte: Problema del logaritmo discreto en curvas isomorfas.

Curva: 0
A: 293493353247694
B: 771825103345424
Punto base P: (604150508091039, 910867655268449)
Punto Q: (96022630066051, 132846410458235)
Cambio de variables u,r,s,t: (1, 0)
                             (0, 0)
Tiempo en segundos: 740.3010571
Pasos de la caminata Rho: 35356229
Puntos distinguidos encontrados: 67

Curva: 1
A: 45584102119070
B: 375215151725967
Punto base P: (668674127881145, 275194143561944)
Punto Q: (886405952856219, 57584016760879)
Cambio de variables u,r,s,t: (80509366823618, 616943416253142)
                             (189461635087234, 72329105974352)
Tiempo en segundos: 1153.14663696
Pasos de la caminata Rho: 54098251
Puntos distinguidos encontrados: 103

Curva: 2
A: 996978875270632
B: 865246227350977
Punto base P: (404140734515191, 568097847225403)
Punto Q: (894436856007258, 90858398262289)
Cambio de variables u,r,s,t: (923607833824186, 636236331906172)
                             (831712418024079, 497048635217135)
Tiempo en segundos: 914.881556988
Pasos de la caminata Rho: 43174622
Puntos distinguidos encontrados: 81

Curva: 3
A: 178202656076862
B: 1044319982550903
Punto base P: (193615108954272, 135178177146581)
Punto Q: (234542872815329, 686118864941253)
```

```
Cambio de variables u,r,s,t: (236514511333595, 1051681030438374)
                             (458324739636714, 234637640435043)
Tiempo en segundos: 919.943671942
Pasos de la caminata Rho: 42691251
Puntos distinguidos encontrados: 75

Curva: 4
A: 676622457098593
B: 411335763477498
Punto base P: (999429442469444, 11727403521130)
Punto Q: (359555834491442, 102360283468724)
Cambio de variables u,r,s,t: (165465395145057, 89801799630348)
                             (964471371645235, 224048414141152)
Tiempo en segundos: 788.93961215
Pasos de la caminata Rho: 37057182
Puntos distinguidos encontrados: 57

Curva: 5
A: 831763137213137
B: 797936534456868
Punto base P: (673572217975278, 1036075596835311)
Punto Q: (834212418193016, 408231043957425)
Cambio de variables u,r,s,t: (397016706992119, 806917407573223)
                             (781525029323621, 365274251180184)
Tiempo en segundos: 674.639209032
Pasos de la caminata Rho: 32197065
Puntos distinguidos encontrados: 81

Curva: 6
A: 343418752964836
B: 308046875336785
Punto base P: (390643305910566, 620610769740185)
Punto Q: (921219318512881, 463115238964918)
Cambio de variables u,r,s,t: (204825700391361, 479438881144439)
                             (149233984835951, 578204474482889)
Tiempo en segundos: 257.410320997
Pasos de la caminata Rho: 11837054
Puntos distinguidos encontrados: 22

Curva: 7
A: 363499689210414
B: 983795082312367
Punto base P: (852213178770676, 685390473686171)
Punto Q: (463470286003298, 821568108320294)
Cambio de variables u,r,s,t: (942765635512216, 251351001768873)
                             (394330775244491, 713473007291081)
```

```
Tiempo en segundos: 267.206452131
Pasos de la caminata Rho: 12351001
Puntos distinguidos encontrados: 31

Curva: 8
A: 376789044394660
B: 127689577389095
Punto base P: (315638009328442, 381672043640285)
Punto Q: (960517664181580, 649946460734698)
Cambio de variables u,r,s,t: (18979636965144, 350935419378688)
                             (230433672756272, 293975162960372)
Tiempo en segundos: 1533.95433211
Pasos de la caminata Rho: 71010156
Puntos distinguidos encontrados: 147

Curva: 9
A: 540369982028221
B: 183088940088306
Punto base P: (356651247077192, 932595764688539)
Punto Q: (524863585861858, 101127445864644)
Cambio de variables u,r,s,t: (408426641989738, 678217801504595)
                             (739774007454284, 12948946956838)
Tiempo en segundos: 479.331922054
Pasos de la caminata Rho: 22675450
Puntos distinguidos encontrados: 45

Curva: 10
A: 389671699157028
B: 104818418016949
Punto base P: (721934383439317, 130090992982238)
Punto Q: (42546217440426, 580135714795968)
Cambio de variables u,r,s,t: (745324031853141, 175678401219057)
                             (64635453797482, 653226963041856)
Tiempo en segundos: 1178.06413913
Pasos de la caminata Rho: 54997592
Puntos distinguidos encontrados: 113

Curva: 11
A: 441216847561801
B: 1049399805319721
Punto base P: (793392448582452, 831827780806688)
Punto Q: (362319966375691, 613411922931641)
Cambio de variables u,r,s,t: (198968577210503, 590375411189389)
                             (38334606182041, 218483545830613)
Tiempo en segundos: 652.900305033
Pasos de la caminata Rho: 30114002
```

```
Puntos distinguidos encontrados: 53

Curva: 12
A: 682911497791520
B: 95176290284283
Punto base P: (396455949040067, 223452479268428)
Punto Q: (894276371480376, 1075171102848259)
Cambio de variables u,r,s,t: (654421535982726, 546048103948153)
                             (384531189610453, 434345898390316)
Tiempo en segundos: 735.748034
Pasos de la caminata Rho: 34000869
Puntos distinguidos encontrados: 60

