# AUTENTICACIÓN DE DISPOSITIVOS MÓVILES BASADA EN CRIPTOGRAFÍA DE CURVAS ELÍPTICAS

**TESIS**

Que para obtener el grado de

**MAESTRÍA EN CIENCIAS EN INGENIERÍA ELÉCTRICA**

presenta

**Esteban Rubén Vera Aguilar**

**Dr. Leonardo Romero**

**Director de Tesis**

**Dr. Juan Manuel García García**

**Co-Director de Tesis**

Universidad Michoacana de San Nicolás de Hidalgo

Noviembre 2007

# Dedicatoria

Para mi fuente de motivación, mis padres: Ramona y Rubén
y hermanos: Ana, María, Miriam y Carlos.
Con mucho cariño y amor.

# Agradecimientos

A Dios, gracias.

A mis padres y hermanos.

A mis maestros.

Al profesor: Ing. Efrain Villanueva Chávez, gracias.

A la Universidad Michoacana de San Nicolás de Hidalgo y al Consejo Nacional de Ciencia y Tecnología, por el apoyo otorgado para realizar estos estudios.

# Resumen

Las redes inalámbricas móviles ad hoc, se han convertido en una tecnología importante en la actualidad a causa del rápido aumento de los dispositivos inalámbricos. Las redes inalámbricas móviles presentan una serie de problemas de seguridad, debido algunos factores, como son: el medio abierto que utilizan para la comunicación, cambios en la topología de la red de manera constante, falta de un punto de administración y el de un monitoreo centralizado, entre otros. Como primera línea de defensa para posibles ataques, esta el uso de la autenticación y el software para codificar.

A pesar de que los nuevos dispositivos móviles cuentan con un poder de cómputo muy aceptable, los modelos tradicionales de autenticación y el software de codificación se consideran computacionalmente costosos para los dispositivos pequeños.

En este documento, se desarrolló un protocolo de autenticación para dispositivos móviles basado en criptografía de curvas elípticas (ECC). Se revisa el rendimiento de la generación de claves usando ECC para 163-bits, 283-bits, 409-bits y 571-bits sobre curvas de Koblitz y RSA-1024, RSA-3072, RSA-7680 y RSA-15360 bits de longitud en un asistente personal digital (PDA).

De la implementación y evaluación se observa que ECC es viable para dispositivos pequeños que integran una red inalámbrica móvil ad hoc. Sobre la PDA Dell Axim X5 con 300 MHz de velocidad, se midió 0,8seg para generar una clave de 163-bits usando ECC y 23seg para RSA-1024. Además, conforme se va incrementando el tamaño de las claves, el buen rendimiento de ECC se refleja con mayor notoriedad que con RSA.

# Abstract

Wireless mobile networks ad hoc have become an important technology nowadays due to the fast growing use of wireless devices. Wireless mobile networks, however, have still certain deficiencies in security. The following factors are the cause to these deficiencies: wireless mobile networks use an open environment for communication. There are constant changes in the topology of a network. Wireless mobile networks also lack of management and centralized monitoring. Authentication and decoding software are used to prevent possible threats the wireless mobile networks.

In spite of the very acceptable computerized software that new mobile devices come with, the traditional models for authentication and decoding software are considered computationally expensive for small devices.

This document contains a protocol of authentication for mobile devices based on elliptic curves cryptography (ECC). The document also includes an observation of the performance for key generation using ECC for 123-bits, 283-bits, 409-bits on Koblitz curves and RSA 1024, 3072, 7680 and 15360 long bits on personal digital assistant (PDA).

The implementation and evaluation show that ECC is viable for small devices which comprise a wireless mobile network ad hoc. PDA Dell Axim X5 at 300 MHz measured 0.8s to generate a key of 163 bits using ECC and 23s for RSA of 1024 long bits. Furthermore, as the key size was being incremented, the performance of ECC outweighed the RSA.

# Índice general

# Índice de figuras

# Índice de cuadros

# Lista de Algoritmos

# Lista de Símbolos

| | |
|---|---|
| $F$ | Campo finito. |
| $F_2$ | Campo característico 2. |
| $F_{2^m}$ | Campo de extensión. |
| $E$ | Curva elíptica. |
| $\#E(F_q)$ | Orden de una curva elíptica. |
| $\triangle$ | Discriminante de la curva elíptica. |
| $\mathcal{O}$ | Punto al infinito. |
| $\lambda$ | Pendiente de la recta. |
| \| | Divisor de paquete. |
| # | Divisor del campo *info* del paquete. |
| k | Valor de un entero positivo. |
| h | Cofactor. |
| n | Orden de un punto base. |
| BNO | Base Normal Óptima. |
| BNG | Base Normal Gaussiana. |
| MCD | Máximo Común Divisor. |
| FNW | Forma Normal de Weierstrass. |
| FR | Representación del campo. |
| $\oplus$ | Símbolo de la operación aditiva. |
| $\ominus$ | Símbolo del inverso aditivo. |

Capítulo 1

# Introducción

Esta tesis se enfoca en el desarrollo de un protocolo de comunicación de acuerdo de clave compartida, siguiendo el modelo del protocolo de acuerdo de claves conocido como Diffie-Hellman, para ser utilizado en dispositivos que cuentan con recursos de hardware limitados, empleando para esto criptografía de curvas elípticas (ECC del inglés *"Elliptic Curve Cryptography"*).

En este primer capítulo se mencionarán los motivos por los cuales se realizó este trabajo. Además, se da una reseña de los trabajos previos que se han realizado. Tratará también, los objetivos que se pretenden, sus alcances y contribuciones. Por último, se explicará la manera como esta organizado este documento.

## 1.1. Motivación

Existe actualmente dos tipos de tecnologías en cuanto a las redes inalámbricas: redes inalámbricas con infraestructura y redes inalámbricas *ad-hoc*. Las redes inalámbricas con infraestructura son aquellas que cuentan con puntos de acceso (AP del inglés *"Access Points"*), en donde los nodos que integran o desean integrar dicha red, utilizan como medio de administración este tipo de dispositivos. Sin embargo, las redes *ad-hoc* tienen la peculiaridad de no contar con un punto que administre la comunicación entre los nodos [anHuang03], como sí es el caso de las redes inalámbricas con infraestructura. Es decir, que son los propios dispositivos los que se encargan de enlazarse unos con otros y fungir a la vez como enrutador que dirigen o direccionan la información entre los nodos.

Las redes inalámbricas en general, presentan grandes restricciones entre las que se

pueden citar (ver [Zhang03, Arbaugh]):

- Ancho de banda bajo.
- Porcentaje muy grande de latencia.
- Impredecible disponibilidad.
- Poca estabilidad.

Tanto las redes inalámbricas con infraestructura, como las redes *ad-hoc*, son más vulnerables para ser atacadas por personas maliciosas que las redes convencionales (redes con cables), debido a que tienen un medio de comunicación totalmente abierto. Muy particularmente, las redes inalámbricas *ad-hoc* presentan las siguientes debilidades, como es: no contar con un monitoreo centralizado, no tener algún punto de administración, contar con algoritmos cooperativos y un constante cambio en la topología debido a la movilidad que presentan los nodos que la integran.

Como una primera línea defensiva contra ataques a las redes inalámbricas, se encuentra la **autenticación** y la **criptografía** [Hu02, Yang02, Vigna04, Sanzgiri05].

Al momento de desarrollar alguna medida de seguridad para las redes inalámbricas, aparte de considerar las restricciones con las que cuenta este tipo de redes, igualmente se debe de considerar los dispositivos que integran dichas redes.

Por lo regular, los elementos que integran las redes inalámbricas pueden ser: computadoras personales portátiles (del inglés, *"PC laptop"*), *pagers*[1], asistentes personales digitales (PDA, del inglés, *"Personal Digital Assistant"*), teléfonos móviles (teléfonos celulares con IP), etc.. A excepción de las computadoras personales portátiles, los demás dispositivos presentan algunas características en común: tamaños de las pantallas muy pequeñas o falta de capacidades gráficas, procesadores muy limitados, poca memoria y baterías que permiten un tiempo de uso muy reducido.

Si se implementa algún sistema de autenticación utilizando los métodos criptográficos tradicionales, para resguardar a las redes inalámbricas, se requiere un consumo considerable de los recursos tanto de la red, como los dispositivos, convirtiéndose en un sistema de seguridad poco confiable. Así pues, en este trabajo se desarrolló un protocolo el cual

[1] Un *pager* es un radio que permite recibir mensajes emitidos en una frecuencia sobre una red especial de estaciones bases de radio frecuencia. Estas contienen una pequeña pantalla que puede mostrar un mensaje de texto o un número de teléfono.

permite establecer una clave compartida usando ECC, para reducir el alto consumo de los recursos de los dispositivos móviles, manteniendo un nivel de seguridad más confiable.

El establecimiento de una clave compartida, es el proceso en el cual dos o más entidades comparten de manera secreta un código. Esta clave, subsecuentemente, es utilizada para alcanzar algún objetivo criptográfico, tal como la integridad o confidencialidad en los datos.

## 1.2. Antecedentes

A pesar de la teoría que abarca las curvas elípticas, resulta ser hasta cierto punto difícil de digerir. Han surgido innumerables artículos y libros que tratan exclusivamente de las curvas elípticas, existiendo una gran variedad, desde aquellos dirigidos a personas con poco o nulo conocimiento hacia el tema, hasta aquellos donde se requiere como mínimo un curso de nivel avanzado.

El tema de curvas elípticas ha sido en los últimos años, la moda para la criptografía. Desde que surgió la propuesta de Víctor Miller en 1986 y más tarde de Neal Koblitz, en 1987 [Koblitz87], en utilizar las curvas elípticas en la criptografía (ver también [Charlap88]), se despertó el interés de las personas dedicadas a la seguridad como: [Qizhi04, Koblitz04, Torii00], en el estudio de este modelo criptográfico. Con ECC se pueden tener longitudes de claves más pequeñas (de hasta 163-bits) que pueden lograr la misma seguridad que los tradicionales sistemas criptográficos que utilizan longitudes de hasta 1024-bits, véase la tabla 1.1.

Por ejemplo, si se usa un sistema RSA (RSA del inglés *"R. Rivest, A. Shamir y L. Adleman"*) en un equipo con suficientes recursos (v.gr. computadoras personales de escritorio) tal vez no se note la diferencia. Pero en cambio, en dispositivos que cuentan con restricciones de potencia, de procesamiento, de memoria, etc., se prefiere ECC. En [Wollinger04] hacen una investigación acerca del rendimiento que tiene ECC respecto a restricciones en cuanto a procesadores, recursos; memoria y consumo de energía, y diferentes arquitecturas. En [Al-Kalayi04] se muestra una propuesta por demás interesante, ya que implementan ECC sobre tarjetas inteligentes (*"Smart Cards"*), satisfaciendo los requerimientos en términos de memoria, procesamiento y costo que presentan este tipo de tecnología.

La criptografía de curvas elípticas esta siendo parte de las herramientas criptográficas como lo propone [iTomé04]. Así, M. Brown en [Brown00] propone la idea de integrar

ECC como parte de una herramienta conocida como PGP. Además, menciona la experiencia en portar esta herramienta a un dispositivo restringido de recursos, que es el *"pager"* RIM[2], como en un asistente personal digital (PalmPilot).

| Tamaño de la clave ECC (Bits) | Tamaño de la clave RSA (Bits) |
|---|---|
| 163 | 1024 |
| 233 | 3072 |
| 409 | 7680 |
| 571 | 15360 |

Cuadro 1.1: Equivalencias en cuanto al tamaño de claves entre ECC y RSA, [Belingueres].

Para la generación de claves, han surgido varias propuestas. Por ejemplo [Memon05] se introduce un subconjunto aleatorio precargado de cadenas *hash* que es un esquema de pre-distribución de claves altamente escalable, que emplea criptografía simétrica. Otras, ofrecen un sistema que administra las claves que utilizarán los nodos contando con algún servicio centralizado [Capkun03].

En [Cagalj05] presenta un conjunto de técnicas muy simples para el establecimiento de claves en redes inalámbricas. Se basa, principalmente, en el protocolo de acuerdo de claves conocido como **Diffie-Hellman** [Diffie76]. En [Law03, Strangio05] desarrollan un protocolo que mejora el propuesto por Diffie-Hellman, ya que el acuerdo de claves para la autenticación se realiza en dos pasos. Además, lo hace más eficiente puesto que usa ECC.

## 1.3. Objetivo de la tesis

Los objetivos principales de esta tesis son los que a continuación se muestran:

- Implementación de un protocolo de comunicación para acuerdo de clave compartida.

[2]El modelo que utilizaron, contaba con un procesador Intel 386 a una velocidad de $10MHz$, con $2MB$ de memoria flash y $304KB$ de SRAM.

- Empleo de criptografía de curvas elípticas para la generación de clave compartida.
- Implementación del protocolo sobre un dispositivo inalámbrico (asistente personal digital).
- Comparación de eficiencia en computadora personal de escritorio y en el asistente personal digital.
- Comparación de eficiencia para la generación de una clave compartida, utilizando ECC contra RSA, sobre una PDA.

## 1.4. Aportaciones

Cabe destacar las aportaciones que de este trabajo se desprenden. Mencionando aquellas que son de mayor trascendencia, como son:

- Implementación y desarrollar un protocolo de autenticación usando ECC.
- Algoritmo de generación de clave compartida.
- Portar librería NTL y ECC a un asistente personal digital.
- Instalación y configuración de compilador multi-plataforma.
- Instalación y configuración de Linux incrustado en un asistente personal digital.

## 1.5. Organización del documento

En el capítulo 2 se abordan las curvas elípticas, comenzando desde la definición de grupo abeliano y campos finitos, siguiendo con la definición general de las curvas elípticas y sus operaciones, desde el punto de vista geométrico y algebraico. Este capítulo finaliza mencionando un tipo de curva elíptica en particular: la curva elíptica propuesta por Koblitz.

La criptografía de curvas elípticas se presenta en el capítulo 3. En el, se encuentra la explicación del por qué son tan seguras las curvas elípticas. También, se puede observar los algoritmos para crear las claves (pública y privada), así como el esquema de firma digital. Finaliza con una breve muestra de curvas elípticas que recomienda una organización de los Estados Unidos.

El capítulo 4 se refiere a la manera como se fue implementando el protocolo de comunicación de acuerdo de clave compartida, mencionando los elementos más importantes que integran dicho protocolo y de qué manera están conformadas para su integración.

Para lograr que la implementación de dicho protocolo de comunicación fuera portable al asistente personal digital, se tuvo que hacer uso de un compilador multi-plataforma, así como preparar el dispositivo con las herramientas necesarias pertinentes para su buen funcionamiento. Todo este desarrollo, así como los resultados obtenidos, se pueden ver en el capítulo 5. En seguida, en el capítulo 6 se mencionan varias conclusiones que se obtuvieron al finalizar este trabajo y las sugerencias o trabajos futuros que restan por hacerse.

Finalmente, se agregaron algunos anexos, como apoyo didáctico. Entre estos se encuentra el apéndice A, donde se examina el modelo criptográfico RSA. Como información sobre las distintas curvas elípticas que recomienda el Instituto Nacional de Tecnología y Estándares (NIST del inglés *"National Institute of Standards and Technology"*), agencia de los Estados Unidos de América, encargada en el desarrollo y establecimiento de estándares, en el apéndice B se mencionan las características de tres de ellas. En los apéndices C, E y F, se analiza la librería NTL, considerando las características más sobresalientes. Por último, se muestra en el apéndice G y H, un resumen de los distintos sistemas operativos que son incrustados y las PDAs que soportan Linux.

Capítulo 2

# Teoría de las curvas elípticas

La forma de comparar la eficiencia y rapidez entre los sistemas criptográficos asimétricos[1], consiste en la dificultad de llevar a cabo dos operaciones sobre un algoritmo; es decir, una operación hacia adelante que debe ser tratable operacionalmente y la operación de la inversa, que debe ser en términos prácticos intratable.

La dificultad para realizar estas dos operaciones, depende de la longitud de las claves que se generen. Entre más grande sea la longitud del par de claves, mayor será la dificultad para ambas operaciones. Pero realizar la operación inversa, conforme crece el tamaño de las claves, su dificultad se incrementa sub-exponecialmente, en cambio la operación hacia adelante, va teniendo un crecimiento lineal.

Con lo anterior se entiende que para obtener mayor seguridad se debe hacer uso de claves mucho más grandes. Aunque la operación hacia adelante, teniendo un par de claves grandes presenta cierta dificultad para realizarla; la operación inversa resulta ser intratable. Pero a pesar de que la operación hacia adelante del algoritmo no se compara con la dificultad que presenta la operación de la inversa, al incrementar el tamaño de las claves, se requieren mayores recursos (poder cómputo, consumo de energía, espacio en memoria, etc.) aunque nunca se va a comparar con el poder de cómputo para realizar la operación inversa.

En el caso de ECC las implementaciones son más pequeñas y la eficiencia se mejora notablemente que otras. Otra característica de ECC, es que se pueden considerar claves con una longitud más pequeña, ofreciendo el mismo nivel de seguridad que otros sistemas

[1] Un *sistema criptográfico asimétrico* es aquel que usa un par de claves. Una clave pública para cifrar y una clave privada para descifrar. Por su parte aquellos sistemas criptográficos que hacen uso de una sola clave se les llama *sistemas criptográficos simétricos*

criptográficos ofrecen pero estos últimos utilizando claves con una longitud mayor.

Para darse una idea de la diferencia que existe entre ECC y su competidor más cercano RSA, en la tabla 1.1 se muestran las equivalencias de tamaños de clave entre cada sistema criptográfico.

Algunos sistemas criptográficos (Diffie-Hellman, RSA, DSAE, NRE, MQV, ECC, etc.) realizan las operaciones sobre grupos, confiando su seguridad en la resolución del logaritmo discreto[2]. Pero lo que lo hace diferente es como se definen los grupos, es decir, la manera como los elementos se les define dentro del grupo, así como la forma en que las operaciones aritméticas fundamentales se llevan a cabo.

Entonces la seguridad que ofrece ECC con tamaños de claves pequeñas se debe a la definición de las operaciones en la curva elíptica. Es decir, en ECC, se usa la curva elíptica para definir los miembros del conjunto sobre el cual el grupo es calculado, también las operaciones entre las cuales definen la manera como se realizarán matemáticamente sobre el grupo.

Para comprender mejor lo anterior; mientras que Diffie-Hellman usa campos finitos con elementos cuya particularidad es que son números enteros primos, ECC integran al conjunto los puntos $(x, y)$ que se encuentran sobre la curva y que además con ese conjunto de elementos (puntos) se puede realizar las operaciones aritméticas fundamentales (suma, resta, multiplicación y división).

Otra ventaja que representa el uso de grupos de puntos sobre una curva elíptica, es que estos se pueden representar sobre campos característicos dos, es decir, los elementos del grupo pueden ser ceros y unos, permitiendo con ello que la implementación sea más compacta y las operaciones hacia adelante se hagan con menor dificultad, pero incrementando de manera notable la dificultad para resolver la operación inversa del algoritmo.