Curva: 13
A: 513983269742213
B: 674728732454899
Punto base P: (381568700244628, 244509834833743)
Punto Q: (999735071530360, 177595647918733)
Cambio de variables u,r,s,t: (924860227518908, 594961924706695)
                             (905124945398916, 525784023529126)
Tiempo en segundos: 813.355988979
Pasos de la caminata Rho: 37657077
Puntos distinguidos encontrados: 87

Curva: 14
A: 806014266056971
B: 297904056820799
Punto base P: (873187117559359, 183384541775)
Punto Q: (276826024650847, 9219134061226)
Cambio de variables u,r,s,t: (1010551773712768, 197231957531981)
                             (519138777830281, 606555594851762)
Tiempo en segundos: 369.452867985
Pasos de la caminata Rho: 17110615
Puntos distinguidos encontrados: 39

Curva: 15
A: 676404832563340
B: 331320282348216
Punto base P: (940320149754148, 1038086036602716)
Punto Q: (375830419238082, 482496004298331)
Cambio de variables u,r,s,t: (566065477024235, 690573028733175)
                             (944481421246771, 548257948204348)
Tiempo en segundos: 1801.91928196
Pasos de la caminata Rho: 84218116
Puntos distinguidos encontrados: 167
```

```
Curva: 16
A: 514261338320005
B: 942455743510532
Punto base P: (803959898788251, 688844501170072)
Punto Q: (214762565074079, 758120458139116)
Cambio de variables u,r,s,t: (598968532759450, 1022856927521555)
                             (824933944319079, 774922990464036)
Tiempo en segundos: 1352.59056687
Pasos de la caminata Rho: 62691269
Puntos distinguidos encontrados: 113

Curva: 17
A: 1076489722221923
B: 547917193997521
Punto base P: (873122998290503, 528563039589525)
Punto Q: (515250371396703, 218097499447139)
Cambio de variables u,r,s,t: (290876359754746, 880430062410750)
                             (453297331601008, 58785320902068)
Tiempo en segundos: 473.600533962
Pasos de la caminata Rho: 21885825
Puntos distinguidos encontrados: 46

Curva: 18
A: 85311676772614
B: 222445798724942
Punto base P: (765439041718777, 65851776075625)
Punto Q: (228129487232380, 248666002668575)
Cambio de variables u,r,s,t: (268953016247314, 419936822812526)
                             (492044654685844, 143676579921566)
Tiempo en segundos: 641.51853013
Pasos de la caminata Rho: 29657139
Puntos distinguidos encontrados: 60

Curva: 19
A: 635447178941614
B: 571981652253347
Punto base P: (214445500088691, 663413672705457)
Punto Q: (1052439708800106, 469336856293508)
Cambio de variables u,r,s,t: (578074541515974, 307908664751621)
                             (581963071967237, 426098181060925)
Tiempo en segundos: 968.352025986
Pasos de la caminata Rho: 44697564
Puntos distinguidos encontrados: 97

Curva: 20
A: 987708751401551
```

```
B: 461834195810740
Punto base P: (912020992975424, 535736001516300)
Punto Q: (1043750313037738, 994581054763515)
Cambio de variables u,r,s,t: (93945356981724, 214867738608003)
                             (386639814437019, 990149306272400)
Tiempo en segundos: 607.698894024
Pasos de la caminata Rho: 28015562
Puntos distinguidos encontrados: 57

Curva: 21
A: 111597509000555
B: 641896916096751
Punto base P: (301376453640223, 617045455097387)
Punto Q: (114653733701741, 93622822849456)
Cambio de variables u,r,s,t: (70151050893662, 717580249337586)
                             (520868393478630, 292070190315088)
Tiempo en segundos: 438.739675045
Pasos de la caminata Rho: 20922900
Puntos distinguidos encontrados: 49

Curva: 22
A: 427488941078436
B: 169232862176494
Punto base P: (466413545514227, 522139089948109)
Punto Q: (483528054018393, 918948495141689)
Cambio de variables u,r,s,t: (519234821549602, 850491316871595)
                             (152980529452289, 363223713361323)
Tiempo en segundos: 274.674345016
Pasos de la caminata Rho: 13057616
Puntos distinguidos encontrados: 19

Curva: 23
A: 131088866085113
B: 883016188174935
Punto base P: (1024680816087078, 883106333326634)
Punto Q: (873432849701389, 891674202024032)
Cambio de variables u,r,s,t: (286945322196372, 243753912395374)
                             (782912148679207, 157675620961831)
Tiempo en segundos: 1608.75685787
Pasos de la caminata Rho: 74614230
Puntos distinguidos encontrados: 135

Curva: 24
A: 491244025616148
B: 665365620204561
Punto base P: (898496259905283, 173265116053749)
```

```
Punto Q: (79208071760014, 87295695988586)
Cambio de variables u,r,s,t: (807740595810195, 799173308098836)
                             (205707118947441, 822160137934529)
Tiempo en segundos: 516.588517904
Pasos de la caminata Rho: 23850403
Puntos distinguidos encontrados: 45