Así, pues, la teoría de curvas elípticas es un tema que por siglos ha sido estudiado. Su estudio se remonta desde el siglo XVIII, convirtiéndose con el paso del tiempo, en una línea de investigación para el mundo de las matemáticas. Pero es hasta finales del siglo pasado cuando se descubre su uso práctico. Esto es, como es de pensarse en la informática y más concretamente en lo que respecta a la seguridad.

Las bases matemáticas necesarias para incursionar en la teoría de curvas elípticas, pueden parecer hasta cierto punto temas triviales, debido a que todo parte del álgebra

[2]El *logaritmo discreto*, se basa en encontrar el logaritmo de un número dentro de un campo finito primo

abstracta, es decir; comenzando con teoría de números, pasando por funciones, polinomios, grupos, etc., hasta campos, siguiendo con un poco de geometría analítica y álgebra lineal. Es necesario estudiar estos temas, tal vez no a profundidad, para comprender el desarrollo matemático de las curvas elípticas.

## 2.1. Introducción

Comenzar el estudio de ECC, sin el conocimiento teórico de curvas elípticas, dificultan el entendimiento del desarrollo de esté modelo criptográfico. Es conveniente primeramente revisar la teoría de curvas elípticas, siendo el objetivo de esté capítulo.

Existe un gran número de autores que se han dedicado al estudio de curvas elípticas [Menezes91, Silverman94, Frey94, Charlap88, Semaev98, Solinas00]. En esté capítulo, se tomaron como base en el estudio de grupos abelianos a los autores [Ash89, Beachy96]. Para el tema de campos finitos se apoyó en las siguientes referencias [Menezes96, García03, Hankerson04] y finalmente, en lo que se refiere a la teoría de curvas elípticas se consultó a las siguientes fuentes de información [Charlap88, Belingueres, García03, Hankerson04, Koblitz04]

Las curvas elípticas se apoyan en los grupos abelianos, y por ello en la primera parte de este capítulo se define su estructura, características y las propiedades que debe de cumplir. A continuación, se da un breve tratado de los campos finitos, ya que estos permiten trabajar con criptografía, siguiendo con la definición de curvas elípticas sobre campos finitos. Finalmente, se analiza un caso específico de curvas elípticas (curvas elípticas de Koblitz).

## 2.2. Grupos Abelianos

Un *grupo abeliano* $(G, *)$ consiste de un conjunto $G$ con la operación binaria $*$: $G \times G \rightarrow G$ satisfaciendo las siguientes propiedades:

- *Cerradura:* Para todo $a, b \in G$, el elemento $a * b$ es un elemento de $G$.
- *Asociatividad:* Para todo $a, b, c \in G$ se tiene

$$a * (b * c) = (a * b) * c.$$

- *Identidad:* Existe un elemento $e \in G$ tal que

$$a * e = e * a = a$$

para toda $a \in G$.

- *Inversa:* Para cada $a \in G$, existe un elemento $a^{-1} \in G$, llamado la inversa de $a$, tal que
$$a * a^{-1} = a^{-1} * a = e.$$
- *Conmutatividad:* Pata todo $a, b \in G$ se tiene
$$a * b = b * a.$$

Son dos las operaciones que se realizan en los grupos: la suma $(+)$ o la multiplicación $(\cdot)$. En la primera operación, el grupo es llamado un grupo *aditivo*. El elemento identidad, de la adición, es usualmente denotado por 0, y la inversa (aditiva) de $a$ es denotado por $-a$. Para la segunda operación, el grupo es llamado un grupo *multiplicativo*. El elemento identidad, de la multiplicación, es usualmente denotado por 1, y la inversa (multiplicativa) de $a$ es denotado por $a^{-1}$.

Ahora bien, un grupo $G$ se dice ser un grupo *finito* si el conjunto $G$ tiene un número finito de elementos. En ese caso, el número finito de elementos es llamado el *orden* de $G$, denotado como $\mid G \mid$.

Algún grupo de orden primo es *cíclico*. Sea $G$ un grupo de orden $n$, y sea $g$ algún elemento de $G$. El conjunto $\langle g \rangle = \{a \in G : \ a = g^i \ para \ 0 \leq i \leq n\}$, es llamado un *subgrupo cíclico* de $G$ generado por $g$. El grupo $G$ es llamado grupo *cíclico* si existe un elemento $g \in G$, tal que $G = \langle g \rangle$. Para este caso $g$ es nombrado un *generador* de $G$.

En el ejemplo 2.1 se observa a los elementos que son generadores del grupo y aquellos que generan subgrupos.

**Ejemplo 2.1** Sea $G$ un grupo de orden $n = 11$, obsérvese que el orden del grupo es un entero primo. Los elementos que integran al grupo son $\{0, 1, 2, \ldots, n-1\}$. Los números generadores de un subgrupo cíclico son: 3, 4, *y* 9, como puede apreciarse en la tabla 2.1. Los números generadores del grupo G son: 2, 6, 7, *y* 8. En la tabla 2.2 se observan las potencias de cada uno de estos números.

## 2.3. Campos Finitos

Un *campo* es un conjunto de elementos que satisfacen las condiciones de: *asociatividad, conmutatividad, identidad, inversa y distributiva*. Además consisten de un conjunto

| $g^i = a$ | | | | | |
|---|---|---|---|---|---|
| $i$ | $g = 3$ | $g = 4$ | $g = 5$ | $g = 9$ | $g = 10$ |
| 1 | 3 | 4 | 5 | 9 | 10 |
| 2 | 9 | 5 | 3 | 4 | 1 |
| 3 | 5 | 9 | 4 | 3 | 10 |
| 4 | 4 | 3 | 9 | 5 | 1 |
| 5 | 1 | 1 | 1 | 1 | 10 |
| 6 | 3 | 4 | 5 | 9 | 1 |
| 7 | 9 | 5 | 3 | 4 | 10 |
| 8 | 5 | 9 | 4 | 3 | 1 |
| 9 | 4 | 3 | 9 | 5 | 10 |
| 10 | 1 | 1 | 1 | 1 | 1 |
| 11 | 3 | 4 | 5 | 9 | 10 |

Cuadro 2.1: Números generadores de un subgrupo cíclico.

$\mathbb{F}$ junto con dos operaciones, la suma (denotada por $+$) y la multiplicación (denotada por $\cdot$), que satisfacen las siguientes propiedades aritméticas:

1. $(\mathbb{F}, +)$ es un grupo abeliano con la identidad aditiva indicada por 0.
2. $(\mathbb{F}^* = \mathbb{F} - 0, \cdot)$ es un grupo abeliano con la identidad multiplicativa indicada por 1.
3. Por la ley distributiva:
$$(a + b) \cdot c = a \cdot c + b \cdot c,$$
para todo $a, b, c \in \mathbb{F}$.

Si el conjunto $\mathbb{F}$ es finito, es decir, si contiene un número finito de elementos, entonces se dice que es un campo *finito* o conocido también como campo de *Galois*.

Un campo finito $\mathbb{F}$ no solo cuenta con las operaciones de la suma y multiplicación, sino también con las inversas de estás. Por lo tanto, la *resta* de los elementos del campo es definida en términos de la suma: para $a, b \in \mathbb{F},\ a - b = a + (-b)$ donde $-b$ es el único elemento en $\mathbb{F}$ tal que $b + (-b) = 0$, por lo tanto a $-b$ se le llama el *negativo* de $b$. Para la división de los elementos del campo, está dada en términos de la multiplicación: para $a, b \in \mathbb{F}$ con $b \neq 0$, se tiene $a/b = a \cdot b^{-1}$, donde $b^{-1}$ es el único elemento en $\mathbb{F}$ tal que $b \cdot b^{-1} = 1$, de este modo, a $b^{-1}$ se le conoce como la *inversa* de $b$.

El *orden* de un campo finito, es el número de elementos que tiene el campo, más concretamente, el orden del campo es un primo o la potencia de un primo ($p^m$).

| $g^i = a$ | | | | |
|---|---|---|---|---|
| $i$ | $g = 2$ | $g = 6$ | $g = 7$ | $g = 8$ |
| 1 | 2 | 6 | 7 | 8 |
| 2 | 4 | 3 | 5 | 9 |
| 3 | 8 | 7 | 2 | 6 |
| 4 | 5 | 9 | 3 | 4 |
| 5 | 10 | 10 | 10 | 10 |
| 6 | 9 | 5 | 4 | 3 |
| 7 | 7 | 8 | 6 | 2 |
| 8 | 3 | 4 | 9 | 5 |
| 9 | 6 | 2 | 8 | 7 |
| 10 | 1 | 1 | 1 | 1 |
| 11 | 2 | 6 | 7 | 9 |

Cuadro 2.2: Números que generan un grupo cíclico.

Así pues, existe un campo finito $\mathbb{F}$ de orden $q$ si y solo si $q$ es una potencia de un primo; $q = p^m$ donde $p$ es un número primo llamado la *característica* de $\mathbb{F}$, y $m$ es un entero positivo. Si $m = 1$, entonces $\mathbb{F}$ es llamado un *campo primo*. Si $m \geq 2$, entonces $\mathbb{F}$ es llamado *campo de extensión*.

Si $q$ es la potencia primo, existe exactamente un campo de orden $q$. Se dice entonces, que alguno de los dos campos finitos de orden $q$ son *isomorfos* y se puede representar por $\mathbb{F}_p$ o también como $GF(p)$. Es decir, alguno de los dos campos de orden $q$ son estructuralmente iguales, excepto que la forma de etiquetar los elementos del campo puede diferir.

Los elementos que no son iguales a cero de un campo $\mathbb{F}_q$, denotado como $\mathbb{F}_q^*$, forman un grupo *cíclico* bajo la operación de la multiplicación. Por lo que existe un elemento $b \in \mathbb{F}_q^*$ llamado el *generador* tal que:

$$\mathbb{F}_q^* = \{b^i : 0 \leq i \leq q-2\}$$

### 2.3.1. Campo primo $\mathbb{F}_p$

Sea $p$ un número primo. El campo $\mathbb{F}_p$, es llamado *campo primo*, que está compuesto por el conjunto de enteros $\{0, 1, 2, \ldots, p-1\}$ con las siguientes operaciones:

- **Suma módulo** $p$: Sea $a, b \in \mathbb{F}_p$, entonces $a+b = r$, donde $r$ es el residuo de la división de $a + b$ entre $p$, $0 \leq r \leq p-1$.

- **Multiplicación módulo** $p$: Si $a, b \in \mathbb{F}_p$, entonces $a \cdot b = s$, donde $s$ es el residuo de la división de $a \cdot b$ entre $p$ y donde $0 \leq s \leq p-1$.
- **Inversa**: Sea $a$ elemento de $\mathbb{F}_p$, considerando que $a \neq 0$, la *inversa* de $a$ módulo $p$, denotado $a^{-1}$, es el único entero $e \in \mathbb{F}_p$ tal que $a \cdot e = 1$ ($a \cdot e$ es llevada a cabo módulo $p$).

**Ejemplo 2.2** Sea $\mathbb{F}_5$ un campo finito con los siguientes elementos $\{0, 1, 2, 3, 4\}$. Las operaciones aritméticas sobre $\mathbb{F}_5$ son:

- **Suma**: $5 + 3 = 3$ ya que $(5 + 3)$ módulo $5 = 3$.
- **Multiplicación**: $4 \cdot 3 = 2$, ya que $(4 \cdot 3)$ módulo $5 = 2$.
- **Inversa**: $4^{-1} = 4$, ya que $(4 \cdot 4)$ módulo $5 = 1$.

Los elementos 2 y 3 son generadores del campo $\mathbb{F}_5$. Las potencias de 2 son:

$$2^0 = 1$$
$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 3$$
$$2^4 = 1$$

Abajo se puede observar las potencias del elemento generador 3:

$$3^0 = 1$$
$$3^1 = 3$$
$$3^2 = 4$$
$$3^3 = 2$$
$$3^4 = 1$$

### 2.3.2. Campos binarios $\mathbb{F}_{2^m}$

Los campos finitos de orden $2^m$ son llamados *campos binarios* o *campos finitos característicos dos*, que se representan como $\mathbb{F}_{2^m}$. Los campos $\mathbb{F}_{2^m}$ pueden ser vistos como espacios vectoriales de dimensión $m$ sobre $\mathbb{F}_2$, es decir, existen $m$ elementos $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ en $\mathbb{F}_{2^m}$ tales que cada elemento $\alpha \in \mathbb{F}_{2^m}$ puede ser escrito de manera única como:

$$\alpha = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{m-1}\alpha_{m-1},$$

donde los coeficientes $a_i$ puede ser algún elemento del conjunto $\{0, 1\}$. Al conjunto $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ se le denomina una base $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$. Dada una base tal, un elemento $\alpha$ del campo puede ser representado por la cadena de bits $(a_0, a_1, \ldots, a_{m-1})$.

Existen diferentes bases de $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$. Es decir, las implementaciones para hardware y software son distintas, y en cada uno de los casos se debe buscar la más óptima, esto depende de la base que se tome. Para los campos finitos se pueden mencionar dos tipos de bases que son popularmente conocidas: *bases polinomiales* y *bases normales*.

**Bases polinomiales**

Una manera de construir $\mathbb{F}_{2^m}$ es usar la representación base polinomial. Los elementos de $\mathbb{F}_{2^m}$ son polinomios binarios cuyos coeficiente están en el campo $\mathbb{F}_2 = \{0, 1\}$ de grado $m - 1$:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + ax + a_0 : a_i \in \{0, 1\}\}.$$

Ahora bien, sea

$$f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_2x^2 + f_1x + f_0$$

donde $f_i \in \{0, 1\}$ para $i = 0, 1, \cdots, m - 1$, un polinomio irreducible de grado $m$ sobre $\mathbb{F}_2$. Un *polinomio irreducible* $f(x)$ se refiere, que $f(x)$ no puede ser factorizado por un producto de polinomios binarios de grado menor que $m$. Entonces $f(x)$ define una *base polinomial* de $\mathbb{F}_{p^m}$.

Al elemento $a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + ax + a_0$ usualmente se le denota por la cadena de bits $(a_{m-1}a_{m-2} \cdots a_1a_0)$ de longitud $m$, de modo que:

$$\mathbb{F}_{2^m} = \{(a_{m-1}a_{m-2} \cdots a_1a_0) : a_i \in \{0, 1\}\}.$$

**Ejemplo 2.3** Sea el campo binario $\mathbb{F}_{2^4}$, los elementos que componen dicho campo son las 16 $(2^4)$ combinaciones polinomiales de grado 3, el cual es mostrado abajo, junto con su representación como cadena de bits:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | (0000) | $x^2$ | (0100) | $x^3$ | (1000) | $x^3 + x^2$ | (1100) |
| 1 | (0001) | $x^2 + 1$ | (0101) | $x^3 + 1$ | (1001) | $x^3 + x^2 + 1$ | (1101) |
| $x$ | (0010) | $x^2 + x$ | (0110) | $x^3 + x$ | (1010) | $x^3 + x^2 + x + 1$ | (1110) |
| $x + 1$ | (0011) | $x^2 + x + 1$ | (0111) | $x^3 + x + 1$ | (1011) | $x^3 + x^2 + x + 1$ | (1111) |

Entonces se definen las siguientes operaciones aritméticas sobre los elementos de $\mathbb{F}_{2^m}$, cuando tiene una representación de base polinomial con reducción polinomial $f(x)$:

- **Suma**: Si $(a_{m-1}a_{m-2}\cdots a_1a_0)$ y $(b_{m-1}b_{m-2}\cdots b_1b_0)$ son elementos de $\mathbb{F}_{2^m}$, entonces $a+b=c$ y $c=(c_{m-1}c_{m-2}\cdots c_1c_0)$, donde $c_i=a_i+b_i$ mód 2.
- **Multiplicación**: Si $(a_{m-1}a_{m-2}\cdots a_1a_0)$ y $(b_{m-1}b_{m-2}\cdots b_1b_0)$ son elementos de $\mathbb{F}_{2^m}$, entonces $a\cdot b=r$ y $r=(r_{m-1}r_{m-2}\cdots r_1r_0)$, donde el polinomio $r_{m-1}x^{m-1}+r_{m-2}x^{m-2}+\cdots+r_1x+r_0$ es el residuo de la división de $(a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_1x+a_0)\cdot(b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\cdots+b_1x+b_0)$ entre $f(x)$.
- **Inversa**: Si $a$ es un elemento de $\mathbb{F}_{2^m}$ diferente de cero, el inverso de $a$, denotado por $a^{-1}$, es el elemento $c\in\mathbb{F}_{2^m}$ tal que $c\cdot a=1$.

Nada más cabe destacar, que en la adición de los elementos en el campo, se lleva a cabo mediante el operador XOR, bit a bit de sus representaciones vectoriales.

En seguida se muestra un ejemplo con las operaciones aritméticas definidas anteriormente.

**Ejemplo 2.4** Considerando el campo binario propuesto en el ejemplo 2.3 junto con los elementos que integran dicho campo. Además, sea el siguiente polinomio irreducible:

$$f(x)=x^4+x+1.$$

Su representación del polinomio $f(x)$ en binario: $\{10011\}$. Ahora, en número hexadecimal: 13.

Sea el polinomio $a=x^3+x^2+1$ y sea el polinomio $b=x^2+x+1$. Véase qué tanto el polinomio $a$ como el polinomio $b$ pertenecen al campo finito $\mathbb{F}_{2^4}$.

- **Suma**: Recuérdese que la operación de la suma de polinomios se hace bit a bit con la operación módulo 2 (XOR), es decir, si se representan los polinomios $a$ y $b$ de forma binaria:

$$\begin{aligned} a &= x^3+x^2+1 \\ &= a_3a_2a_1a_0 \\ &= 1101 \\ b &= x^2+x+1 \\ &= b_3b_2b_1b_0 \\ &= 0111 \end{aligned}$$

se tiene entonces que:

$$\begin{aligned} c &= (a+b) \text{ mód } 2 \\ &= c_3c_2c_1c_0 \end{aligned}$$

donde:

$$\begin{aligned} c_3 &= (a_3+b_3) \text{ mód } 2 \\ &= (1+0) \text{ mód } 2 \\ &= 1 \\ c_2 &= (a_2+b_2) \text{ mód } 2 \\ &= (1+1) \text{ mód } 2 \\ &= 0 \\ c_1 &= (a_1+b_1) \text{ mód } 2 \\ &= (0+1) \text{ mód } 2 \\ &= 1 \\ c_0 &= (a_0+b_0) \text{ mód } 2 \\ &= (1+1) \text{ mód } 2 \\ &= 0 \end{aligned}$$

por lo tanto,

$$c = 1010$$

Así pues, la suma de los polinomios, $a$ y $b$, quedaría como sigue:

$$(x^3+x^2+1)+(x^2+x+1) = 1010 = x^3+x.$$

- **Resta**: Siguiendo con el procedimiento de la suma, pero ahora tomando en cuenta la resta, se tiene que
$$(x^3+x^2+1)-(x^2+x+1) = x^3+x.$$

- **Multiplicación**: La multiplicación de los polinomios se realiza de la manera tradicional, el único detalle es que al simplificar términos semejantes se lleva a cabo módulo 2. Multiplicando los polinomios $a$ y $b$

$$(x^3+x^2+1)\cdot(x^2+x+1) = x^5+x^4+x^3+x^4+x^3+x^2+x^2+x+1$$

los términos semejantes son:

$$x^5 + \underline{x^4} + \underline{\underline{x^3}} + \underline{x^4} + \underline{\underline{x^3}} + \underline{\underline{x^2}} + \underline{\underline{x^2}} + x + 1$$

$x^4$, $x^3$, y $x^2$. Por lo tanto estos valores pasan a nulificarse, por ejemplo, el término $x^4$ se simplifica haciendo lo siguiente: $x^4 + x^4 = (1+1)$ módulo $2 = 0$. Así para los demás términos semejantes.