Curva: 25
A: 732840758446744
B: 362976778419533
Punto base P: (9933495481869, 243970360935458)
Punto Q: (19920074153299, 696875826001822)
Cambio de variables u,r,s,t: (585020462153697, 612592199473994)
                             (622235319123491, 309581305037810)
Tiempo en segundos: 727.023602962
Pasos de la caminata Rho: 33662140
Puntos distinguidos encontrados: 58
```

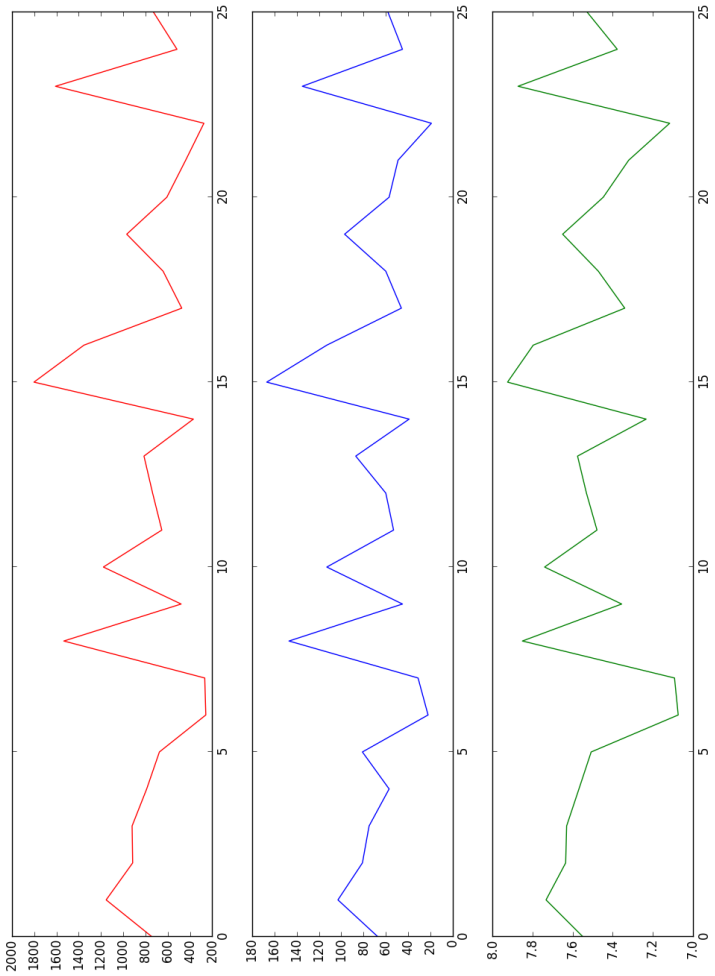

Figura 5.3: Datos obtenidos

Los mínimos en puntos distinguidos fueron alcanzados en las curvas $6, 7, 14, 22$. Sin embargo, no se encontró una reducción significativa en el tiempo para resolver el problema del logaritmo discreto utilizando el algoritmo de ρ de Pollard con puntos distinguidos.

5.8. Mejorando el tiempo para Eccp-79

Algoritmo 7 Algoritmo para inversiones múltiples

Input: $0 < x_1, \ldots, x_k < n$, n orden del grupo.
Output: $y_1 = \frac{1}{x_1} (\text{mód } n), \ldots, y_k = \frac{1}{x_k} (\text{mód } n)$

```
z_1 = x_1
for i = 2 to k do
    z_i = z_{i-1} x_i (mód n)
end for
q = 1/z_k (mód n)
for i = k down to 2 do
    y_i = q z_{i-1} (mód n)
    q = q x_i (mód n)
end for
y_1 = q.
```

Las siguientes pruebas se realizaron para buscar una forma de mejorar el tiempo obtenido en los primeros intentos en resolver Eccp-79. Para las siguientes pruebas tenemos las siguientes notas y nomenclatura:

- **ND**: Número de últimos bits iguales a cero para que se considere al punto como distinguido.
- **NS**: Tamaño de partición.
- **L**:

$$L = \frac{\text{Número de pasos realizados hasta encontrar una solución}}{\sqrt{N}}.$$

  Donde N es el orden del punto base. Para estas pruebas $N = 1077040344223619$.

- **L0**: L para el tiempo esperado. $L0 \approx 1.416$.
- El algoritmo 7 es utilizado en las pruebas donde se requiere realizar varias inversiones en un solo paso.
- Para las pruebas donde se realiza el calculo de los puntos de manera concurrente, cabe aclarar que no fue de forma paralela, sino que se realizaba el calculo sobre varios puntos iniciales hasta encontrar un punto distinguido en vez de utilizar un solo punto inicial.

```
Prueba 1:
ND = 19.
NS = 47.
Adiciones = 47.
Duplicaciones = 0.
Hash:  Coordenada x módulo NS.
Criterio punto distinguido: Últimos ND bits de la coordenada x son cero.
Tipo de caminata: Aditiva.
Coordenadas: Afines.

Pasos realizados: [82436844, 5308820, 68656604, 35149151,
                   35594621, 21944965, 8672421, 47802617,
                   35169909, 56688912]

Promedio pasos realizados: 39742486.4

Puntos distinguidos: [66, 4, 54, 29, 27, 19, 5, 40, 29, 40]
Promedio puntos: 31.3

L: [2.5119174976290335, 0.16176405115366818, 2.092022408273993,
    1.0710231389220217, 1.0845969711234196,  0.6686808821593985,
    0.2642557016945661, 1.4565845106195368,  1.0716556520179352,
    1.7273571265580288]

Promedio L: 1.21098579402
Diferencia L0: 0.205014205985

Tiempo: [1048.3622419834137, 66.9652509689331,  863.9461259841919,
         443.363343000412,   450.572322845459,  276.91058707237244,
         109.52768993377686, 601.2838129997253, 453.8949248790741,
         731.1619720458984]

Promedio tiempos: 504.598827171


Prueba 2:
ND = 19.
NS = 47.
Adiciones = 47.
Duplicaciones = 0.
Hash:  Coordenada x módulo NS.
Criterio punto distinguido: Últimos ND bits del xxhash de la coordenada x son
cero.
Tipo de caminata: Aditiva.
Coordenadas: Afines.
```

```
Pasos realizados: [11893667, 76811262, 4916370, 40908673,
                   19859505, 18175471, 41851335, 59894321,
                   40904330, 63470179]

Promedio pasos realizados: 37868511.3

Puntos distinguidos: [23, 148, 10, 89, 43, 38, 81, 108, 66, 97]
Promedio puntos: 70.3

L: [0.3624096799274971, 2.340501451423444,  0.14980578135449302,
    1.246520445560536,  0.6051352245332351, 0.5538213427062911,
    1.2752441212528027, 1.825028538556256,  1.246388110828117,
    1.9339878320396013]

Promedio L: 1.15388425282
Diferencia L0: 0.262115747182

Tiempos: [202.99267196655273, 1309.5286099910736, 85.07338500022888,
          703.8299520015717,  342.7558400630951,  312.5014588832855,
          718.0520720481873,  1022.4128830432892, 708.0735518932343,
          1098.051519870758]

Promedio tiempos: 650.327194476

Prueba 3:
ND = 19.
NS = 32.
Adiciones = 32.
Duplicaciones = 0.
Hash: xxhash de la concatenación de las coordenadas (x,y) modulo NS.
Criterio punto distinguido: Últimos ND bits del xxhash de la coordenada x son
cero.
Tipo de caminata: Aditiva.
Coordenadas: Afines.

Se calculaban 32 puntos de forma concurrente.

Pasos realizados: [62827587, 58577475, 33483932, 70661330,
                   49119440, 63556911, 58989175, 61991642,
                   45073963, 84040265]

Promedio pasos realizados: 58832172.0

Puntos distinguidos: [119, 104, 80, 133, 75, 89, 105, 107, 77, 162]
Promedio puntos: 105.1
```

```
L: [1.9144075326210983, 1.7849031728995746, 1.0202825654051084,
    2.1531080354403103, 1.4967091754475637, 1.936630626424165,
    1.7974480058115898, 1.888935610472362,  1.3734402101466139,
    2.560775035964269]

Promedio: 1.79266399706
Diferencia L0: -0.376663997063

Tiempo: [1342.1151537895203, 1258.7162220478058, 714.0713651180267,
         1553.479295015335,  1078.6067850589752, 1388.8105487823486,
         1297.4684348106384, 1339.9536118507385, 964.4492571353912,
         1815.5586109161377]

Promedio tiempos: 1275.32292845

Prueba 4:
ND = 19.
NS = 32.
Adiciones = 32.
Duplicaciones = 0.
Hash: xxhash de la concatenación de las coordenadas (x,y) modulo NS.
Criterio punto distinguido: Últimos ND bits del xxhash de la coordenada x son
cero.
Tipo de caminata: Aditiva.
Coordenadas: Afines.

Pasos realizados: [67309936, 2756441, 48664414, 55791188,
                   41508481, 49033790, 48934768, 79196514,
                   21437421, 25283138]

Promedio pasos realizados: 43991609.1

Puntos distinguidos: [120, 3, 80, 105, 68, 98, 91, 137, 50, 43]
Promedio puntos: 79.5

L: [2.050988342089981,  0.0839909928997533, 1.4828441641757089,
    1.700002748173025,  1.2647970817987921, 1.4940993504805633,
    1.4910820698281135, 2.4131820144378975, 0.6532156048324713,
    0.7703977209167484]

Promedio L: 1.34046000896
Diferencia L0: 0.0755399910367

Tiempo: [1415.645150899887,  56.87382698059082, 1041.893405199051,
         1188.9193441867828, 887.0573468208313, 1044.0265679359436,
         1044.799882888794,  1686.125818014145, 458.3987669944763,
```

```
         540.8674051761627]

Promedio tiempos: 936.46075151

Prueba 5:
ND = 19.
NS = 32.
Adiciones = 32.
Duplicaciones = 0.
Hash: xxhash de la concatenación de las coordenadas (x,y) modulo NS.
Criterio punto distinguido: Últimos ND bits del xxhash de la coordenada x son
cero.
Tipo de caminata: Aditiva.
Coordenadas: Proyectivas.

Pasos realizados: [64274144, 55972924, 69184955, 27158741,
                   34496855, 26731871, 32004809, 11135944,
                   42384999, 27115892]

Promedio pasos realizados: 39046113.4

Puntos distinguidos: [118, 110, 120, 57, 58, 46, 58, 24, 65, 48]
Promedio puntos: 70.4

L: [1.9584852976507463, 1.705540391491213, 2.1081216917665757,
    0.8275488655470002, 1.051147150753025, 0.8145417904312557,
    0.9752124879425899, 0.3393212455612329,1.291505296164529,
    0.8262432217640345]

Promedio L: 1.18976674391
Diferencia L0: 0.226233256093

Tiempo: [1742.8021280765533, 1472.1845998764038, 1828.1252889633179,
         712.8097970485687,  902.787006855011,   704.8001050949097,
         838.0032780170441,  291.72630500793457, 1118.3706469535828,
         712.741837978363]

Promedio tiempos: 1032.43509939

Prueba 6:
ND = 19.
NS = 32.
Adiciones = 32.
Duplicaciones = 0.
Hash: xxhash de la concatenación de las coordenadas (x,y) modulo NS.
Criterio punto distinguido: Últimos ND bits del xxhash de la coordenada x son
```

```
cero.
Tipo de caminata: Aditiva.
Coordenadas: Proyectivas.

Se calculaban 32 puntos de forma concurrente.
Se utiliza el algoritmo para inversiones múltiples para invertir 32 números al mismo tiempo.