Simplificando cada uno de los términos semejantes, se tiene

$$(x^3 + x^2 + 1) \cdot (x^2 + x + 1) = x^5 + x + 1$$

haciendo módulo el polinomio irreducible $f(x) = x^4 + x + 1$

$$\begin{aligned}(x^3 + x^2 + 1) \cdot (x^2 + x + 1) &= (x^5 + x + 1) \text{ mód } (x^4 + x + 1) \\ &= x^2 + 1.\end{aligned}$$

Por lo tanto, el resultado de multiplicar el polinomio, $a$ y $b$, sería: $x^2 + 1$.

- **Inversa**: $(x^3 + x^2 + 1)^{-1} = x^2$ ya que

$$(x^3 + x^2 + 1) \cdot x^2 \text{ mód } (x^4 + x + 1) = 1$$

El elemento generador del campo finito cíclico $\mathbb{F}_{2^4}^*$ es el polinomio $x$, que al representarlo como vector quedaría $\alpha = (0010)$. Entonces las 15 potencias del elemento generador $\alpha$ son:

$$\begin{array}{llll} \alpha^0 = (0001) & \alpha^1 = (0010) & \alpha^2 = (0100) & \alpha^3 = (1000) \\ \alpha^4 = (0011) & \alpha^5 = (0110) & \alpha^6 = (1100) & \alpha^7 = (1011) \\ \alpha^8 = (0101) & \alpha^9 = (1010) & \alpha^{10} = (0111) & \alpha^{11} = (1110) \\ \alpha^{12} = (1111) & \alpha^{13} = (1101) & \alpha^{14} = (1001) & \alpha^{15} = (0001) \end{array}$$

**Bases normales**

Otro tipo de bases son las normales. Una *base normal* de $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$ es una base de la forma:

$$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\},$$

donde $\beta \in \mathbb{F}_{2^m}$. Cualquier elemento $a \in \mathbb{F}_{2^m}$ puede ser escrito como:

$$a = \sum_{i=0}^{m-1} a_i \beta^{2^i} = a_0\beta + a_1\beta^2 + \cdots + a_{m-1}\beta^{2^{m-1}},$$

donde $a_i \in \mathbb{F}_2$; el vector asociado con $a$ es $A = (a_0 a_1 a_2 \cdots a_{m-1})$. Por tanto, los elementos de $\mathbb{F}_{2^m}$ puede ser identificado como una cadena binaria $2^m$ de longitud $m$. La representación en bases normales presentan la ventaja de que al elevar un elemento al cuadrado puede hacerse de manera muy eficiente:

$$\begin{aligned} a &= \left(\sum_{i=0}^{m-1} a_i \beta^2\right)^2 \\ &= \left(a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + \cdots + a_{m-1}\beta^{2^{m-1}}\right)^2 = (a_0, a_1, a_2, \cdots a_{m-1})^2 \\ &= \left(a_0\beta^2 + a_1\beta^{2^2} + a_2\beta^{2^3} + \cdots + a_{m-1}\beta\right) = (a_{m-1}, a_1, a_2, \cdots a_{m-2}) \end{aligned}$$

es decir, elevar al cuadrado se reduce a un corrimiento de bits.

La multiplicación de los elementos, por su parte, puede ser muy complicada. Considérese $\{\beta^{2^i}\}$, $0 \leq i \leq m - i$, una base normal para $\mathbb{F}_{2^m}$ y sean $A, B \in \mathbb{F}_{2^m}$, tenemos que

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$$

y

$$B = \sum_{i=0}^{m-1} b_i \beta^{2^i}$$

Representando $A$ y $B$ en sus bases normales, se tiene $A = (a_0 a_1 a_2 \cdots a_{m-1})$ y $B = (b_0 b_1 b_2 \cdots b_{m-1})$. Entonces para el producto, hace

$$A \cdot B = C = \sum_{i=0}^{m-1} c_i \beta^{2^j}, \quad (C = c_0 c_1 c_2 \cdots c_{m-1}).$$

Ahora cada $c_k \in \mathbb{F}_{2^m}$ tienen la forma

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \beta^{2^i} \beta^{2^j}.$$

Sea $\beta^{2^i}\beta^{2^j} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j d_{ij}^{(k)}$, quedando finalmente

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j d_{ij}^{(k)}.$$

Entonces, se define la matriz $T_i = \left(d_{ij}^{(k)}\right)$, $0 \leq j,k \leq m-1$, y al conjunto $\{T_0, T_1, T_2, \cdots, T_{m-1}\}$ como la tabla resultante.

Así pues, la multiplicación de $A$ por $B$ significa calcular todos los $c_k$. La rapidez del calculo, depende de que tantos coeficientes $d_{ij}^{(k)} \in \mathbb{F}_2$ sean igual a 1. Por otro lado, para que los $d_{ij}^{(k)}$ sean ceros o unos, depende de la base que se tome.

El número de bases es finito y por lo tanto se puede buscar una base que contenga el menor número de $d_{ij}^{(k)}$ iguales a uno. Para esto, se puede trabajar solo con los elementos de $T_0$, ya que se puede mostrar que todas las $T_i$ tienen el mismo número de unos, no siendo estos, más que $2m-1$. Al número de unos que tiene $T_0$ se le llamará *complejidad* de $T$ que se representa por $C_T$, entonces $C_T \geq 2m-1$. La base normal se dice ser *óptima* si $C_T = 2m-1$.

La representación de una base normal óptima (BNO), está clasificada en tipo I y tipo II. El valor de $m$ determina el tipo que se está utilizando. Es decir:

1. $m+1$ es primo y 2 es un primitivo en $\mathbb{F}_{2^{m-1}}$, en tal caso las $n$ raíces $(n+1)$-ésimas no triviales de la unidad forman una BNO para $\mathbb{F}_{2^m}$ denominada una *BNO de tipo I*.

2. $2m+1$ es primo y se cumple que 2 es primitivo en $\mathbb{F}_{2^{m+1}}$ ó que $2m+1 \equiv 3 \pmod 4$ y el orden multiplicativo de 2 en $\mathbb{F}_{2^{m+1}}$ es $m$, entonces $\alpha = \gamma + \gamma^{-1}$ genera una BNO de $\mathbb{F}_{2^m}$, donde $\gamma$ es una $(2m+1)$-ésima raíz de la unidad. A está se le va a denominar una *BNO tipo II*.

Para dejar más claro la multiplicación en BNO, se muestra el siguiente procedimiento:

1. Si $\mathbb{F}_{2^m}$ tiene una BNO tipo I, entonces $f(x) = x^m + x^{m-1} + \ldots + x^2 + x + 1$. De otra manera, si $\mathbb{F}_{2^m}$ es una BNO tipo II se calcula $f(x) = f_m(x)$ usando la siguiente fórmula recursiva:

$$\begin{aligned} f_0(x) &= 1, \\ f_1(x) &= x+1, \\ f_{i+1}(x) &= xf_i(x) + f_{i-1}(x), i = 1,2,3,\cdots,m. \end{aligned}$$

   Tenemos que $f(x)$ es un polinomio de grado $m$ con coeficientes en $\mathbb{F}_2$. El conjunto de polinomios $x, x^2, x^{2^2}, \ldots, x^{2^{m-1}}$ módulo $f(x)$ forman una base de $\mathbb{F}_{2^m}$ sobre $\mathbb{F}_2$ denominado *base normal*.

2. Se construye la matriz $A$ de tamaño $m \times m$ cuya i-ésima fila, $0 \leq i \leq m-1$, es el vector binario correspondiente al polinomio $x^{2^i}$ mód $f(x)$. Las filas y columnas de $A$ son índices de enteros de 0 hasta $m-1$. Los elementos de $A$ son aquellos que se encuentran en $\mathbb{F}_2$.

3. Se determina la inversa de la matriz $A^{-1}$ de $A$ sobre $\mathbb{F}_2$.

4. Se construye la matriz $T'$ de tamaño $m \times m$ cuya i-ésima fila, $0 \leq i \leq m-1$, se obtiene calculando el polinomio $x \cdot x^{2^i}$ mód $f(x)$. Entonces se resuelve la matriz $T = T'A^{-1}$ sobre $\mathbb{F}_2$.

5. Se determinan los coeficientes $\lambda_{ij}$, $i = 1, 2, \cdots, m-1$, tal que $\lambda_{ij} = T(j-i, -i)$, donde los índices de $T$ toman el módulo $m$.

Cada producto del término $\lambda_{ij}$ es un elemento de $\mathbb{F}_2$. Se da el caso de que $\lambda_{0j} = 1$ para precisamente una sola $j$, $0 \leq j \leq m-1$. También ocurre que para cada $i$, $0 \leq i \leq m-1$, $\lambda_{ij} = 1$ para precisamente dos diferentes $j$, $0 \leq j \leq m-1$. De aquí que exactamente $2m-1$ de los $m^2$ coeficientes de la matriz $T$ son $1's$ y el resto son $0's$. Esta escasez de unos es por lo que la base normal es llamada una *base normal óptima*.

**Ejemplo 2.5** Representación de la base polinomial óptima para el campo $\mathbb{F}_{2^4}$. Siguiendo con el procedimiento que se vio anteriormente se hace:

1. El campo $\mathbb{F}_{2^4}$ tiene un tipo I de BNO; por ello $f(x) = x^4 + x^3 + x^2 + x + 1$. Así el conjunto de polinomios $x, x^2, x^4, x^8$ forman la base normal de $\mathbb{F}_{2^4}$ sobre $\mathbb{F}_2$.

2. Los renglones de la matriz $A$ se construyen como sigue:

   - **Renglón 0**: $x$ mód $f(x) = x = (0010)$.
   - **Renglón 1**: $x^2$ mód $f(x) = x^2 = (0100)$.
   - **Renglón 2**: $x^4$ mód $f(x) = x^3 + x^2 + x + 1 = (1111)$.
   - **Renglón 3**: $x^8$ mód $f(x) = x^3 = (1000)$.

   Con lo anterior tenemos que

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

3. La inversa de $A$, $A^{-1}$, sobre $F_2$ es

$$A^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

4. Los renglones de $T'$ se construyen como sigue:

   - **Renglón 0**: $v = x \cdot x \bmod f(x) = x^2 = (0100)$.
   - **Renglón 1**: $v = x \cdot x^2 \bmod f(x) = x^3 = (1000)$.
   - **Renglón 2**: $v = x \cdot x^4 \bmod f(x) = x^5 \bmod f(x) = 1 = (0001)$.
   - **Renglón 3**: $v = x \cdot x^8 \bmod f(x) = x^9 \bmod f(x) = x^3 + x^2 + x + 1 = (1111)$.

   Por lo tanto

$$T' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix},$$

   ahora tenemos que

$$T = T' \cdot A^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

5. Se determinan los coeficientes $\lambda_{ij}$:

$$\begin{array}{llll} \lambda_{0,0} = T(0,0) = 0 & \lambda_{1,0} = T(3,3) = 0 & \lambda_{2,0} = T(2,2) = 1 & \lambda_{3,0} = T(1,1) = 0 \\ \lambda_{0,1} = T(1,0) = 0 & \lambda_{1,1} = T(0,3) = 0 & \lambda_{2,1} = T(3,2) = 1 & \lambda_{3,1} = T(2,1) = 1 \\ \lambda_{0,2} = T(2,0) = 1 & \lambda_{1,2} = T(1,3) = 1 & \lambda_{2,2} = T(0,2) = 0 & \lambda_{3,2} = T(3,1) = 0 \\ \lambda_{0,3} = T(3,0) = 0 & \lambda_{1,3} = T(2,3) = 1 & \lambda_{2,3} = T(1,2) = 0 & \lambda_{3,3} = T(0,1) = 1 \end{array}$$

   así los coeficientes $\lambda_{ij}$ con valor de 1 son: $\lambda_{0,2}, \lambda_{1,3}, \lambda_{2,0}, \lambda_{2,1}, \lambda_{3,1}$ y $\lambda_{3,3}$.

La multiplicación quedaría de la siguiente forma: como $(a_0a_1a_2a_3) \cdot (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$, entonces para $c_i$, $i = 0, 1, 2, 3$, se tiene

$$\begin{aligned} c_0 &= a_0b_2 + a_1(b_1 + b_3) + a_2(b_0 + b_1) + a_3(b_1 + b_3) \\ c_1 &= a_1b_3 + a_2(b_3 + b_0) + a_3(b_1 + b_2) + a_0(b_2 + b_0) \\ c_2 &= a_2b_0 + a_3(b_0 + b_1) + a_0(b_2 + b_3) + a_1(b_3 + b_1) \\ c_3 &= a_3b_1 + a_0(b_1 + b_2) + a_1(b_3 + b_0) + a_2(b_0 + b_2) \end{aligned}$$

Como se dijo en la introducción de este capítulo, una buena manera para comenzar el estudio de las curvas elípticas es conociendo acerca de grupos y más específicamente sobre grupos abelianos (ver sección 2.2), continuando así sobre los campos finitos y más concretamente sobre los campos finitos con característica 2 (ver sección 2.3). Ahora bien, teniendo ya estas bases, se puede comenzar a estudiar lo referente a las curvas elípticas.

## 2.4. Curvas elípticas

Una *curva elíptica* $E$ sobre el campo $\mathbb{F}$ es definida por la ecuación

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.1}$$

donde $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ y $\triangle \leq 0$. A la ecuación 2.1 se le conoce como *Forma Normal de Weierstrass* (FNW). Si $\triangle \leq 0$, la curva elíptica es *no singular*, eso significa que no hay puntos en los cuales la curva tenga dos o más líneas tangentes distintas. Donde $\triangle$ es el *discriminante* de $E$ y esta definido como sigue:

$$\begin{aligned} \triangle &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^1 + 4a_6 \\ d_8 &= a_1^2 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{aligned}$$

Se dice que $E$ esta definida sobre $\mathbb{F}$, debido a que los elementos $a_1, a_2, a_3, a_4, a_6$ definidos en la ecuación 2.1 son elementos de $\mathbb{F}$. Se puede escribir $E/\mathbb{F}$ para hacer énfasis que $E$ está definida sobre $\mathbb{F}$, y por lo tanto, $\mathbb{F}$ es llamado el *campo subyacente*.

Si $L$ es algún campo de extensión de $\mathbb{F}$, el conjunto de puntos racionales sobre $E$ son:

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\}, \qquad (2.2)$$

donde $\mathcal{O}$ es llamado el *punto al infinito* de la curva elíptica. El punto al infinito ($\mathcal{O}$) es considerado un *punto racional* $L$ para todos los campos de extensión $L$ de $F$. Es decir, $\mathcal{O}$, es considerado el único punto en la linea al infinito que satisface la forma proyectiva de la ecuación 2.1.

A partir de la FNW es posible encontrar formas particulares para la ecuación de la curva, dependiendo de la característica del campo subyacente [Hankerson04]. Como a continuación se describe.

Supóngase que se tienen dos curvas elípticas llamadas $E_1$ y $E_2$ definidas sobre $\mathbb{F}$ y dadas por la Forma Normal de Weierstrass, se tiene que:

$$\begin{aligned} E_1 &: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \\ E_2 &: y^2 + \overline{a_1}xy + \overline{a_3}y = x^3 + \overline{a_2}x^2 + \overline{a_4}x + \overline{a_6}, \end{aligned}$$

se dicen ser *isomorfas* sobre $\mathbb{F}$ si existen $u, r, s, t \in \mathbb{F}$, $u$ sea diferente de 0, tal que el cambio de variables

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t),$$

transforma la ecuación $E_1$ a la ecuación $E_2$. La transformación anterior es llamada un *cambio admisible de variables*.

La ecuación 2.1 puede ser simplificada, aplicando el cambio admisible de variables. Se puede considerar los casos de manera separada, donde el campo subyacente $F$ tenga características $car(\mathbb{F}) \neq \{2, 3\}$, ó igual $car(\mathbb{F}) = \{2, 3\}$.

1. Si la característica de $\mathbb{F}$ no es igual a 2 ó 3, entonces el cambio admisible de variables

$$(x, y) \rightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24}\right),$$

transforma a la curva $E$

$$y^2 = x^3 + ax + b, \qquad (2.3)$$

donde $a, b \in \mathbb{F}$. La discriminante de esa curva es $\triangle = -16(4a^3 + 27b^2)$.

2. Si la característica de $\mathbb{F}$ es 2, entonces hay dos casos para considerar:

   *a*) Si $a_1 \neq 0$, entonces el cambio admisible de variables

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

   transformando la curva $E$

   $$y^2 + xy = x^3 + ax^2 + b \tag{2.4}$$

   donde $a, b \in \mathbb{F}$. Tiene como discriminante $\triangle = b$.

   *b*) Si $a_1 = 0$, entonces el cambio admisible de variables

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

   transforma la curva $E$

   $$y^2 + cy = x^3 + ax + b \tag{2.5}$$

   donde $a, b, c \in \mathbb{F}$ y tiene como discriminante $\triangle = c^4$.

3. Si la característica de $\mathbb{F}$ es 3, entonces hay dos casos para considerar:

   *a*) Si $a_1^2 \neq -a_2$, entonces el cambio admisible de variables

   $$(x, y) \to \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3\right)$$

   donde $d_2 = a_1^2 + a_2$ y $d_4 = a_4 - a_1 a_3$ transformando la curva $E$

   $$y^2 = x^3 + ax^2 + b \tag{2.6}$$

   donde $a, b \in \mathbb{F}$ y tiene como discriminante $\triangle = -a^3 b$.

   *b*) Si $a_1^2 = -a_2$, entonces el cambio admisible de variables

   $$(x, y) \to (x, y + a_1 x + a_3)$$

   transforma la curva $E$

   $$y^2 + cy = x^3 + ax + b \tag{2.7}$$

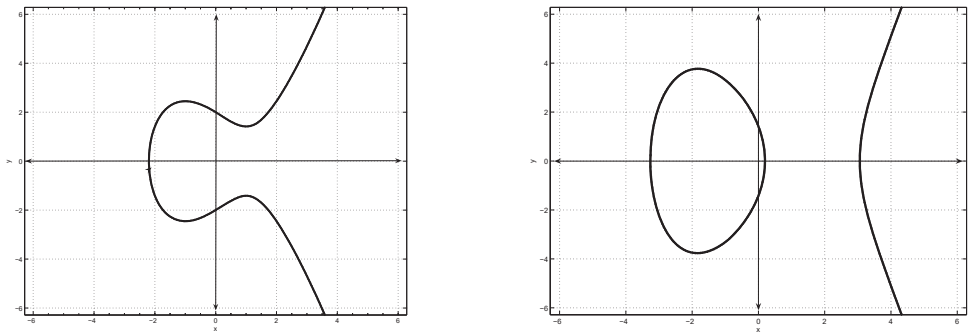
   donde $a, b, c \in \mathbb{F}$ y tiene como discriminante $\triangle = -a^3$.