Pasos realizados: [50105002, 60480568, 37215852, 59357585,
                   71977807, 42329185, 77385293, 17247396,
                   98442687, 30314320]

Promedio pasos realizados: 54485569.5

Puntos distinguidos: [114, 109, 77, 113, 137, 90, 142, 35, 173, 47]
Promedio puntos: 103.7

L: [1.52674004893416,   1.8428919601257732, 1.1339971946035738,
    1.8086737573129636, 2.1932221573677118, 1.2898045983162143,
    2.357992641565094,  0.5255421447349076, 2.9996285154841464,
    0.9237019170302754]

Promedio: 1.66021949355
Diferencia L0: -0.244219493547

Tiempo: [1540.7452638149261, 1827.4921131134033, 1123.7595670223236,
         1798.6115171909332, 2200.7707738876343, 1312.6894781589508,
         2373.156543970108,  530.8403239250183,  2956.032433986664,
         916.3773698806763]

Promedio tiempos: 1658.0475385

Prueba 7:
ND = 19.
NS = 23.
Adiciones = 18.
Duplicaciones = 5.
Hash: Coordenada x módulo NS.
Criterio punto distinguido: Ultimos ND bits de la coordenada x son cero.
Tipo de caminata: Mixta.
Tipo de coordenadas: Afines.

Pasos realizados: [31919644, 33209072, 49183869, 33095306, 29048000,
                   48640754, 62073273, 68466795, 39178379, 51166181]

Promedio pasos realizados: 44598127.3
```

```
Puntos distinguidos: [34, 29, 42, 29, 29, 46, 56, 73, 42, 51]
Promedio puntos: 43.1

L: [0.9726174413189519,1.0119073582780824,1.4986723793331316,
    1.0084408159874136,0.8851161195730414,1.482123224786109,
    1.891422973249726, 2.086238774098145, 1.1937969840141123,
    1.5590750337404253]

Promedio L: 1.35894111044
Diferencia L0: 0.0570588895621

Tiempo: [417.31861901283264,433.57844495773315,652.7477939128876,
         437.0846881866455, 386.9354019165039, 647.5778369903564,
         823.7150390148163, 906.299211025238,  512.3107421398163,
         672.1387231349945]

Promedio tiempos: 588.970650029
```

Los mejores tiempos fueron observados en las pruebas $1, 2$ y 7. Por lo que se seleccionaron los tipos de caminatas, función hash, coordenadas y criterio para los puntos distinguidos utilizados en estas pruebas para intentar resolver Eccp-79 en un menor tiempo.

5.9. Otro intento al reto Eccp-79

Utilizando los resultados de las pruebas anteriores se realizaron los siguientes cambios al código para resolver el reto Eccp-79

- Los puntos distinguidos, son aquellos cuyos últimos 29 bits son cero. Ya no se aplica la función xxhash.
- El tamaño de partición es un número primo grande, 2003 con 1603 Adiciones y 400 Duplicaciones.
- La función partición consiste en la coordenada x modulo 2003. Ya no se aplica la función xxhash.

El mejor resultado obtenido, después de 3 días, 22 horas y 56 minutos de operaciones ininterrumpidas, se llego al resultado correcto de

$$92221507219705345685350(\text{mód } 466597814831947642887217).$$

Se encontraron 550 puntos distinguidos y se realizo un total de $595,743,548,260$ pasos de la caminata ρ.

Capítulo 6

Conclusiones

En este trabajo resolvimos Eccp-79 utilizando el algoritmo distribuido de ρ de Pollard con puntos distinguidos. Después de haber hecho algunas pruebas en curvas más pequeñas se logro escribir un programa que alcanzó como mejor tiempo 3 días, 22 horas y 57 minutos. Esto se logro utilizando una red de tres computadoras Intel quad core i5-2500 3.30GHz con 4GB de memoria RAM trabajando en forma distribuida.

Al realizar este trabajo se nos presentaron varias complicaciones en software y principalmente en hardware. Las diversas pruebas realizadas durante el transcurso de este trabajo fueron en curvas sobre un campo finito de orden menor al presentado en Eccp-79. Esto debido al tiempo que tomaba realizar cada prueba sobre dicho reto. Sin embargo, se espera que los programas presentados sean útiles en futuros trabajos en el área.

6.1. Trabajo futuro

Quedan muchas áreas que explorar, usar el algoritmo con un mayor número de equipos, sería un buen comienzo. Realizar un mayor número de pruebas para diferentes tamaños de partición, al igual que para diferentes funciones iteradoras en la caminata ρ. La creación de una plataforma que permita resolver el problema del logaritmo discreto de forma distribuida a través de Internet o la utilización de otros tipos de hardware como procesamiento en paralelo con procesadores gráficos GPU o FPGA, sería una forma de extender el presente trabajo. De igual forma es posible considerar curvas binarias, o curvas con un campo finito de orden mayor al presentado Eccp-79.

6.2. Eccp-89

La curva Eccp-89, otra de las curvas presentadas por Certicom, posee un grado de dificultad mayor al de la curva Eccp-79 pues trabaja sobre un campo finito

de 89 bits de tamaño.

Utilizando el mismo software y equipo de computo que se uso en el transcurso de este trabajo de tesis fue posible resolver el reto Eccp-89 en 68 días, 5 horas y 47 minutos de operaciones ininterrumpidas, se llego al resultado correcto de

$$333373190151749761757285479(\text{mód } 41636331555612445828589498 3).$$

Se encontraron 4866 Puntos distinguidos y se realizo un total de $10,398,392,652,653$ pasos de la caminata ρ.

Bibliografía

[BZ10] Richard P Brent and Paul Zimmermann, **Modern computer arithmetic**, vol. 18, Cambridge University Press, 2010.

[Cer02] Certicom ECC Certicom, **Challenge, 2009**, Certicom Research (2002).

[ESST99] Adrian E. Escott, John C. Sager, Alexander P. L. Selkirk, and Dimitrios Tsapakidis, **Attacking elliptic curve cryptosystems using the parallel pollard rho method**, CryptoBytes Technical Newsletter **4** (1999), no. 2, 15–19.

[Gal12] Steven D Galbraith, **Mathematics of public key cryptography**, Cambridge University Press, 2012.

[HMV06] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone, **Guide to elliptic curve cryptography**, Springer Science & Business Media, 2006.

[HPS08] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman, **An introduction to mathematical cryptography**, Springer, 2008.

[KKM11] Ann Hibner Koblitz, Neal Koblitz, and Alfred Menezes, **Elliptic curve cryptography: The serpentine course of a paradigm shift**, Journal of Number Theory **131** (2011), no. 5, 781–814.

[Lid97] Rudolf Lidl, **Finite fields**, vol. 20, Cambridge University Press, 1997.

[MR12] Alfonso Manuel Hernández Magdaleno and Alonso Castillo Ramírez, **Álgebra moderna: anillos y campos**, Editorial Universitaria, 2012.

[MVOV10] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone, **Handbook of applied cryptography**, CRC press, 2010.

[Ple11] Vera Pless, **Introduction to the theory of error-correcting codes**, vol. 48, John Wiley & Sons, 2011.

[PPP09] Bart Preneel, Christof Paar, and Jan Pelzl, **Understanding cryptography: a textbook for students and practitioners**, Springer, 2009.

[Pre96] Oliver Pretzel, **Error-correcting codes and finite fields (student ed.)**, Oxford University Press, Inc., 1996.

[Tes01] Edlyn Teske, **On random walks for pollard's rho method**, Mathematics of computation **70** (2001), no. 234, 809–825.

[VOW94] Paul C Van Oorschot and Michael J Wiener, **Parallel collision search with application to hash functions and discrete logarithms**, Proceedings of the 2nd ACM Conference on Computer and communications security, ACM, 1994, pp. 210–218.

[Was08] Lawrence C Washington, **Elliptic curves: number theory and cryptography**, CRC press, 2008.

Apéndice A

Rho de Pollard con detección de ciclos de Floyd: Codigo Python