Haciendo uso de la ecuación 2.3, véase el ejemplo 2.6.

**Ejemplo 2.6** Curvas elípticas sobre $R$. Considerando las siguientes curvas elípticas:

$$E_1 \ : \ y^2 = x^3 - 3x + 4$$
$$E_2 \ : \ y^2 = x^3 - 10x + 2,$$

definidos sobre el campo de los números reales ($\mathbb{R}$). Los puntos $E_1(\mathbb{R}) - \{\mathcal{O}\}$ y $E_2(\mathbb{R}) - \{\mathcal{O}\}$ se pueden apreciar gráficamente en la figura 2.1.



(a) $E_1 : y^2 = x^3 - 3x + 4$ (b) $E_2 : y^2 = x^3 - 10x + 2$

Figura 2.1: Curvas elípticas sobre $\mathbb{R}$.

### 2.4.1. Estructura de un grupo

Sea $E$ una ecuación FNW definida sobre el campo $\mathbb{F}$. Entonces $E \subset \mathbb{F}^2$ es el conjunto de puntos $P = (x, y)$ que satisfacen la ecuación junto con el punto al infinito ($\mathcal{O} = [0 : 1 : 0]$). Sea $L \subset \mathbb{F}^2$ una línea recta. Dado que la ecuación tiene grado 3, $L$ intersecta a $E$ en exactamente tres puntos, por ejemplo $P, Q$ y $R$.

**Ley de composición:** Sean $P, Q \in E$, $L$ la línea que pasa por $P$ y $Q$ (en caso en que $P = Q$, se forma una línea tangente sobre $E$) y $R$ un tercer punto de intersección de $L$ con $E$. Sea $L'$ la línea que conecta $R$ y $\mathcal{O}$. Entonces $P \oplus Q$ es el punto tal que $L'$ intersecta a $E$ en $R, \mathcal{O}$ y $P \oplus Q$.

**Teorema 2.1 (Composición)** *La ley de composición tiene las siguientes propiedades:*

1. *Si una línea $L$ intersecta a $E$ en los puntos (no necesariamente distintos) $P, Q$ y $R$, entonces*

$$(P \oplus Q) \oplus R = \mathcal{O}.$$

2. *$(P \oplus \mathcal{O}) = P$ para toda $P$ que pertenezca a $E$.*

3. *$(P \oplus Q) = Q \oplus P$ para toda $P, Q \in E$.*

4. *Sea $P \in E$, existe un punto de $E$, denotado $\ominus P$, tal que*

$$P \oplus (\ominus P) = \mathcal{O}.$$

5. *Sean $P, Q, R \in E$, entonces*

$$(P \oplus Q) \oplus R = P \oplus (Q \oplus R),$$

*así pues, la ley de composición hace de $E$ un grupo abeliano con el elemento de identidad $\mathcal{O}$.*

6. *Supóngase que $E$ esta definido sobre $\mathbb{F}$, entonces*

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F}^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\},$$

*es un subgrupo de $E$.*

Para ver más a detalle el teorema anterior así como sus demostraciones consultar [Silverman94]. Dado que las propiedades de operación $\oplus$ son similares a las de la suma, por simpleza se denotará como $+$ (el símbolo tradicional) y de igual forma, el inverso aditivo de $P$, $\ominus P$, se denotará como $-P$.

### 2.4.2. Curvas elípticas sobre campos $\mathbb{F}_p$

Sea $\mathbb{F}_p$ un campo finito de característico $p$ diferente de 2 y 3, y sean $a, b \in \mathbb{F}_p$ tal que satisfacen la siguiente condición de equivalencia $4a^3 + 27b^2 \equiv 0 \pmod{p}$, entonces una curva elíptica, denotada $E_{(a,b)}$, sobre $\mathbb{F}_p$ se define por:

$$E_{(a,b)} = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\} \tag{2.8}$$

Los puntos de la curva forman un grupo abeliano bajo la operación aditiva. Véase el ejemplo 2.7.

**Ejemplo 2.7** *Curva elíptica sobre el campo* $\mathbb{F}_{23}$. Los elementos que integran dicho campo son $\{0, 1, 2, \ldots, 22\}$. Sea $p = 23$ y considerando la curva elíptica $E : y^2 = x^3 + x + 4$ sobre $\mathbb{F}_{23}$. Tenemos que $a = 1$ y $b = 4$. Se observa que

$$4a^3 + 27b^2 = 4 + 432 = 436 \equiv 22 \pmod{23},$$

entonces $E$ es realmente una curva elíptica. El conjunto de puntos que satisfacen a $E(\mathbb{F}_{23})$ junto con el punto infinito ($\mathcal{O}$) son:

| | | | |
|---|---|---|---|
| $\mathcal{O}$ | $(0, 2)$ | $(4, 16)$ | $(9, 11)$ |
| $(11, 14)$ | $(15, 16)$ | $(18, 14)$ | $(0, 21)$ |
| $(7, 3)$ | $(9, 12)$ | $(13, 12)$ | $(15, 17)$ |
| $(1, 11)$ | $(7, 20)$ | $(10, 5)$ | $(13, 11)$ |
| $(17, 9)$ | $(22, 19)$ | $(1, 12)$ | $(8, 8)$ |
| $(10, 18)$ | $(14, 5)$ | $(17, 14)$ | $(4, 7)$ |
| $(8, 15)$ | $(11, 9)$ | $(14, 18)$ | $(18, 9)$ |

Si se observa, los pares de números que se obtienen, tanto $x$ como $y$ pertenecen a el campo finito $\mathbb{F}_{23}$.

Existe una regla llamada **regla de la tangente**, que sirve para sumar dos puntos sobre una curva elíptica en $\mathbb{F}_p$ ($E(\mathbb{F}_p)$), obteniendo un tercer punto sobre la misma curva elíptica.

Muchos autores coinciden de que la mejor forma de comprender la operación de la suma y duplicación de dos puntos, es desde el punto de vista geométrico.

Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ dos puntos distintos sobre la curva elíptica $E$. Entonces el punto resultante de la suma de $P$ y $Q$ denotada por $R = (x_3, y_3)$, es definida como sigue:

Primero se traza una línea recta a través de $P$ y $Q$; esta línea intersectará a la curva elíptica en un punto más, que se va llamar $-R$. El punto $-R$ es la *reflexión* en el eje de las x's al punto $R$. Eso se puede apreciar más claramente en la figura 2.2.

Sea $P(x_p, y_p)$, si se desea sumar los puntos $P$ y $-P$, se debe trazar una línea recta que cruce a los dos puntos, la cual, no intersectará a la curva elíptica en un tercer punto. Es por está razón que el grupo de la curva elíptica incluye el punto al infinito ($\mathcal{O}$). Por definición $P + (-P) = \mathcal{O}$, véase la figura 2.3.

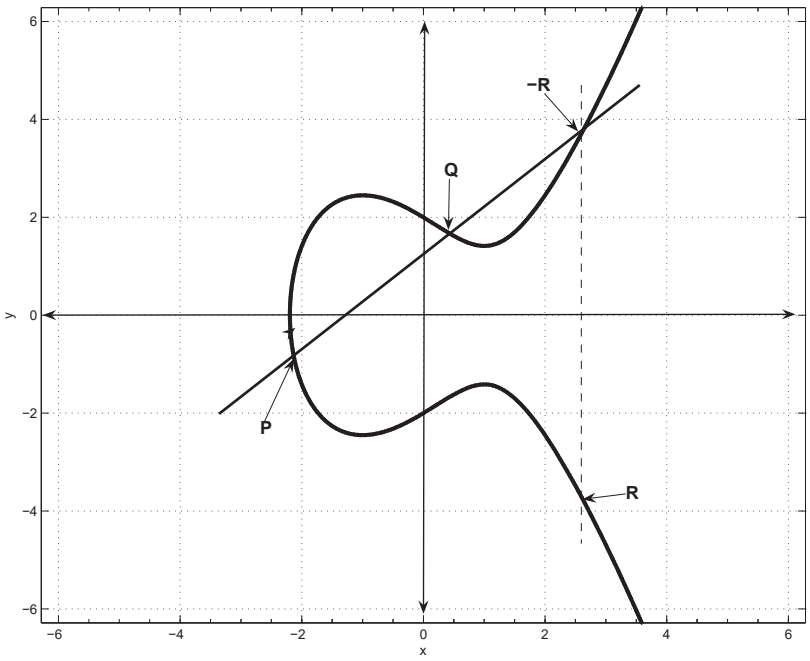



Figura 2.2: Descripción geométrica de la suma de dos puntos distintos de la curva elíptica: $P + Q = R$.

Sea $P = (x_p, y_p)$. Si se suma el punto $P$ consigo mismo $(P + P)$, se dice que se duplica el punto $P$ $(2P)$. El resultado de dicha operación es un punto sobre la curva denotado por $R = (x_r, y_r)$. Para está operación se tiene que considerar dos casos:

1. Cuando $y_p \neq 0$: Primero se traza una línea tangente al punto $P$ de la curva elíptica. Esta línea intersecta a la curva elíptica en otro punto, $-R$. Entonces $-R$ es la reflexión en el eje de las $x's$ a $R$. Esta operación se puede observar geométricamente en la figura 2.4.

2. Cuando $y_p = 0$: La línea tangente a la curva elíptica en el punto $P$ es vertical y no intersecta a la curva elíptica en otro punto. Por definición $2P = \mathcal{O}$. Véase la figura 2.5.

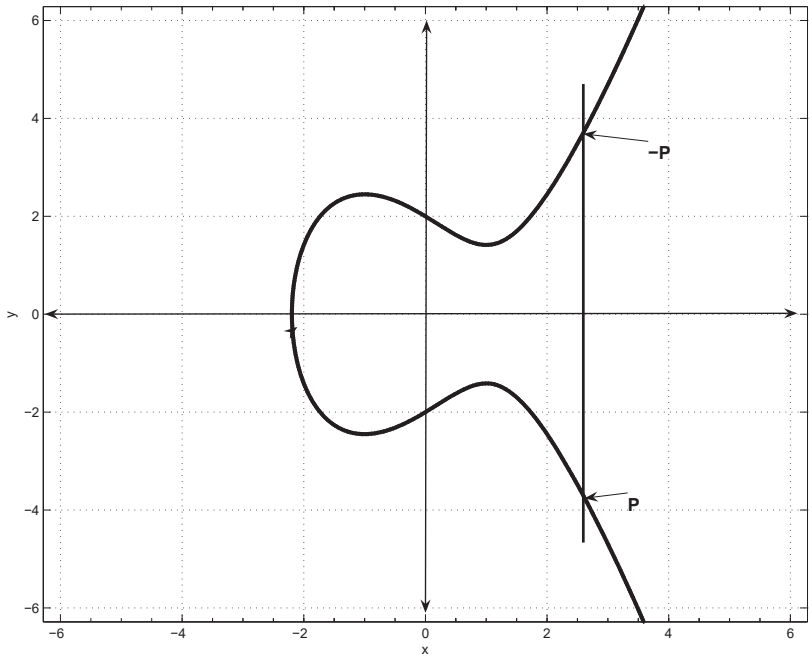



Figura 2.3: Descripción geométrica de la suma de dos puntos iguales en la curva elíptica: $P + (-P) = \mathcal{O}$.

Las fórmulas algebraicas siguientes pueden se deducidas de la representación geométrica.

1. **Identidad:** $P + \mathcal{O} = \mathcal{O} + P = P$ para todo $P \in E(\mathbb{F}_p)$.

2. **Negativo:** Si $P = (x, y) \in E(\mathbb{F}_p)$, entonces tendremos $(x, y) + (x, -y) = \mathcal{O}$. El punto $(x, -y)$ es denotado como $-P$, y es llamado el negativo. Obsérvese que $-P$ es realmente un punto sobre la curva elíptica $(E)$.

3. **Suma de un punto:** Sea $P = (x, y) \in E(\mathbb{F}_p)$ y $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde

$$\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)}, \tag{2.9}$$

$$x_3 = \lambda^2 - x_1 - x_2, \tag{2.10}$$

$$y_3 = \lambda(x_1 - x_3) - y_1. \tag{2.11}$$

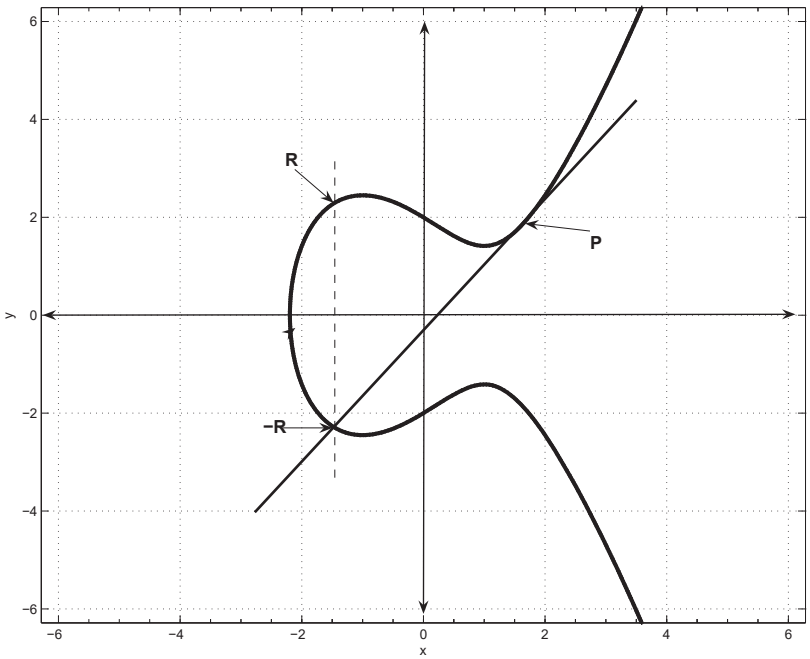



Figura 2.4: Descripción geométrica del doble de un punto sobre una curva elíptica: $P+P=R$, cuando $y_p \neq 0$.

4. **Doblando un punto:** Sea $P=(x_1,y_1) \in E(\mathbb{F}_p)$ donde $P \neq -P$. Entonces $2P=(x_3,y_3)$, donde

$$\lambda = \frac{(3x_1^2+9)}{2y_1}, \tag{2.12}$$

$$x_3 = \lambda^2 - 2x_1, \tag{2.13}$$

$$y_3 = \lambda^2(x_1-x_3) - y_1. \tag{2.14}$$

Observando la operación de $\lambda$, se puede apreciar que es la misma que se hace para encontrar la pendiente de la recta, visto desde el punto de vista de geometría analítica.

En el ejemplo 2.8, se definen dos puntos diferentes sobre la curva y se calcula la suma de estos dos, obteniendo un tercero.

**Ejemplo 2.8** *Suma de un par de puntos.* Tomando en cuenta los datos de la curva elíptica vista en el ejemplo 2.7. Sea $P=(4,7)$ y $Q=(13,11)$, entonces $P+Q=(x_3,y_3)$ es calculado como sigue:

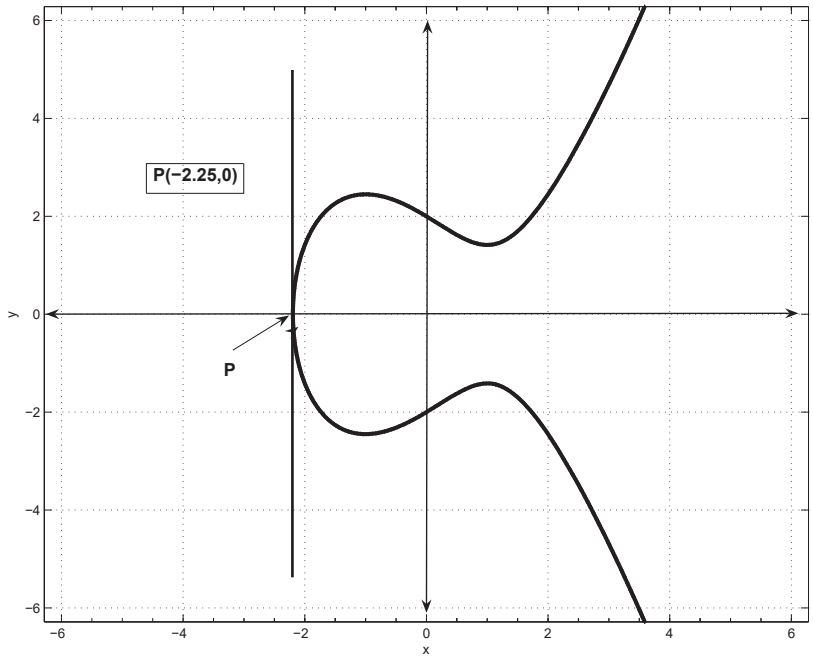



Figura 2.5: Descripción geométrica del doble de un punto sobre una curva elíptica: $P + P = R$, cuando $y_p = 0$.

Considerando la ecuación 2.9 tenemos que

$$\begin{aligned} \lambda &= \frac{11-7}{13-4} \\ &= \frac{4}{9} \\ &= 3, \end{aligned}$$

teniendo ya el valor de $\lambda$ se puede calcular la coordenada $(x_3, y_3)$. Haciendo uso de la ecuación 2.10 se calcula el valor de $x_3$

$$\begin{aligned} x_3 &= 3^2 - 4 - 13 \\ &= 15, \end{aligned}$$

ahora con la ecuación 2.11 se puede calcular el valor de $y_3$

$$\begin{aligned} y_3 &= 3(4-15) - 7 \\ &= 6. \end{aligned}$$

Así, se tiene que la suma de los puntos $P + Q$ es igual a el punto $(15, 6)$.

Para entender la operación de duplicar un punto, en el ejemplo 2.9 se ilustra como se lleva a cabo este proceso.

**Ejemplo 2.9** *Duplicando un punto.* Considerando los datos de la curva elíptica vista en el ejemplo 2.7. Sea $P = (4, 7)$, entonces $2P = P + P = (x_3, y_3)$ es calculado como sigue:

Considerando la ecuación 2.12 tenemos que

$$\begin{aligned} \lambda &= \frac{3(4^2) + 1}{14} \\ &= \frac{49}{14} \\ &= 15, \end{aligned}$$

teniendo ya el valor de $\lambda$, se puede calcular las coordenadas $(x_3, y_3)$. Haciendo uso de la ecuación 2.14 se calcula $x_3$

$$\begin{aligned} x_3 &= 15^2 - 2(4) \\ &= 217 \\ &= 10, \end{aligned}$$

ahora con la ecuación 2.14 se obtiene el valor de $y_3$

$$\begin{aligned} y_3 &= 15(4 - 10) - 7 \\ &= -97 \\ &= 18. \end{aligned}$$

Por lo tanto se tiene que el doble del punto $P$ es igual a $(10, 18)$.

### 2.4.3. Curvas elípticas sobre el campo $\mathbb{F}_{2^m}$

Ahora se considerará a las curvas elípticas definidas sobre los campos con característica 2, es decir sobre los campos $\mathbb{F}_{2^m}$. Una curva elíptica $E_{(a,b)}$ sobre el campo $\mathbb{F}_{2^m}$ esta definido por una ecuación de la forma:

$$E_{(a,b)} = \{(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m} : y^2 + xy = x^3 + ax^2 + b; b \neq \mathcal{O}\} \cup \{\mathcal{O}\}.$$

**Ejemplo 2.10** *Curva elíptica sobre el campo* $\mathbb{F}_{2^4}$. Tomando en cuenta el campo finito del ejemplo 2.3, se tiene la característica del campo $p = 2$ y su extensión $m = 4$, siendo el polinomio irreducible

$$f(x) = x^4 + x + 1.$$

Ahora considerando la curva elíptica $E : y^2 + xy = x^3 + \alpha^4 x^2 + 1$ sobre el campo $\mathbb{F}_{2^4}$. Se tiene que $a = \alpha^4$ y $b = 1$. Recuérdese que $\alpha$ corresponde al elemento generador ($\alpha = x = (0010)$) del campo $\mathbb{F}_{2^4}$, por lo tanto $\alpha^4 = (0011)$ corresponde a la potencia 4 del elemento generador.

Obsérvese que $b \neq 0$, entonces se puede decir que $E$ es realmente una curva elíptica. Los puntos que satisfacen a la curva ($E(\mathbb{F}_{2^4})$) junto con el punto al infinito ($\mathcal{O}$) son:

| | | | |
|---|---|---|---|
| $\mathcal{O}$ | $(1, \alpha^6)$ | $(1, \alpha^{13})$ | $(\alpha^3, \alpha^8)$ |
| $(\alpha^3, \alpha^{13})$ | $(0, 1)$ | $(\alpha^5, \alpha^{11})$ | $(\alpha^6, \alpha^8)$ |
| $(\alpha^6, \alpha^{14})$ | $(\alpha^9, \alpha^{10})$ | $(\alpha^5, \alpha^3)$ | $(\alpha^{10}, \alpha)$ |
| $(\alpha^{10}, \alpha^8)$ | $(\alpha^{12}, 0)$ | $(\alpha^{12}, \alpha^{12})$ | $(\alpha^9, \alpha^{13})$ |

Se puede verificar que cada uno de los valores de los puntos que pertenecen a la curva elíptica ($E(\mathbb{F}_{2^4})$), se encuentran dentro del campo finito $\mathbb{F}_{2^4}$.

Como se explicó en la subsección 2.4.2, donde se tratan las curvas elípticas sobre el campo $\mathbb{F}_p$, la regla de la tangente y la cuerda para sumar un par de puntos sobre una curva elíptica ($E(\mathbb{F}_{2^m})$), se obtiene un tercer punto que pertenece a la curva elíptica. Junto con la operación de la suma, el conjunto de puntos que satisfacen a la curva $E(\mathbb{F}_{2^m})$ forman un grupo con el punto al infinito ($\mathcal{O}$) que sirve como su identidad.

Así también como en las curvas elípticas sobre $\mathbb{F}_p$, se pueden representar las fórmulas algebraicas para la suma de dos puntos distintos, así como el doble de un punto sobre la curva. Para esto es necesario mencionar las leyes de grupo para $E$ sobre $\mathbb{F}_{2^m}$ como sigue:

1. **Identidad:** $P + \mathcal{O} = \mathcal{O} + P = P$ para todo $P \in E(\mathbb{F}_{2^m})$.

2. **Negativo:** Si $P = (x, y)$ y $(x, y) \in E(\mathbb{F}_{2^m})$, entonces $(x, y) + (x, x + y) = \mathcal{O}$. El punto $(x, x+y)$ es denotado como $-P$ y es llamado el negativo de $P$, ya que $-P$ es realmente un punto en la curva ($E(\mathbb{F}_{2^m})$).

3. **Suma de puntos:** Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, tal que $P, Q \in E(\mathbb{F}_{2^m})$, los puntos sobre la curva, donde $P \neq \pm Q$. Entonces tendremos que la suma de los puntos

$P + Q$ sería como resultado un punto $R = (x_3, y_3)$, donde

$$\lambda = \frac{(y_1 + y_2)}{(x_1 + x_2)}, \tag{2.15}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \tag{2.16}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1. \tag{2.17}$$

4. **Doble de un punto:** Sea $P = (x_1, y_1)$, tal que $P \in E(\mathbb{F}_{2^m})$, donde $P \neq -P$. Entonces $2P = (x_3, y_3)$, donde

$$\lambda = \frac{(x_1 + y_1)}{x_1}, \tag{2.18}$$

$$x_3 = \lambda^2 + \lambda + a \tag{2.19}$$

$$= x_1^2 + \frac{b}{x_1^2},$$

$$y_3 = x_1^2 + \lambda x_3 + x_3. \tag{2.20}$$

recuerde que $\lambda$ representa la pendiente de la recta.

Para la suma de puntos se puede apreciar ejemplo 2.11. En el ejemplo 2.12 se observa la duplicación de un punto.

**Ejemplo 2.11** *Suma de puntos.* Considerando la curva elíptica y la información que se definió en el ejemplo 2.10. Sean $P = (\alpha^6, \alpha^8)$ y $Q = (\alpha^3, \alpha^{13})$ dos puntos que se encuentran sobre la curva. Entonces la suma de estos puntos $P + Q = (x_3, y_3)$ es calculada como sigue.

Primero se calcula la pendiente, para ello se hace uso de la ecuación 2.15

$$\lambda = \frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3}$$

$$= \frac{\alpha^3}{\alpha^2},$$

ahora se calcula el valor de $x$, esto con la ecuación 2.16

$$x_3 = \left(\frac{\alpha^3}{\alpha^2}\right)^2 + \frac{\alpha^3}{\alpha^2} + \alpha^6 + \alpha^3 + \alpha^4$$

$$= 1,$$

y se calcula el valor de $y$, con ayuda de la ecuación 2.17

$$y_3 = \left(\frac{\alpha^3}{\alpha^2}\right)(\alpha^{13}) + \alpha^2$$

$$= \alpha^{13}.$$

Por lo tanto la suma de los puntos $P + Q$ es $(1, \alpha^{13})$.

**Ejemplo 2.12** *Doble de un punto.* Considerando la curva elíptica y la información que se definió en el ejemplo 2.10. Sea $P = (\alpha^6, \alpha^8)$ un punto que se encuentran sobre la curva. Entonces el doble de este punto $2P = (x_3, y_3)$ es calculado como sigue.

Primero se calcula la pendiente, para ello se hace uso de la ecuación 2.18

$$\begin{aligned} \lambda &= \frac{\alpha^6 + \alpha^8}{\alpha^6} \\ &= \alpha^6 + \frac{\alpha^8}{\alpha^6}, \end{aligned}$$

ahora se calcula el valor de $x$, esto con la ecuación 2.19

$$\begin{aligned} x_3 &= (\alpha^6)^2 + \frac{1}{(\alpha^6)^2} \\ &= \alpha^{12} + \alpha^3 \\ &= \alpha^{10}, \end{aligned}$$

y con la ecuación 2.20 se calcula el valor de $y$,

$$\begin{aligned} y_3 &= (\alpha^6)^2 + \left(\alpha^6 + \frac{\alpha^8}{\alpha^6}\right)\alpha^{10} + \alpha^{10} \\ &= \alpha^{12} + \alpha^{13} + \alpha^{10} \\ &= \alpha^8. \end{aligned}$$

Por lo tanto el doble de $P$ es $(\alpha^{10}, \alpha^8)$.

### 2.4.4. Propiedades básicas

#### Orden de una curva elíptica

Sea $E$ una curva elíptica definida sobre $\mathbb{F}_q$. El número de puntos en $E(\mathbb{F}_q)$ (incluyendo el punto al infinito, $\mathcal{O}$), denotado por $\#E(\mathbb{F}_q)$, es llamado el *orden* de una curva elíptica.

El siguiente teorema, conocido como el *teorema de Hasse*, provee el límite para $\#E(\mathbb{F}_q)$.

**Teorema 2.2 (Hasse)** *Sea $E$ una curva elíptica definida sobre $\mathbb{F}_q$, entonces*

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

Al intervalo $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$ se le llama, el *intervalo de Hasse*. Una alternativa para la formulación del teorema de Hasse es el siguiente: si $E$ está definida sobre $\mathbb{F}_q$, entonces $\#E(\mathbb{F}_q) = q+1-t$ donde $|t| \leq 2\sqrt{q}$; $t$ es llamado la *traza* de la curva elíptica $(E)$ sobre $\mathbb{F}_q$. Ya que $2\sqrt{q}$ es relativamente pequeña para $q$, tenemos que $\#E(\mathbb{F}_q) \approx q$.

Sea $E$ una curva elíptica y $P$ un punto sobre la curva. Si $k$ es un entero positivo, definimos la potencia $k$ de $P$, denotado por $kP$, como

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

Ahora bien, denotamos el *orden de un punto* $(P)$, como $ord(P)$, como el menor entero $k$ tal que $kP = \mathcal{O}$ (el punto al infinito).

**Estructura de un grupo**

El siguiente teorema describe la estructura de un grupo $E(\mathbb{F}_q)$. Se usa $\mathbb{Z}_n$ para denotar un grupo cíclico de orden $n$.

**Teorema 2.3 (Estructura)** *Sea $E$ una curva elíptica definida sobre el campo $\mathbb{F}_q$, entonces $E(\mathbb{F}_q)$ es isomorfo a $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$; donde $n_1$ y $n_2$ son únicamente determinados enteros positivos tal que $n_2$ divide tanto a $n_1$ como a $q-1$.*

Note que $\#E(\mathbb{F}_q) = n_1 n_2$. Si $n_2 = 1$, entonces $E(\mathbb{F}_q)$ es un *grupo cíclico*. Si $n_2$ es un entero pequeño (por ejemplo, $n = 2, 3$ ó $4$) se dice que es *casi cíclico*. En el caso cíclico, $E(\mathbb{F}_q)$ es *isomorfa* a $\mathbb{Z}_{n_1}$ y existe un punto que pertenece a $E(\mathbb{F}_q)$, con lo que:

$$E(\mathbb{F}_q) = \{kP : 0 \leq k \leq n_1 - 1\};$$

tal punto es llamado un *generador* de la curva elíptica sobre el campo finito $\mathbb{F}_q)$. Véase el ejemplo 2.13.

**Ejemplo 2.13** *Estructura de un grupo.* Considerando el campo finito del ejemplo 2.1. Sea la curva elíptica siguiente $E : y^2 = x^3 + 4x + 20$, definida sobre $\mathbb{F}_{29}$. Se tiene que $\#E(\mathbb{F}_{29}) = 37$. Ya que 37 es un número primo, $E(\mathbb{F}_{29})$ es un grupo cíclico y algún punto en $E(\mathbb{F}_{29})$, excepto $\mathcal{O}$, es un generador de $E(\mathbb{F}_{29})$. Enseguida se multiplican los múltiplos de $(1, 5)$ generando los puntos en $E(\mathbb{F}_{29})$:

| | | | |
|---|---|---|---|
| $1P = (1, 5)$ | $2P = (17, 4)$ | $3P = (4, 19)$ | $4P = (20, 3)$ |
| $5P = (6, 12)$ | $6P = (17, 19)$ | $7P = (24, 22)$ | $8P = (8, 10)$ |
| $9P = (14, 22)$ | $10P = (13, 23)$ | $11P = (10, 25)$ | $12P = (19, 13)$ |
| $13P = (16, 27)$ | $14P = (5, 22)$ | $15P = (3, 1)$ | $16P = (0, 22)$ |
| $17P = (27, 2)$ | $18P = (2, 22)$ | $19P = (2, 6)$ | $20P = (27, 22)$ |
| $21P = (0, 7)$ | $22P = (3, 28)$ | $23P = (5, 7)$ | $24P = (16, 2)$ |
| $25P = (19, 6)$ | $26P = (10, 4)$ | $27P = (13, 6)$ | $28P = (14, 6)$ |
| $29P = (8, 19)$ | $30P = (24, 7)$ | $31P = (17, 10)$ | $32P = (6, 17)$ |
| $33P = (15, 2)$ | $34P = (20, 26)$ | $35P = (4, 10)$ | $36P = (1, 24)$ |
| $37P = \mathcal{O}$ | | | |

Ahora bien, el ejemplo 2.14 muestra la estructura del grupo en un campo finito característico 2.

**Ejemplo 2.14** *Estructura de un grupo en* $\mathbb{F}_{2^4}$. Considerando el campo $\mathbb{F}_{2^4}$ visto en el ejemplo 2.3, representado por la reducción polinomial $f(x) = x^4 + x + 1$. La curva elíptica $y^2 + xy = x^3 + \alpha^3 x^2 + (\alpha^3 + 1)$ definida sobre $\mathbb{F}_{2^4}$ que tiene $\#E(\mathbb{F}_{2^4}) = 22$. Ya que 22 no tiene factores repetidos, $E(\mathbb{F}_{2^4})$ es cíclico. Considerando el punto $P = (\alpha^3, 1) = (1000, 0001)$ de orden 11; sus múltiplos se muestran a continuación:

| | |
|---|---|
| $0P = \mathcal{O}$ | $5P = (1011, 0010)$ |
| $1P = (1000, 0001)$ | $6P = (1011, 1001)$ |
| $2P = (1001, 1111)$ | $7P = (1111, 0100)$ |
| $3P = (1100, 0000)$ | $8P = (1100, 1100)$ |
| $4P = (1011, 0010)$ | $9P = (1001, 0110)$ |
| | $10P = (1000, 1001)$ |

### 2.4.5. Problema del logaritmo discreto de curvas elípticas (PLDCE)

**Definición:** Sea $G$ un grupo cíclico finito de orden $n$. Sea $\alpha$ un generador de $G$ y sea $\beta \in G$. El *logaritmo discreto* de $\beta$ a la base $\alpha$, denotado $log_\alpha \beta$ es el único entero $x$, $0 \leq x \leq n - 1$, tal que $\beta = \alpha^x$.

El *problema del logaritmo discreto* es, dado un grupo cíclico finito $G$ de orden $n$, un generador $\alpha \in G$ y un elemento $\beta \in G$, encontrar el entero $x$, $0 \leq x \leq n - 1$, tal que $\alpha^x = \beta$.

Ahora definiendo para el grupo de puntos de una curva elíptica sobre un campo finito. Sea $E$ una curva elíptica definida sobre un campo finito $\mathbb{F}_q$ y sea $P \in E(\mathbb{F}_q)$ y $Q \in E(\mathbb{F}_q)$ dos puntos sobre $E$ de orden $n$, el *Problema del Logaritmo Discreto de curvas elípticas* (PLDCE) consiste en encontrar un entero $k$, $0 \leq k \leq n-1$, suponiendo que existe, tal que $Q = kP$, como se observa en el ejemplo 2.15.

**Ejemplo 2.15** Sea la siguiente curva elíptica $E : y^2 = x^3 + 9x + 17$ sobre el campo $\mathbb{F}_{23}$. Determinar el logaritmo discreto $k$ de $Q = (4, 5)$ sobre la base $P = (16, 5)$. Para solucionar este problema, se buscan los múltiplos de $P$ hasta encontrar $Q$. Los múltiplos de $P$ son:

$$\begin{array}{ll} 1P = (16, 5) & 2P = (20, 20) \\ 3P = (14, 14) & 4P = (19, 20) \\ 5P = (13, 10) & 6P = (7, 3) \\ 7P = (8, 7) & 8P = (12, 17) \\ 9P = (4, 5) & \end{array}$$

Puesto que $9P = (4, 5) = Q$, el logaritmo discreto de $Q$ con base en $P$ es $k = 9$.

En una aplicación real, el valor de $k$ es lo suficientemente grande para hacer imposible o casi imposible intentar romperlo por fuerza bruta.

### 2.4.6. Curvas de Koblitz

Las curvas de Koblitz, también conocidas como *curvas binarias anómalas*, son curvas elípticas definidas sobre $\mathbb{F}_2$. La principal ventaja de esas curvas es que los algoritmos de multiplicación de un punto, pueden ser pensados para que no usen alguna duplicación de punto. Muchos autores, si no es que la gran mayoría, coinciden en que una buena referencia para este tema se encuentra en [Solinas00].

Las curvas de Koblitz son las siguientes curvas elípticas definidas sobre $\mathbb{F}_2$:

$$E_0 \quad : \quad y^2 + xy = x^3 + 1 \tag{2.21}$$

$$E_1 \quad : \quad y^2 + xy = x^3 + x^2 + 1 \tag{2.22}$$

Por lo regular, los sistemas criptográficos se encuentran dentro del grupo $E_0(\mathbb{F}_{2^m})$ ó $E_1(\mathbb{F}_{2^m})$ de los puntos racionales $\mathbb{F}_{2^m}$ para alguna extensión del campo $\mathbb{F}_{2^m}$. Sea $a \in \{0, 1\}$. Para cada divisor propio $l$ de $m$, $E_0(\mathbb{F}_{2^l})$ es un subgrupo de $E_a(\mathbb{F}_{2^m})$ y de ahí el orden de la curva elíptica $\#E_0(\mathbb{F}_{2^l})$ dividido entre $\#E_a(\mathbb{F}_{2^m})$. En particular, ya que $E_0(\mathbb{F}_2) = 4$ y $E_1(\mathbb{F}_2) = 2$, $E_0(\mathbb{F}_{2^m})$ es un múltiplo de 4 y $E_1(\mathbb{F}_{2^m})$ es un múltiplo de 2.

Ahora bien, una curva de Koblitz $E_a$ tiene el orden de grupo casi primo sobre $\mathbb{F}_{2^m}$ si $\#E_a(\mathbb{F}_{2^m}) = hn$ donde $n$ es un primo y $h = 4$ si $a = 0$ ó $h = 2$ si $a = 1$, donde $h$ es llamado el *cofactor*.

La tabla 2.3 lista los extensión ($m \in [100, 600]$), y las curvas de Koblitz $E_a$ para las cuales $\#E_a(\mathbb{F}_{2^m})$ es casi primo.

| $m$ | **Curva** | **Factorización del primo** $\#E(F_{2^m})$ |
|---|---|---|
| 101 | $E_1$ | $2 \cdot 1267650600228230886142808508011$ |
| 103 | $E_0$ | $2^2 \cdot 2535301200456459535862530067069$ |
| 107 | $E_1$ | $2 \cdot 81129638414606692182851032212511$ |
| 233 | $E_0$ | $2^2 \cdot 345087317339528189371737793113851276057094098886225212632808702474134 3$ |
| 409 | $E_0$ | $2^2 \cdot 330527984395124299475957654016385519914202341482140609642324395022880711289249191050673258457777458014096366590617731358671$ |
| 571 | $E_0$ | $2^2 \cdot 193226876150862917234767594546599367214946366485321749932861762572575957114478021226813397852270671183470671280082535146127367497406661731192968242161709250355573368 5276673$ |

Cuadro 2.3: Curvas de Koblitz con un grupo de casi primos.

## 2.5. Resumen

En esté capítulo se explicó (aunque no fue de manera exhaustiva) los aspectos más generales que tienen los grupos. También se examinaron los campos finitos, donde se enfocó más precisamente en dos de ellos, que fueron: campos finitos primos y campos finitos binarios. Los temas de grupos y campos finitos, permiten tener una base para ahondar en el tema de curvas elípticas, que a su vez permite definir la curva elíptica, la forma como se representa gráficamente, propiedades importantes de las curvas y las operaciones aritméticas que se realizan sobre los campos finitos. Se trata a su vez, el problema del logaritmo discreto de las curvas elípticas y finalmente se describen las curvas elípticas de Koblitz.