```
import sys,time

import gmpy2 as gmp
import xxhash as xhash

#Parametros Curva Eliptica
P  = gmp.mpz(sys.argv[1])
A  = gmp.mpz(sys.argv[2])
B  = gmp.mpz(sys.argv[3])
px = gmp.mpz(sys.argv[4])
py = gmp.mpz(sys.argv[5])
qx = gmp.mpz(sys.argv[6])
qy = gmp.mpz(sys.argv[7])
N  = gmp.mpz(sys.argv[8])

#Parametros Pollard
NS = 32
G = []
U = []
V = []

def add(p1,p2):
    (x1,y1,inf1) = p1
    (x2,y2,inf2) = p2
    np = (0,0,True)
```

```
    if (inf1 or inf2):
        if inf1:
            np = p2
        else:
            np = p1
    elif x1 != x2:
        m1 = gmp.f_mod(y2-y1,P)
        m2 = gmp.invert(x2-x1,P)
        m = gmp.f_mod(m1 * m2,P)
        x3 = gmp.f_mod(m**2 -x1-x2,P)
        y3 = gmp.f_mod(m * (x1-x3)-y1,P)
        np = (x3,y3,False)
    return np

def double(p):
    (x,y,inf) = p
    np = (0,0,True)
    if y != 0 and not inf:
        m1 = gmp.f_mod(3*x**2 + A,P)
        m2 = gmp.invert(2*y,P)
        m = gmp.f_mod(m1 * m2,P)
        x3 = gmp.f_mod(m**2 - 2*x,P)
        y3 = gmp.f_mod(m*(x-x3)-y,P)
        np = (x3,y3,False)
    return np

def scalar_mult(k,p):
    a = k
    b = (0,0,True)
    c = p
    while a != 0:
        (q,r) = gmp.f_divmod(a,2)
        if r == 0:
            a = q
            c = double(c)
        else:
            a = a-1
            b = add(b,c)
    return b

def s(p):
    x = p[0]
    h = gmp.mpz(xhash.xxh64(str(x)))
    return gmp.f_mod(h,NS)

def rho_walk(w):
```

```
    p_i,a,b = w
    j = s(p_i)
    np = (0,0,True)
    if j == 0:
        np = double(p_i)
        a = gmp.f_mod(2*a,N)
        b = gmp.f_mod(2*b,N)
    else:
        np = add(p_i,G[j])
        a = gmp.f_mod(a + U[j],N)
        b = gmp.f_mod(b + V[j],N)
    return (np,a,b)

def precompute_G(p,q):
    state = gmp.random_state()
    for i in xrange(NS):
        u = gmp.mpz_random(state,N)
        v = gmp.mpz_random(state,N)
        g = add(scalar_mult(u,p),scalar_mult(v,q))
        G.append(g)
        U.append(u)
        V.append(v)

def rho(p,q):
    precompute_G(p,q)
    x1 = p
    a1 = 1
    b1 = 0
    (x2,a2,b2) = rho_walk((x1,a1,b1))
    while x1 != x2:
        (x1,a1,b1) = rho_walk((x1,a1,b1))
        (x2,a2,b2) = rho_walk(rho_walk((x2,a2,b2)))
    if b1 == b2:
        return 'Fallo'
    else:
        b_inv = gmp.invert((b1-b2),N)
        dlp = gmp.f_mod((a2-a1)*b_inv,N)
        dlp = gmp.f_mod(dlp, N)
        return dlp

p = (px, py, False)
q = (qx, qy, False)

start = time.time()
r = rho(p,q)
end = time.time()
```

```
elapsed = end - start
print r, elapsed
```

Apéndice B

Rho de Pollard con detección de ciclos de Floyd: Codigo Sage