El objetivo en todo momento fue explicar los conceptos generales, y tratar de no entrar en detalles en las definiciones o teoremas que fueron dados, aunque en ocasiones fue imposible evitarlos.

Capítulo 3

# Criptografía de curvas elípticas

En esté capítulo se revisan los algoritmos para generar la clave pública y privada, cifrar y descifrar, un esquema de firma digital mediante curvas elípticas y un algoritmo para generar una clave compartida de curvas elípticas (protocolo de acuerdo de claves).

Antes de entender la mecánica de cada uno de los algoritmos, es preciso, obtener una curva elíptica que este sobre un campo finito. Es decir, se deben de definir los requerimientos del campo, así como los de la curva. Para esto, se puede hacer uso de dos métodos que permiten generar aleatoriamente una curva elíptica. Cada método depende del orden del campo finito.

## 3.1. Introducción

Existen numerosos artículos y libros que tratan el tema de ECC [Smart97, Torii00, Belingueres, Al-Kalayi04, Qizhi04, Hankerson04, López05, Strangio05]. En el desarrollo de este capítulo, se consideraron como fuente de información a [X9.6299, X9.63rd, Johnson01, García03].

## 3.2. Dominio de parámetros

El *dominio de parámetros* consiste de una apropiada elección de la curva elíptica $E$, definida sobre un campo finito $\mathbb{F}_q$ de característica $p$, y un punto de base $P$ que pertenezca a $E(\mathbb{F}_q)$. El dominio de parámetros puede, entre otros, ser compartido por un grupo de entidades ó especificado a una única entidad.

Para facilitar la interoperabilidad, algunas restricciones son puestas sobre el tamaño del campo subyacente $q$ y la representación usada por los elementos de $\mathbb{F}_q$.

### 3.2.1. Requerimientos del campo

El orden del campo finito subyacente, tal que $q = p$, siendo un número primo impar o una potencia de 2 ($q = 2^m$).

En el caso donde $q = 2^m$, el campo finito subyacente es $\mathbb{F}_{2^m}$ cuyos elementos son representados con respecto a una base polinomial o una base normal.

### 3.2.2. Requerimientos de la curva elíptica

La razón por la que se debe de poner mucho cuidado en la elección de los requerimientos de la curva elíptica, son los posibles ataques que se pueden realizar. Tanto para descubrir el PLDCE [Pohlig78, Pollard78], como en la selección de la curva [Menezes91, Frey94]. Así como aquellos donde el objetivo son las curvas sobre campos finitos anómalos [Smart97], es decir donde $E(\mathbb{F}_q) \neq q$.

Una forma prudente de resguardarse contra ataques (y ataques similares contra clases especiales de curvas que pueden ser descubiertos en el futuro), es seleccionando aleatoriamente la curva elíptica $E$, sujeta a la condición de que $\#E(\mathbb{F}_q)$ sea divisible por un primo grande. La probabilidad de que una curva escogida aleatoriamente, sucumba a los ataques de propósito general suele ser insignificante. Una curva puede ser seleccionada de manera que se pueda corroborar la aleatoriedad (que sea verificablemente aleatoria), escogiendo los coeficientes de la ecuación que define a la curva elíptica como la salida de una función de un camino (*"one-way"*) [García03].

De manera resumida, los parámetros del dominio se considera como un séptupla

$$D = (q, FR, a, b, P, n, h),$$

donde cada uno de los elementos representan a:

1.- Un número $q$ que específica una potencia primo $q = p$ ó $q = 2^m$.

2.- $FR$, que indica el método usado para la representación de los elementos del campo en $\mathbb{F}_q$.

3.- Dos elementos $a, b \in \mathbb{F}_q$, que son parámetros de la ecuación de la curva elíptica $E$ sobre $\mathbb{F}_q$ (es decir, $y^2 = x^3 + ax + b$ en el caso de que $p > 3$, y $y^2 + xy = x^3 + ax^2 + b$ cuando $p = 2$).

4.- Dos elementos $x_P$ y $y_P$ en $\mathbb{F}_q$ los cuales definen un punto finito $P = (x_P, y_P)$ (punto base) de orden primo en $E(\mathbb{F}_q)$.

5.- El orden de $P$ representado por $n$, el cual debe de cumplir que $n \geq 2^{160}$ y $n > 4\sqrt{q}$ [X9.6299].

6.- Un entero $h$ el cual es el cofactor $h = \frac{\#E(\mathbb{F}_q)}{n}$.

## 3.3. Generando una curva elíptica verificablemente aleatoria

Existen dos maneras de obtener curvas elípticas que sean aleatorias. Una es tomando las que ofrecen ciertas organizaciones, que en el apéndice B se puede encontrar un poco más de información y la otra es generarla usando algoritmos

Para generar curvas elípticas que sean verificablemente aleatorias, se consideran dos métodos: cuando $q = p$ y para $q = 2^m$.

### 3.3.1. Método 1. Caso $q = p$

Sea

$$t = \lceil log_2 p \rceil, \tag{3.1}$$

$$s = \lfloor \frac{(t-1)}{160} \rfloor \tag{3.2}$$

$$v = t - 160 \cdot s. \tag{3.3}$$

El algoritmo para generar la curva elíptica se puede apreciar en el algoritmo 3,1 [Hankerson04].

Después de generar la curva elíptica, resulta conveniente verificar que la curva fuese generada aleatoriamente sobre el campo $\mathbb{F}_p$. Para ello se tiene el algoritmo 3,2 [Hankerson04].

**Algoritmo 3.1**: Generando una curva elíptica aleatoria sobre el campo $\mathbb{F}_p$.

**Entrada**: Un tamaño de campo $p$, donde $p$ es un primo impar y $SHA-1$ una función hash.

**Salida** : Una cadena de bits llamada $semE$ de longitud de al menos 160-bits. Un campo de elementos $a, b \in \mathbb{F}_p$ que defina una curva elíptica $E$ sobre $\mathbb{F}_p$.

**Inicio**

Escoger una cadena de bits $semE$ de longitud $g \geq 160$-bits.

Calcular $H = SHA-1(semE)$, y sea $c_0$ denotado la cadena de bits de longitud $v$, ecuación 3.3, obtenida de tomar $v$ más a la derecha de bits de $H$.

Sea $W_0$ denotado por la cadena de bits de longitud $v$, obtenida por poner el bit más a la izquierda de $c_0$ a 0. Esto asegura que $r < p$.

Sea $z$ el entero cuya expansión binaria es dada por la cadena $g$-bit $semE$.

**para** $i = 1$ **hasta** $s$ **hacer**

Sea $s_i$ la cadena de $g$-bit el cual es la expansión binaria del entero $(z + i)$ mód $2^g$.

Calcular $W_i = SHA-1(s_i)$.

Sea $W$ la cadena de bits obtenida por concatenar $W_0, W_1, \ldots, W_s$ como sigue

$$W = W_0||W_1||\cdots||W_s.$$

Sea $r$ el entero cuya expansión binaria es dada por $W$.

**si** $r == 0$ *ó* $4r + 27 \equiv 0 \pmod{p}$ **entonces**

pasar al paso 1.

Escoger enteros arbitrarios $a, b \in \mathbb{F}_p$, que no sean ambos 0, tal que $r \cdot b^2 \equiv a^3$ mód $p$. Por ejemplo, una forma de tomarla sería $a = r$ y $b = r$.

La curva elíptica seleccionada sobre $\mathbb{F}_p$ es $E : y^2 = x^3 + ax + b$.

**Fin**

**Algoritmo 3.2**: Generando una curva elíptica aleatoria sobre el campo $\mathbb{F}_p$.

**Entrada**: Un tamaño de campo $p$, donde $p$ es un primo impar y sea $SHA-1$ una función hash. Un tamaño de campo $p$, donde $p$ es un primo impar y sea $SHA-1$ una función hash. Un tamaño de campo $p$, donde $p$ es un primo impar y sea $SHA-1$ una función hash.

**Salida** : Aceptar o rechazar que la curva fue aleatoriamente generada.

**Inicio**

Calcular $H = SHA-1(semE)$ y sea $c_0$ la cadena de bits de longitud $v$, obtenida de tomar los $v$ bits más a la derecha de $H$.

Sea $W_0$ indicando la cadena de bits de longitud $v$ bits obtenida de poner el bit más a la izquierda de $c_0$ a $0's$.

Sea $z$ el entero cuya expansión binaria es dada por la cadena $g$-bit $semE$.

**para** $i = 1$ **hasta** $s$ **hacer**

  Sea $s_i$ la cadena de $g$-bits la cual es la expansión binaria del entero $(z+i)$ mód $2^g$.

  Calcular $W_i = SHA-1(s_i)$.

Sea $W'$ la cadena de bits obtenida por concatenar $W_0, W_1, \ldots, W_s$ como sigue

$$W' = W_0 \| W_1 \| \cdots \| W_s.$$

Sea $r'$ el entero cuya expansión binaria es dada por $W'$.

**si** $r' \cdot b^2 \equiv a^3 \pmod{p}$ **entonces**

  se acepta.

**sino**

  se rechaza.

**Fin**

### 3.3.2. Método 2. Caso $q = 2^m$

La siguiente notación será usada:

$$s = \lfloor \frac{(m-1)}{160} \rfloor \tag{3.4}$$

$$v = m - 160 \cdot s \tag{3.5}$$

En el caso, donde el campo es $\mathbb{F}_{2^m}$, el algoritmo 3.3 es el encargado de generar la curva.

**Algoritmo 3.3**: Generando una curva elíptica aleatoria sobre el campo $\mathbb{F}_{2^m}$.

**Entrada**: Un tamaño de campo $q = 2^m$.

**Salida** : Una cadena de bits llamada *semE* con una longitud de al menos 160-bits. Los elementos de campo $a, b \in \mathbb{F}_{2^m}$ el cual se define una curva elíptica $E$ sobre $\mathbb{F}_{2^m}$.

**Inicio**
- Escoger una cadena de bits arbitraria *semE* de longitud $g \geq 160$ bits.
- Calcular $H = SHA-1(semE)$, y sea $b_0$ la cadena de bits de longitud $v$, obtenida de tomar el $v$-bit más a la derecha de $H$.
- Sea $z$ el entero cuya expansión binaria es dada por la cadena $g$-bit *semE*.
- **para** $i = 1$ **hasta** $s$ **hacer**
  - Sea $s_i$ la cadena de $g$-bit, el cual es la expansión binaria del entero $(z + i)$ mód $2^g$.
  - Calcular $b_i = SHA-1(s_i)$.
- Sea $b$ el elemento de campo obtenido de concatenar $b_0, b_1, \ldots, b_s$ como sigue $b = b_0 \| b_1 \| \cdots \| b_s$. **si** $r == 0$ **entonces**
  - ir a el paso 1.
- Sea $a$ un elemento arbitrario de $\mathbb{F}_{2^m}$.
- La curva elíptica seleccionada sobre $\mathbb{F}_{2^m}$ es $E : y^2 + xy = x^3 + ax^2 + b$.

**Fin**

Al igual que el método anterior (ver 3.3.1) ocurre que se puede verificar si la curva elíptica que se generó, es aleatoria sobre $\mathbb{F}_{2^m}$. Para corroborar lo anterior, es necesario seguir el algoritmo 3.4.

## 3.4. Generación de claves

Un par de claves de una curva elíptica es asociado con un particular conjunto de parámetros de dominio $D = (q, FR, a, b, P, n, h)$. La *clave pública* es un punto $Q$ aleatoriamente seleccionado del conjunto $\langle G \rangle$ generado por $G$. La correspondiente *clave privada* es

**Algoritmo 3.4**: Verificando que la curva elíptica fue generada aleatoriamente sobre el campo $\mathbb{F}_{2^m}$

**Entrada**: Un tamaño de campo $q = 2^m$. Una cadena de bits $semE$ de longitud $g \geq 160$ bits. Los elementos del campo $a, b \in \mathbb{F}_{2^m}$ el cual definen una curva elíptica $E : y^2 + xy = x^3 + ax^2 + b$ sobre $\mathbb{F}_{2^m}$.

**Salida** : Aceptar o rechazar que la curva fue aleatoriamente generada, según el algoritmo 3.3.

**Inicio**
- Calcular $H = SHA - 1(semE)$ y sea $b_0$ el bit de la cadena de bits de longitud $v$ obtenida de tomar $v$ bits más a la derecha de $H$.
- Sea $z$ el entero cuya expansión binaria es dada por la cadena $g$-bit $semE$.
- **para** $i = 1$ **hasta** $s$ **hacer**
  - Sea $s_i$ la cadena de $g$-bits la cual es la expansión binaria del entero $(z + i)$ mód $2^g$.
  - Calcular $b_i = SHA - 1(s_i)$.
- Sea $b'$ la cadena de bits obtenida por concatenar $b_0, b_1, \ldots, b_s$ como sigue $b' = b_0 \| b_1 \| \cdots \| b_s$.
- **si** $b == b'$ **entonces**
  - se acepta
- **sino**
  - se rechaza.

**Fin**

obtenida como sigue:

$$d = log_G Q. \tag{3.6}$$

Esta asociación puede ser asegurada criptográficamente (es decir, todas las entidades usan el mismo dominio de parámetros). Es por ello que para generar una clave pública y una privada, se sigue el procedimiento del algoritmo 3.5.

**Algoritmo 3.5**: Generación de un par de claves usando curvas elípticas.

**Entrada**: Parámetros del dominio $D = (q, FR, a, b, P, n, h)$.

**Salida** : Clave pública $Q$. Clave privada $d$.

**Inicio**

  Seleccionar $d \in [1, n-1]$.

  Calcular $Q = dP$.

**Fin**

Se puede observar que tratar de resolver el valor de $d$ (clave privada) a partir de $Q$ (clave pública) es precisamente tratar de resolver el problema del logaritmo discreto de curvas elípticas (PLDCE).

### 3.4.1. Validar la clave pública

Una clave pública es válida si posee ciertas propiedades aritméticas. La validación de la clave pública es muy importante para los protocolos de establecimiento de claves. De manera general, se mencionan dos razones por las que se tiene que llevar a cabo la validación de la clave pública [García03]:

- Prevención de una inserción maliciosa de una clave pública que puede permitir algunos ataques.
- Detectar código malintencionado o los errores de transmisión.

El algoritmo 3.6 desarrolla el proceso para realizar la validación de la clave pública que se generó.

## 3.5. Representación de los puntos de la curva

Cualquier punto sobre la curva elíptica, excepto el punto al infinito $\mathcal{O}$, se representa por la dupla $(x, y)$. De esta manera un punto $P$ puede ser representado como $(x_P, y_P)$. De

**Algoritmo 3.6**: Validación de la clave pública.

**Entrada**: Parámetros del dominio. Clave pública $Q$.

**Salida** : Aceptar o rechazar si $Q$ es válida o no.

**Inicio**
- Verificar que $Q \neq \mathcal{O}$.
- Verificar que $x_Q$ y $y_Q$ son propiamente elementos que estén representados en $\mathbb{F}_q$ (por ejemplo, enteros que se encuentren en el rango $[0, q-1]$ si se trata de $\mathbb{F}_q$, y una cadena de bits de longitud $g$-bits si es el caso de $\mathbb{F}_{2^m}$).
- Verificar que $Q$ satisfaga la ecuación de la curva elíptica definidos por $a$ y $b$.
- Verificar que $nQ = \mathcal{O}$.
- **si** *falla alguna verificación* **entonces**
  - $Q$ no es aceptada como una clave pública válida
- **sino**
  - $Q$ es válida.

**Fin**

manera más compacta un punto se puede representar almacenando la coordenada $x_P$ y un cierto bit nombrado $\tilde{y_P}$ extraído de las coordenadas $x_P$ y $y_P$.

La técnica de compresión de puntos se lleva a cabo sobre los campos $\mathbb{F}_p$ y $\mathbb{F}_{2^m}$.

Para el caso del campo $\mathbb{F}_p$, se tiene un punto $P = (x_P, y_P)$ en la curva $(E : y^2 = x^3 + ax + b)$ sobre $\mathbb{F}_p$ $(E(\mathbb{F}_p))$. Entonces $\tilde{y_P}$ es el bit menos significativo de $y_P$.

El algoritmo 3.7 trata justamente, cuando se proporcionan $x_P \in P$ y el bit $\tilde{y_P}$ y se desea calcular el elemento $y_P$.

Ahora, considérese el campo $\mathbb{F}_{2^m}$, en [García03] se describe una técnica que se puede aplicar cuando los elementos del campo que pertenecen a $\mathbb{F}_{2^m}$ están representados con respecto a una base normal óptima (BNO).

Tomando en cuenta el punto $P = (x_P, y_P)$ que pertenece a la curva elíptica $E$ (tal que, $E : y^2 + xy = x^2 + ax^2 + b$) definida sobre $\mathbb{F}_{2^m}$, entonces $\tilde{y_P}$ es el bit menos significativo del elemento del campo $y_P \cdot x_P^{-1}$.

Por su parte, el algoritmo 3.8 permite recuperar el elemento $y_P$.

## 3.6. Cifrado de clave pública

Los esquemas de clave pública se pueden usar para proveer confidencialidad. En cambio, los esquemas de clave simétrica son considerados lentos en comparación de los antes

**Algoritmo 3.7**: Recuperar $y_P$ a partir de $x_P$ y el bit $\tilde{y_P}$.

**Entrada**: Las coordenadas $x_P$ y $\tilde{y_P}$. Los elementos $a$ y $b$.Valor de $p$.

**Salida** : El elemento $y_P$.

**Inicio**

Calcular

$$\alpha = x^3 + ax_P + b \pmod{p}.$$

Calcular

$$\beta = \sqrt{a \bmod p}.$$

**si** *el bit menos significativo de* $\beta == \tilde{y_P}$ **entonces**

$y_P \leftarrow \beta$

**sino**

$y_P \leftarrow (p - \beta)$.

**Fin**

**Algoritmo 3.8**: Recuperar $y_P$ a partir de $x_P$ y el bit $\tilde{y_P}$.

**Entrada**: Las coordenadas $x_P$ y $\tilde{y_P}$. Los elementos $a$ y $b$.Valor de $m$.

**Salida** : El elemento $y_P$.

**Inicio**

**si** $x_P == 0$ **entonces**

$y_P = b_{m-2} \dots b_1 b_0 b_{m-1}$. Donde $y_P$ se obtiene de un corrimiento cíclico del vector de representación $b$ a la izquierda, es decir , tenemos originalmente a $b = b_{m-2} b_{m-1} \dots b_2 b_1 b_0$.

**sino**

Calcular

$$\alpha = x_P + a + bx_P^{-2}.$$

Sea el vector $\alpha = \alpha_{m-1}\alpha_{m-2} \dots \alpha_1 \alpha_0$.

Se construye el elemento $z = z_{m-1} + z_{m-2} \dots z_1 z_0$.

$z_0 = \tilde{y_P}$.

**para** $i = 1$ **hasta** $m - 1$ **hacer**

$z_i = \alpha_{i-1} \oplus z_{i-1}$.

Retornar el valor de $y_P \leftarrow x_P \cdot z$.

**Fin**

mencionados. Los esquemas de clave simétrica, son usados casi por lo regular únicamente para cifrar datos pequeños y/o transportar claves de sesión.

Los esquemas de clave pública consisten de cuatro algoritmos [García03]:

1.- **Algoritmo de generación de parámetros de dominio:** considera al conjunto $D$ como el dominio de parámetros.

2.- **Algoritmo de generación de claves:** toma como entrada al conjunto $D$ de dominio de parámetros y genera un par de claves $(Q, d)$.

3.- **Algoritmo de cifrado:** toma al conjunto del dominio de parámetros $D$, una clave pública $Q$, un mensaje en texto plano $M$ y produce como salida un texto cifrado $M_c$.

4.- **Algoritmo de descifrado:** toma como entrada el dominio del parámetro $D$, una clave privada $d$ y un texto cifrado $M_c$. Devolverá texto plano $M$ o de lo contrarió a $M_c$ como inválido.

Para los casos de los algoritmos 1 y 2, se recomienda revisar las sección 3.2 en la página 41 y la sección 3.4. A continuación se seguirá el proceso que propone [García03] para un esquema de cifrado de curva elíptica (ECES, *Elliptic Curves Encryption Scheme*).

El proceso que utiliza para el cifrado, consiste de cuatro pasos:

1.- El formateo de bloque a cifrar.

2.- Cálculos sobre la curva elíptica

3.- Inclusión de los datos.

4.- Conversión de puntos a cadena de bytes.

El algoritmo 3.9 describe con mayor detalle estos pasos.

**Algoritmo 3.9:** Cifrando con ECES.

**Entrada**: El texto en claro $M$ en forma de cadena de bytes (la longitud de $M \neq l-2$ bytes, donde $l$ es la longitud del orden del campo $\mathbb{F}_p$ en bytes, es decir $l = \lceil \frac{t}{8} \rceil$, donde $t = \lceil \log_2 q \rceil$). Los parámetros de $D$ y la clave pública $Q$.

**Salida** : Retornará $MC$ que es la cadena en bytes (la longitud de $MC = 2l+1$ bytes si se utiliza la compresión de puntos y $3l+1$ bytes en caso contrario) y representa al texto cifrado.

**Inicio**

*/* Formateo de bloque. */*

Asignar a la izquierda de $M$ con una cadena de relleno $R$ ($R$ consiste de $-2-|M|$ bytes, cada uno con el valor de **ff** (hexadecimal)), seguido de un byte con el valor de 00 (hexadecimal). Con esto se forma una cadena $M'$ de longitud $l-1$ tal que $M' = R\|00\|M$.

*/* Calcular la curva elíptica. */*

Seleccionar un entero aleatorio $k \in [1, n-1]$.

Calcular $(x_1, y_1) = kP$ y $(x_2, y_2) = kQ$.

**si** $x_i r == 0$ **entonces**

  ir a el paso 1.

*/* Inclusión de datos. */*

Convertir $M'$ a una cadena binario, y entonces concatenar una cadena binario de ceros de longitud $8 - 8l + t$ a la izquierda de esta cadena para formar un elemento $m$ del campo.

Calcular $c = m \cdot x_2$ El bit más significativo del elemento $m$ es 0. Esto para asegurar, en el caso en el cual $q$ es igual a un número primo, donde $m <$ módp.

*/* Conversión de un punto a cadena de bytes. */*

Añadir una cadena binaria de ceros de longitud $8l - t$ a la izquierda del elemento $x_1$ del campo.

Convertir la cadena binaria resultante de $8l$ bits en una cadena de bytes $x_1$ de longitud $l$.

Calcular el valor de $\tilde{y_1}$.

**si** *Se utiliza la compresión de puntos entonces* **entonces**

  **si** $\tilde{y_1} == 0$ **entonces**

    $PC = 01$.

  **sino**

    $PC = 03$.

**sino**

  $PC = 00$.

  Añadir una cadena binaria de ceros de longitud $8l - t$ a la izquierda del elemento $y_1$ del campo.

  Convertir la cadena binaria de $8l$ bits resultante a una cadena de bytes, $Y_1$, de longitud $l$.

Añadir una cadena de ceros de longitud $8l - t$ a la izquierda del elemento $c$ del campo.

Convertir la cadena binaria de $8l$ bits resultante a una cadena de bytes $C$ de longitud $l$.

**si** $Y_1$ *es una cadena nula, esto es que se utiliza la compresión de puntos* **entonces**

  $MC = X_1\|PC\|C$.

**sino**

  $MC = X_1\|PC\|Y_1\|C$.

Retorna el texto plano cifrado $MC$, de longitud $2l+1$ ó $3l+1$, según sea el caso.

**Fin**

Ahora bien, para el descifrado de un mensaje que se encuentra cifrado, siguiendo ECES, se tiene que realizar cuatro etapas:

1.- Conversión de la cadena de bytes a puntos.

2.- Cálculos de la curva elíptica.

3.- Extracción de datos.

4.- Análisis de bloque descifrado.

El algoritmo 3.10, muestra el método que hay que seguir para descifrar un mensaje.

**Algoritmo 3.10**: Descifrando con ECES.

**Entrada**: Una cadena de bytes $MC$, que tiene una longitud $2l+1$ ó $3l+1$ según sea el caso, que es el mensaje cifrado. Los parámetros del dominio $D$. La clave privada $d$.

**Salida** : Retornará el mensaje descifrado en una cadena de bytes $M$.

**Inicio**

*/* Conversión de la cadena de bytes a puntos. */*

Analizar el mensaje $MC$ para descomponerlo en $X_1$, $PC$, $Y_1$ y $C$. $X_1$ y $C$ tienen la longitud de $l$ bytes.

$PC$ es un solo byte.

**si** *El bit menos significativo de* $PC$ *es* 1 **entonces**

  $Y_1$ es un byte nulo

**sino**

  $Y_1$ es de $l$ bytes de longitud.

*/* Cálculos sobre la curva elíptica. */*

Usar $x_1$ y $\tilde{y_1}$ para obtener el punto $(x_1, y_1)$ sobre $E$.

Calcular el punto $(x_2, y_2) = d(x_1, y_1)$.

*/* Extracción de los datos. */*

Calcular $m = c \cdot x_2^{-1}$.

Descartar los $8 - 8l + t$ bits más significativos de $m$, y convertir la cadena binaria resultante en una cadena de bytes $M'$ de longitud $l - 1$.

*/* Análisis del bloque de cifrado. */*

Se hace el análisis de $M'$ y se obtiene $R$, $00h$ y $M$.

Retornar el mensaje en claro $M$.

**Fin**

En seguida se muestra a manera de ejemplo, los algoritmo para cifrar y descifrar utilizando el esquema de cifrado de curvas elípticas (ECES).

**Ejemplo 3.1** *Procedimiento de cifrado y descifrado con ECES* Sea $m = 4$ por lo que tenemos un campo subyacente $\mathbb{F}_{2^4}$ y la curva elíptica $E : y^2 + xy = x^3 + \alpha^4 x^2 + 1$ sobre $\mathbb{F}_{2^4}$. Se selecciona un punto $P = (x_P, y_P) = (\alpha^5, \alpha^{11})$. El orden de $P$ es de $n = 5$ ya que $5P = \mathcal{O}$. También tenemos dos entidades llamadas A y B. La entidad A hace lo siguiente:

1.- Selecciona un entero aleatorio, sea $d = 3$, que se encuentre entre $[1, 4]$.

2.- Calcula el punto

$$\begin{aligned} Q &= dP \\ &= 3P \\ &= (\alpha, 0) \\ &= (1100, 0000). \end{aligned}$$

3.- La clave pública es el punto $Q$.

4.- Así, la clave privada será $d = 3$.

Suponiendo que la entidad B envía los datos $M = 0011$ hacia la entidad A. Por consiguiente, la entidad B realiza el siguiente procedimiento para cifrar dichos datos.

1.- Obtiene la clave pública de A, tal que: $Q = (x_Q, y_Q) = (\alpha, 0)$.

2.- La entidad B representa los datos de $M$ como un elemento de campo, es decir: $m = (0111) = \alpha^7$.

3.- Se selecciona aleatoriamente un número entero $k = 2$ en el intervalo $[1, 4]$.

4.- Calcula el punto

$$\begin{aligned} 2P &= (x_1, y_1) \\ &= (\alpha, \alpha) \\ &= (1100, 1100). \end{aligned}$$

5.- Calcula el punto

$$\begin{aligned} 2Q &= (x_2, y_2) \\ &= (\alpha^5, \alpha^{11}) \\ &= (1010, 1110). \end{aligned}$$

6.- Calcular

$$\begin{aligned} c &= m \cdot x_2 \\ &= \alpha^7 \cdot \alpha^5 \\ &= \alpha^{12} = (0001). \end{aligned}$$

7.- Por lo tanto, B transmite los siguientes datos cifrados: $(x_1, y_1, c) = (\alpha, \alpha, \alpha^{12}) = (1100, 1100, 0001)$, hacia la entidad A.

Al momento en que la entidad A recibe los datos cifrados, sigue con el proceso de descifrado. Así pues, la entidad A recibió el texto cifrado $C = (\alpha, \alpha, \alpha^{12})$ que fue enviado por B, llevando a cabo los siguientes pasos para descifrar el mensaje.

1.- Calcula el punto:

$$\begin{aligned} d(x_1, y_1) &= 3(\alpha^5, \alpha^{11}) \\ &= (1010, 1110) \\ &= (x_2, y_2). \end{aligned}$$

2.- Recupera los datos:

$$\begin{aligned} m &= c \cdot x_2^{-1} \\ &= \alpha^{12} \cdot \alpha^{-5} \\ &= \alpha^7 \\ &= (0111). \end{aligned}$$

3.- El texto descifrado es: $m = (0111)$.

## 3.7. Esquema de firma digital

El esquema de firma digital se puede entender como la contra parte de las firmas que son echas a mano. Las firmas digitales son usadas, comúnmente, para proveer la autenticación los datos, la integridad de los datos, y el no rechazo. Quienes por lo regular hacen uso de las firmas digitales son las autoridades de certificados confiables que proporcionan certificados firmados que unen a la entidad y su clave pública.

Un esquema de firma digital consiste de cuatro algoritmos [García03], que son:

1.- **Algoritmo de generación de dominio de parámetros:** genera un conjunto $D$ de dominio de parámetros.

2.- **Algoritmo de generación de claves:** toma como entrada al conjunto $D$ de dominio de parámetros y genera un par de claves $(Q, d)$.

3.- **Algoritmo de generación de firmas:** toma como entrada un conjunto de parámetros de dominio $D$, una clave privada y un mensaje $m$. Produce de salida una firma digital $(\sum)$.

4.- **Algoritmo de verificación de firma:** toma como entrada los parámetros de dominio $D$, una clave pública $Q$, un mensaje $m$, una firma digital $(\sum)$ y como salida, aceptará o rechazará dicha firma.

Hay que asumir que el dominio de parámetro $D$ es un dominio válido y que la clave pública $Q$ sea también válida y asociada con $D$.

Existen algunos esquemas de firmas digitales previamente estandarizados, como es el caso del **Algoritmo de Firma Digital de curva elíptica** (ECDSA, *Elliptic Curve Digital Signature Algorithm*) [Johnson01], el **Algoritmo de Firma Digital basado en el Certificado Análogo a la curva elíptica** (EC-KCDSA, *Elliptic Curve analogue of the Korean Certificate-based Digital Signature Algorithm*) [Lim98].

En este apartado nos enfocaremos únicamente en explicar el proceso para ECDSA. Debido a que el algoritmo de firma digital de curva elíptica es uno de los esquemas que por excelencia se ha estandarizado más ampliamente. Las organizaciones que han acreditado oficialmente dicho esquema son: ANSI X9.62, FIPS 186-2, IEEE 1363-2000, ISO 1488-3 y ISO/IEC 15946-2.

### 3.7.1. Algoritmo de firma digital de curva elíptica (ECDSA)

El Algoritmo de Firma Digital de curva elíptica (ECDSA) [Johnson01] es el análogo del Algoritmo de Firma Digital (DSA, *Digital Signature Algorithm*). Es decir, conceptualmente ECDSA es obtenido a partir de DSA. Dicho de otra manera, la única diferencia significativa que se encuentra entre DSA y ECDSA es la forma en que se obtiene la clave privada.

En el caso de DSA, toma el elemento aleatorio $x = g^k$ mód $p$, con ello se obtiene un entero en el intervalo $[1, q-1]$. Por su parte ECDSA, de manera muy general, genera

la clave privada en el intervalo $[1, n-1]$ tomando a la $x$ del punto aleatorio, reduciéndolo módulo $n$.

El algoritmo de firma digital de curva elíptica (véase el algoritmo 3.11), consiste de cuatro fases [Hankerson04]:

1.- Resumen del mensaje.

2.- Cálculos sobre la curva elíptica.

3.- Cálculos modulares.

4.- Conversión de enteros a cadena de bytes.

**Algoritmo 3.11:** Proceso para genera una firma con ECDSA.

**Entrada**: Una cadena de bytes $M$. Los parámetros del dominio $D$ (el entero que se encuentra en el rango $[0, n-1]$ es representado mediante una cadena binaria de longitud $h = \lceil \log_2 h \rceil$ bits o equivalentemente por una cadena de $f = \lceil \frac{h}{8} \rceil$ bytes de longitud). La clave privada $d$. Una función hash criptográfica $H$.

**Salida** : Retornar una cadena de bytes $SI$ (de longitud $2f$) que representa a la firma de $M$.

**Inicio**

*/* Resumen del mensaje. */*

Convertir $M$ en una cadena binaria $M'$.

Calcular el valor hash: $e = H(M')$ (si $H$ es el algoritmo SHA-1 [180-193], entonces el valor de hash $e$, es una cadena binaria de 160 bits de longitud).

*/* Cálculos sobre la curva elíptica. */*

Seleccionar un entero aleatorio, tal que, $1 \leq k \leq n-1$.

Calcular $kP = (x_1, y_1)$.

Convertir $x_1$ ($x_1$ es una cadena de bits de longitud $t$) a un entero $\overline{x_1}$.

Calcular $r = \overline{x_1}$ mód $n$.

*/* Cálculos modulares. */*

Calcular $s = k^{-1}(\overline{e} + rd)$ mód $n$.

**si** $s == 0$ **entonces**

  ir a el paso 3.

*/* Conversión de un entero a cadena de bytes. */*

Convertir $r$ a una cadena binaria de longitud $h$.

Añadir una cadena binaria de ceros de longitud $8f - h$ a la izquierda.

Convertir la cadena binaria resultante de $8f$ bits en una cadena de bytes, $R$, de longitud $f$.

Convertir $s$ a una cadena binaria de longitud $h$.

Añadir una cadena binaria de ceros de longitud $8f - h$ a su izquierda.

Convertir la cadena binaria resultante de $8f$ bits en una cadena de bytes, $S$, de longitud $f$.

Retornar la cadena de bytes:

$$SI = R \| S.$$

**Fin**

A continuación, lo que queda por hacer es la verificación de la firma producida por el algoritmo 3.11.

El proceso de verificación de la firma digital de curva elíptica (véase el algoritmo 3.12). Consiste de cuatro etapas:

1.- Conversión de la cadena de bytes a un entero.

2.- Resumen del mensaje.

3.- Cálculos sobre la curva elíptica.

4.- Verificación.

**Algoritmo 3.12**: Verificación de la firma con ECDSA.

**Entrada**: Una cadena de bytes $M$ que corresponde a el mensaje. Una cadena de bytes $SI$, que es la firma de $M$. Los parámetros del dominio $D$. La clave pública $Q$. Una función hash criptográfica $H$.

**Salida** : Aceptar o rechazar la firma.

**Inicio**

*/* Conversión de cadena de bytes a un entero. */*

Hacer un análisis de $SI$, con lo que se obtiene dos cadenas de bytes, $R$ y $S$ de longitud $f$.

Convertir $R$ a una cadena binaria de longitud $8f$.

Convertir el resultado del paso anterior a un entero $r$.

Convertir $S$ a una cadena binaria de longitud de $8f$.

Convertir el resultado del paso anterior a un entero $s$.

*/* Resumen del mensaje. */*

Convertir $M$ a una cadena binaria $M'$.

Calcular el valor hash: $e = H(M')$. **si** *$H$ se trata del algoritmo SHA-1* **entonces**

    es el valor de hash, $e$, es una cadena binaria de 160 bits de longitud.

*/* Cálculos sobre la curva elíptica. */*

Calcular $w = s^{-1}$ mód $n$. Calcular $u = w\overline{e}$ mód $n$.

Calcular $v = wr$ mód $n$.

Calcular el punto $uP + vQ$ de la curva elíptica.

*/* Verificación. */*

Convertir la cadena binaria $x_1$ de longitud $t$ a un entero $\overline{x_1}$.

Hacer $r' = \overline{x_1}$ mód $n$

**si** $r == r'$ **entonces**

    Se acepta la firma.

**Fin**

Para ilustrar de la mejor manera el proceso de la verificación de una firma digital, el ejemplo 3.2 muestra el desarrollo del algoritmo con los datos del ejemplo 3.1.

**Ejemplo 3.2** *Firma digital ECDSA* Tomando en cuenta los valores de entrada del ejemplo 3.1. Sean A una entidad que desea firmar el siguiente mensaje $M = 11100011010111100$, y considerando el valor hash de $M$ como $H(M) = e = 7$. La entidad A hace lo siguiente:

1.- Asúmase que selección aleatoria de $k$ en el intervalo $[1, 4]$ da un valor $k = 2$.

2.- Calcula el punto

$$\begin{aligned} 2P &= (x_1, y_1) \\ &= (\alpha, \alpha). \end{aligned}$$

3.- Representa $x_1 = \alpha = (1100)$ como el entero $8 + 4 = 12$.

4.- Hacer

$$\begin{aligned} r &= x_1 \text{ mód } n \\ &= 12 \text{ mód } 5 \\ &= 2. \end{aligned}$$

5.- Calcular

$$\begin{aligned} s &= k^{-1}(e + rd) \text{ mód } n \\ &= 3 \cdot (7 + 2 \cdot 3) \text{ mód } 5 \\ &= 4. \end{aligned}$$

6.- La firma digital del mensaje $M$ es $(r, s) = (2, 4)$.

La entidad B por su parte verifica la firma $(r, s)$ de la siguiente manera que a continuación se muestra:

1.- Obtiene la clave pública de A, tal que: $Q = (x_Q, y_Q) = (\alpha, 0)$.

2.- Calcula la función hash $H(M) = e = 7$.

3.- Calcula $w = s^{-1}$ mód $n$ y con esto calcula $u = w\overline{e}$ mód $n = 3$ y $v = wr$ mód $n = 3$

4.- Calcula el punto

$$\begin{aligned}(x_1, y_1) &= uP + vQ \\ &= 3P + 3Q \\ &= (\alpha, 0) + (\alpha^5, \alpha^3) \\ &= (\alpha, \alpha).\end{aligned}$$

5.- Representa $x_1 = \alpha = (1100)$ como entero $8 + 4 = 12$.

6.- Calcula $r' = (x_1 \text{ mód } n) = 2$.

7.- Se acepta la firma ya que $r' == r$, esto es $2 == 2$.