```
import sys, time

def do(f,A,B,(px,py),(qx,qy),order):
    F = GF(f)
    curve_params = [A, B]
    e = EllipticCurve(F,curve_params)
    base_p = (px,py)
    print 'Base: ' + str(base_p)
    public_p = (qx,qy)
    print 'Public: ' + str(public_p)
    base = e(base_p)
    public = e(public_p)
    #r = discrete_log_rho(public,base,ord=order,operation='+')
    return r

f  = int(sys.argv[1])
A  = int(sys.argv[2])
B  = int(sys.argv[3])
px = int(sys.argv[4])
py = int(sys.argv[5])
qx = int(sys.argv[6])
qy = int(sys.argv[7])
n  = int(sys.argv[8])
```

```
start = time.time()
r = do(f,A,B,(px,py),(qx,qy),n)
end = time.time()
elapsed = end -start

print r, elapsed
```

Apéndice C

Rho de Pollard con puntos distinguidos: Codigo Python

C.1. Servidor

```
import sys, json, math, time

import leveldb
import xxhash as xhash
import gmpy2 as gmp
import zmq

#Parametros ZMQ

address = 'tcp://127.0.0.1'
#address = 'tcp://148.209.231.73'
#address = 'tcp://192.168.0.5'
port_cli = '5555'
full_address_cli = address + ":" + port_cli
context = zmq.Context()

#Socket servidor-cliente
sock = context.socket(zmq.REP)
sock.bind(full_address_cli)

#GMP

#Parametros Curva Eliptica
P  = int(sys.argv[1])
A  = int(sys.argv[2])
```

```
B  = int(sys.argv[3])
px = int(sys.argv[4])
py = int(sys.argv[5])
qx = int(sys.argv[6])
qy = int(sys.argv[7])
N  = int(sys.argv[8])

#Parametros Pollard
NS = 64

#Main

disp_found = 0 # contador de puntos distinguidos.

#Calculando los valores para la funcion rho walk esta debe ser la misma
#para todos los clientes.
seed_str = "La libertad no necesita alas, lo que necesita es echar raices"
seed = xhash.xxh64(str(seed_str))

#Empaquetando
pack = json.dumps([NS,seed,{'P':P,'A':A,'B':B,'px':px,'py':py,'qx':qx,
                            'qy':qy,'N':N}])
dic = {'data':pack}

#Base de datos
leveldb.DestroyDB('dis_points.db')
db = leveldb.LevelDB('dis_points.db')

done = False
#Comandos
while True:
    raw = sock.recv_json()
    msg_raw = json.loads(raw)
    msg = msg_raw[0]
    data = msg_raw[1]
    if msg in dic.keys():
            sock.send_json(dic[msg])
    elif msg == 'point':
        if done:
            sock.send_json(json.dumps('Done'))
        else:
            key = data[0]
            value = data[1]
            try:
                val = None
                val = db.Get(key)
```

```
                    if val is not None:
                        vall = val.split(':')
                        valuel = value.split(':')
                        if valuel[1] == vall[1]:
                            print 'Fallo: ' + str(val) + ' - ' + str(value)
                        else:
                            print 'Found: ' + str(val) + ' - ' + str(value)
                            a1 = gmp.mpz(vall[0])
                            b1 = gmp.mpz(vall[1])
                            a2 = gmp.mpz(valuel[0])
                            b2 = gmp.mpz(valuel[1])
                            b_inv = gmp.invert((b1-b2),N)
                            dlp = gmp.f_mod((a2-a1)*b_inv,N)
                            print 'DLP Encontrado: ' + str(dlp)
                            print str(disp_found) + ' Puntos distinguidos'
                            sock.send_json(json.dumps('Done'))
                            done = True
                            continue
                except KeyError:
                    db.Put(key, value)
                    disp_found += 1
                sock.send_json(json.dumps('Punto agregado'))
        elif msg == 'end':
            sock.send_json(json.dumps('ACK'))
            break
        else:
            sock.send_json(json.dumps('NO'))

print 'Terminado'
```

C.2. Cliente

```
import json, math, multiprocessing, random, time

import zmq
import gmpy2 as gmp
import xxhash as xhash

#Parametros multiprocessing
PR = 4

#Parametros rho
G = []
U = []
V = []

#Funciones

def add(p1,p2):
    (x1,y1,inf1) = p1
    (x2,y2,inf2) = p2
    np = (0,0,True)
    if (inf1 or inf2):
        if inf1:
            np = p2
        else:
            np = p1
    elif x1 != x2:
        m1 = gmp.f_mod(y2-y1,P)
        m2 = gmp.invert(x2-x1,P)
        m = gmp.f_mod(m1 * m2,P)
        x3 = gmp.f_mod(m**2 -x1-x2,P)
        y3 = gmp.f_mod(m * (x1-x3)-y1,P)
        np = (x3,y3,False)
    return np

def double(p):
    (x,y,inf) = p
    np = (0,0,True)
    if y != 0 and not inf:
        m1 = gmp.f_mod(3*x**2 + A,P)
        m2 = gmp.invert(2*y,P)
        m = gmp.f_mod(m1 * m2,P)
        x3 = gmp.f_mod(m**2 - 2*x,P)
```

```
            y3 = gmp.f_mod(m*(x-x3)-y,P)
            np = (x3,y3,False)
    return np

def scalar_mult(k,p):
    a = k
    b = (0,0,True)
    c = p
    while a != 0:
        (q,r) = gmp.f_divmod(a,2)
        if r == 0:
            a = q
            c = double(c)
        else:
            a = a-1
            b = add(b,c)
    return b

#No en uso por el momento
def montgomery_inversion(nums):
    z = []
    z.append(nums[0])
    for i in range(1,len(nums)):
        zi = z[i-1]
        z.append(gmp.f_mod(zi*nums[i],P))
    q = gmp.invert(z[-1],P)
    y = []
    for i in range(1,len(nums)):
        zi = z[-(i+1)]
        y.append(gmp.f_mod(q*zi,P))
        q = gmp.f_mod(q*nums[len(nums)-i],P)
    y.append(q)
    return y[::-1]

def s(p):
    x = p[0]
    y = p[1]
    c = str(x) + str(y)
    h = gmp.mpz(xhash.xxh64(c))
    return gmp.f_mod(h,NS)

def rho_walk_add(w):
    p_i,a,b = w
    j = s(p_i)
    np = add(p_i,G[j])
    a = gmp.f_mod(a + U[j],N)
```

```
    b = gmp.f_mod(b + V[j],N)
    return (np,a,b)

def rho_walk(w):
    p_i,a,b = w
    j = s(p_i)
    np = (0,0,True)
    if j == 0:
        np = double(p_i)
        a = gmp.f_mod(2*a,N)
        b = gmp.f_mod(2*b,N)
    else:
        np = add(p_i,G[j])
        a = gmp.f_mod(a + U[j],N)
        b = gmp.f_mod(b + V[j],N)
    return (np,a,b)

def precompute_G(p,q,state):
    for i in xrange(NS):
        u = gmp.mpz_random(state,N)
        v = gmp.mpz_random(state,N)
        g = add(scalar_mult(u,p),scalar_mult(v,q))
        G.append(g)
        U.append(u)
        V.append(v)

def is_dis_point(point):
    if point[2]:
        return False
    h = gmp.mpz(xhash.xxh64(str(point[0])))
    if gmp.f_mod(h,2**ND)==0:
        return True
    return False

def send_point(point,pointsq):
    p = point[0]
    a = point[1]
    b = point[2]
    x = str(p[0])
    dat = str(a) + ':' + str(b)
    pointsq.put([x,dat])

def rho_client(state,p,q,endq,pointsq,iden):
    done = False
    op = 0
    while not done:
```

```
            a = gmp.mpz_random(state,N)
            b = gmp.mpz_random(state,N)
            x = add(scalar_mult(a,p),scalar_mult(b,q))
            while is_dis_point(x) == False:
                if not endq.empty() and endq.get_nowait():
                    done = True
                    break
                (x,a,b) = rho_walk_add((x,a,b))
                op += 1
            if not done:
                send_point((x,a,b),pointsq)
        print 'Operaciones ' + str(iden) + ': ' + str(op)
        print 'Cliente ' + str(iden) + ' terminado'

    def rho_server(p,q,seed):
        state = gmp.random_state(seed)
        precompute_G(p,q,state)

        pointsq = multiprocessing.Queue()

        jobs = []
        job_endq = []
        rand = random.SystemRandom()
        iden = 0
        for i in xrange(PR):
            nseed = rand.randint(1,N)
            s = gmp.random_state(nseed)
            endq = multiprocessing.Queue(1)
            job = multiprocessing.Process(target=rho_client,
                                          args=(s,p,q,endq,pointsq,iden))
            jobs.append(job)
            job_endq.append(endq)
            iden += 1

        start = time.time()
        for job in jobs:
            job.start()

        while True:
            point = pointsq.get()
            msg = ['point', point]
            sock.send_json(json.dumps(msg))
            raw = sock.recv_json()
            data = json.loads(raw)
            if data == 'Done':
                for endq in job_endq:
```

```
                endq.put(True)
            break

    end = time.time()
    elapsed = end - start

    #Cerrando queues
    pointsq.close()
    pointsq.join_thread()

    #Cerrando Procesos
    for i in range(len(jobs)):
        job = jobs[i]
        endq = job_endq[i]
        endq.close()
        endq.join_thread()
        job.join()

    return elapsed

def send_msg(msg):
    msg = [msg, 'dummy']
    sock.send_json(json.dumps(msg))

#Parametros ZMQ

address = 'tcp://127.0.0.1'
#address = 'tcp://148.209.231.73'
#address = 'tcp://192.168.0.5'
port_cli = '5555'
full_address_cli = address + ":" + port_cli

context = zmq.Context()
sock = context.socket(zmq.REQ)
sock.connect(full_address_cli)

#Main

send_msg('data')
raw = sock.recv_json()
data = json.loads(raw)

NS = data[0]
seed = data[1]
data_curve = data[2]
```

```
P  = gmp.mpz(data_curve['P'])
A  = gmp.mpz(data_curve['A'])
B  = gmp.mpz(data_curve['B'])
px = gmp.mpz(data_curve['px'])
py = gmp.mpz(data_curve['py'])
qx = gmp.mpz(data_curve['qx'])
qy = gmp.mpz(data_curve['qy'])
N  = gmp.mpz(data_curve['N'])

ND = 29
p = (px,py,False)
q = (qx,qy,False)
t = rho_server(p,q,seed)
print 'Tiempo: ' + str(t) + ' segundos.'
print 'Terminado Clientes'
```