## 3.8. Protocolo de acuerdo de clave

Sea $A$ y $B$ dos entidades, las cuales desean establecer una clave única entre ellas. Asúmase que el domino de parámetros es el mismo tanto para una entidad, como para la otra. Sea $(Q_A, d_A)$ la clave pública y privada de la entidad $A$. $Q_B, d_B$ la clave pública y privada de la entidad $B$, respectivamente. Entonces haciendo uso del algoritmo 3.13, se puede obtener una clave que pueden compartir tanto la entidad $A$, como la entidad $B$.

**Algoritmo 3.13**: Protocolo de acuerdo de clave

**Entrada**: $n$

**Salida** : $K$

**Inicio**

*/* La entidad $A$ realiza lo siguiente: */*

Seleccionar un entero aleatorio $k_A$, $1 \leq k_A \leq n-1$.

Calcular el punto $R_A \leftarrow k_A P$, tal que $R_A = (x_{R_A}, y_{R_A})$.

Se le envía $R_A$ a la entidad $B$.

*/* Por su parte la entidad $B$ lleva a cabo el siguiente proceso: */*

Selecciona un entero aleatorio $k_B$, $1 \leq k_B \leq n-1$.

Calcular el punto $R_B \leftarrow k_B P$, tal que $R_B = (x_{R_B}, y_{R_B})$.

Le envía $R_B$ a la entidad $A$.

*/* Como $A$ tiene $R_B$, hace: */*

Calcula el entero $s_A \leftarrow k_A + x_{R_B} d_A x_{Q_A}$ mód $n$.

Se calcula la clave $K \leftarrow s_A(R_B + x_{R_B} x_{Q_B} Q_B)$.

*/* Por su parte la entidad $B$ lleva a cabo: */*

Calcula el entero $s_B \leftarrow k_B + x_{R_A} d_B x_{Q_B}$ mód $n$.

Se calcula la clave $K \leftarrow s_B(R_A + x_{R_A} x_{Q_A} Q_A)$.

**Fin**

## 3.9. Resumen

En este capítulo se explicaron algunos algoritmos usando curvas elípticas, desde el punto de vista criptográfico se describió el dominio de parámetros, el cual consiste en elegir la curva apropiada, sobre un campo primo y un punto sobre la curva. Para generar una curva elíptica y que está sea verificable se consideran dos métodos, cuando la curva éste sobre el campo $\mathbb{F}_p$ y cuando éste sobre el campo $\mathbb{F}_{2^m}$.

También, se vieron algoritmos para la generación de claves (pública y privada), el cifrado y para la firma digital sobre curvas elípticas. Finalmente, se mostró el algoritmo 3.13 que permite generar un clave que puede ser compartida entre dos entidades, la cual se utiliza para la autenticidad o para tener mayor privacidad.

Capítulo 4

# Implementación del protocolo de comunicación

En el capítulo 2 se explicaron conceptos generales de la teoría de curvas elípticas. Esos conceptos permitieron que en el capítulo 3 se desarrollara una variedad de algoritmos para distintos usos criptográficos.

En este capítulo se explica el desarrollo de un protocolo de comunicación que sirve para autentificar dos entidades, utilizando ECC. La implementación del protocolo, se basó en el algoritmo 3.13 visto en el capítulo 3 y se uso el lenguaje de programación orientado a objetos $C++$.

El diseño de algoritmos es una de las etapas del análisis de sistemas, que da una visión clara de la secuencia que se tiene que seguir para obtener el resultado esperado. También los algoritmos, dan la posibilidad que se puedan implementar en cualquier lenguaje de programación, ya que estos muestran la solución de los problemas de manera generalizada.

La elección de un lenguaje de programación, depende del rendimiento que se desee obtener de la aplicación. Es decir, el rendimiento es medido en no exceder los recursos físicos de los dispositivos donde la aplicación va a residir. Uno de los lenguajes de aplicación de alto nivel que ofrece un buen rendimiento es el lenguaje $C$.

El lenguaje $C$, da la posibilidad de emplearlo de manera modular, es decir, usando el paradigma de programación imperativo, pero también dispone del paradigma orientado a objetos (POO), donde es conocido como $C++$.

## 4.1. Introducción

Para el desarrollo del protocolo se hizo uso de la biblioteca desarrollada por el Dr. Juan Manuel García García[1] que fue creada para fines didácticos, donde se puede llevar a cabo operaciones aritméticas de puntos sobre la curva elíptica para fines criptográficos. A está biblioteca se le conoce como ECC y está implementada en $C++$. Actualmente, solamente se pueden usar curvas de Koblitz.

Los valores que se generan para las claves, son a su vez valores enteros muy largos. Por esa razón la biblioteca ECC hace uso de la biblioteca NTL para manipular dichos números. NTL fue creado por Víctor Shoup[2] usando $C++$. Está biblioteca permite usar estructuras de datos y clases para manipular enteros que tengan una longitud grande, así como también manejar polinomios de enteros y de precisión punto flotante arbitrarios. Si se desea obtener un poco más de información sobre NTL, se recomienda ver el apéndice C.

El protocolo de comunicación de acuerdo de clave utilizando ECC, se puede representar de manera generalizada como un conjunto de siete módulos que se relacionan entre si. Estos siete módulos son: **NTL**, **ec_point**, **eccMet**, **packet_struct**, **Packet**, **Socket**, **ServerException** y **ProtocolKeyAgreement**.

La relación entre estos siete módulos se puede observar en la figura 4.1. Las flechas indican la interacción entre los módulos. El grupo de módulos que se encuentran dentro del rectángulo con línea punteada, reciben los valores (está acción se puede apreciar con las flechas más anchas) de la curva que están almacenados; este grupo de módulos devuelve las claves generadas.

Como NTL y ECC se han desarrollado bajo $C++$, es necesario aprovechar dicho paradigma para implementar el protocolo de comunicación.

Para apreciar claramente cómo quedan integradas todas las clases y paquetes, se presenta la figura 4.2, donde se muestra la asociación y ligaduras de las clases, así como la multiplicidad que cada clase tiene con la que se encuentra asociada. Este modelado se realizó empleando el lenguaje UML (por sus siglas en inglés, *"Unified Modeling Language"*).

[1] http://www.criptored.upm.es/software/sw_m128_02.htm
[2] http://www.shoup.net/

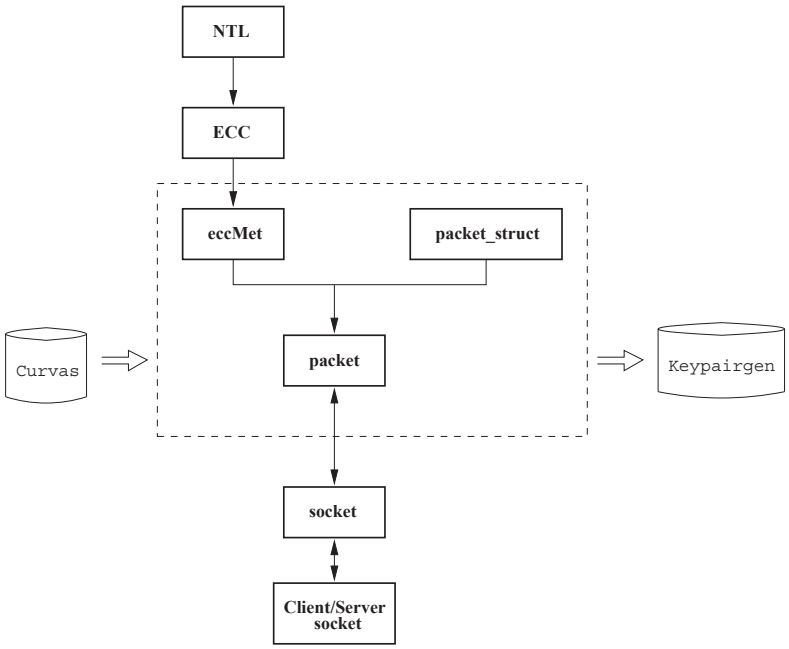



Figura 4.1: Módulos que conforman el protocolo de comunicación.

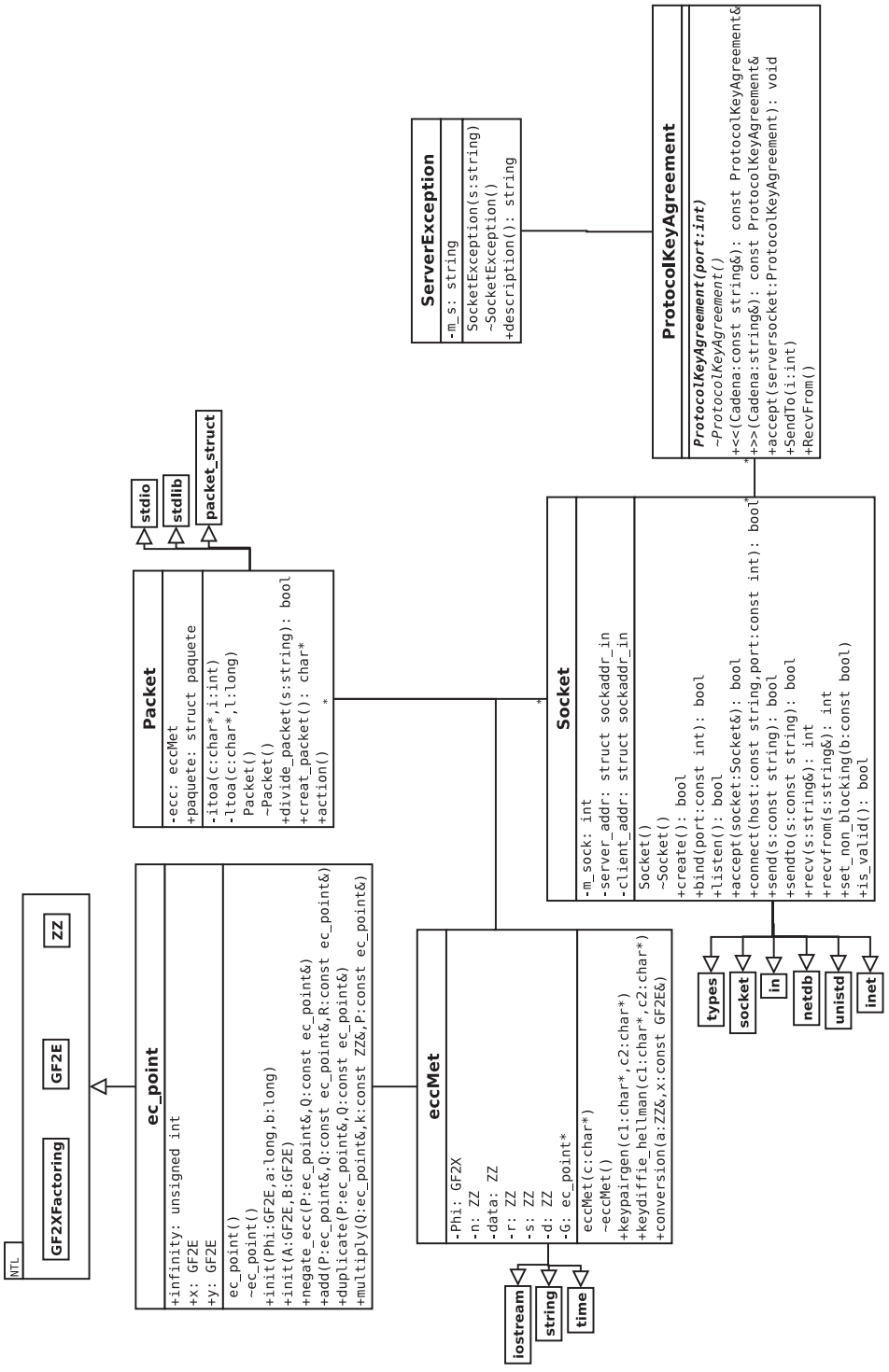



Figura 4.2: Modelado de las clases del protocolo con UML

Por lo tanto, esté capítulo está organizado de la siguiente forma. Se explican los aspectos más importantes de la clase ECC (en el diagrama 4.2, se le conoce como *ec_point*). Siguiendo, ya de lleno, con la implementación del protocolo. Se van considerando los puntos que se creen más sobresalientes de cada una de las clases y las operaciones que se hacen. Por último, se muestra la implementación de la clase *eccMet*, que es donde se encuentran los métodos para llevar a cabo las operaciones de creación de la clave pública y privada, y la clave compartida.

## 4.2. Biblioteca ECC

La biblioteca de criptografía de curvas elípticas que se uso para el desarrollo de este proyecto, fue implementada por el Dr. Juan Manuel García García[3]. Es una implementación que se baso en el paradigma de programación orientado a objetos, para la aritmética de puntos sobre curvas elípticas, para ser utilizado en aplicaciones criptográficas. Para esta biblioteca se empleo el lenguaje de programación C++ para su desarrollo, originalmente se uso el entorno de desarrollo MinGW[4] bajo Windows de Microsoft, y en un momento posterior se probó bajo Linux usando el compilador g++ de GNU[5].

Básicamente esta biblioteca implementa la clase llamada *ec_point()*, que representa a un punto de la curva elíptica definida sobre un campo característico 2. Para definir estos tipos de campos se usan las clases GF2E, GF2XFactoring y la clase ZZ de la biblioteca NTL (ver apéndice C).

Entre los métodos que se definen en esta clase, se encuentran la negación (*negate()*), la suma de puntos (*add()*), la duplicación de un punto (*duplicate()*) y la multiplicación de un punto por un valor escalar (*multiply()*).

Como dato extra, esta biblioteca incluye además, dos implementaciones criptográficas, que son: el esquema de cifrado sobre curvas elípticas (ECES) y el algoritmo de firma digital sobre curvas elípticas (ECDSA). El resto de la sección se mencionará los métodos implementados para las operaciones aritméticas con curvas elípticas.

[3]Esta biblioteca se puede localizar en la siguiente dirección URL=http://www.criptored.upm.es/software-/sw_m128_02.htm

[4]http://www.mingw.org/

[5]http://gcc.gnu.org/

### 4.2.1. Operaciones aritméticas de la biblioteca ECC

#### Duplicar un punto

Para duplicar un punto sobre la curva elíptica se emplea el método *duplicate()*, el cual recibe como entrada dos parámetros, dicho método se define como sigue:

```
void duplicate(ec_point& P, const ec_point& Q);
```

Se puede apreciar en la declaración anterior que los valores de entrada del método deben de ser de tipo *ec_point*. El segundo parámetro ($Q$) contiene los valores $(x, y)$ que se desea duplicar, y el resultado de esa operación se asigna al primer parámetro ($P$).

**Ejemplo 4.1** *Duplicar un punto con ECC.* Considérese los valores de la curva de Koblitz de 163 como entrada, propuesta por NIST, los cuales son:

- Polinomio irreducible:

$$x^{163} + x^7 + x^6 + x^3 + 1$$

.

- Valores de los parámetros de la curva elíptica:

$$a = 0x1$$

y

$$b = 0x1$$

- Valores $x$ y $y$ del punto generador $G$:

$$\begin{aligned} G_x &= 0x8eee49c5e5d6e4ed397d70aaca11cbb7350c31ef2 \\ G_y &= 0x9d3aadcc835d635008e2f12385ff83d50bf070982 \end{aligned}$$

- Orden del punto generador:

$$n = 5846006549323611672814741753598448348329118574063$$

Por lo tanto al efectuar la duplicación de un punto (el valor de $G$ pasa como segundo el parámetro, $Q = G$), $P = 2Q$, su resultado sería el siguiente:

$$\begin{aligned} P_x &= 0xbe14c5a8d828b77a24b00bfcaa003ef8372ac5bc \\ P_y &= 0xb6afefe515466696a3af5d3dca09f58ba9e97c922 \end{aligned}$$

Todos aquellos valores que inicien con $0x$ representan valores numéricos hexadecimales, ya que de esa forma se pueden mostrar numeros muy grandes.

### Suma de puntos

La suma de puntos es un método que recibe como entrada tres parámetros de tipo *ec_point* y no retorna valor alguno. Este se declara así:

```
void add(ec_point& P, const ec_point& Q, const ec_point& R);
```

El segundo parámetro ($Q$) y el tercero ($R$) son los puntos que se desean sumar y el primer parámetro ($P$) es donde se coloca el resultado de dicha operación.

**Ejemplo 4.2** *Suma de puntos con ECC.* Considerando los valores entrada del ejemplo 4.1. El resultado de dicho ejemplo sería el valor que se pasa como segundo parámetro ($Q$) y el punto generador es el tercer parámetro de entrada ($R$), por lo tanto la respuesta de la operación $P = Q + R$ sería:

$$\begin{aligned} P_x &= 0x02596f02bd330028f4208282f3e8fa2a9ccfcfca2 \\ P_y &= 0x4efdce9014085e41fd71c4b7cdab519f74c9275 \end{aligned}$$

### Negación de un punto

Este método recibe dos parámetros como entrada y no retorna valor alguno. Así, se define dicho método:

```
void negate(ec_point &P, const ec_point &Q)
```

El segundo parámetro ($Q$) es el punto que se desea negar y debe de ser de tipo *ec_point*. El resultado de dicha operación se coloca en el primer parámetro ($P$) del método que debe de ser también de tipo *ec_point*.

**Ejemplo 4.3** *Negación de un punto.* Se negará el valor del punto generador definido en el ejemplo 4.1 $P = -G$, entonces el valor de $G$ será representado por el parámetro de entrada $Q$ y el resultado aparecerá en el parámetro $P$. Por lo tanto, el resultado de dicha operación será:

$$\begin{aligned} P_x &= 0x8eee49c5e5d6e4ed397d70aaca11cbb7350c31ef2 \\ P_y &= 0x13d4e409668b87bd319f81894fee48623efc4177 \end{aligned}$$

**Multiplicación de un punto por un escalar**

En este método se implementa la operación de multiplicar un valor escalar por un punto. Dicho método recibe tres parámetros de entrada que son el punto donde se asignará el resultado ($Q$); que debe de ser de tipo *ec_point* , el valor del escalar ($k$); que es de tipo *ZZ* y el punto ($P$); por el que se va a multiplicar el valor escalar. La definición de este método se muestra a continuación:

```
void multiply(ec_point &Q, const ZZ &k, const ec_point &P);
```

**Ejemplo 4.4** *Multiplicación de un punto por un escalar.* Dado los valores del ejemplo 4.1 y sea el valor de $k = 357176711731246010345816404725289908197533412702 7$ y el valor de $P$ representado por el punto generador $G$. Entonces tendremos que la multiplicación de $kP$ sería:

$$\begin{aligned} Q_x &= 0xd5ec74bc28724746e4d71081c0386ceae03321c03 \\ Q_y &= 0xe0294448ad5e463ad98295c6bcf837b0dfabd79c7 \end{aligned}$$

## 4.3. Protocolo de Comunicación

La figura 4.1 representa a los módulos con una línea continua formando un rectángulo, las figuras cilíndricas representan a los datos que están almacenados, como el caso de los valores de las curvas elípticas, y los resultados que serán guardados de manera física. Las flechas sencillas (en uno o dos sentidos) representan la manera como van interactuando los módulos.

El módulo *NTL* se ve más a detalle en el apéndice C. Por su parte el módulo *Socket* se puede profundizar en el apéndice D. En cuanto al módulo llamado *ProtocolKeyAgreement* simplemente su función es hacer que el módulo *Socket* siga una secuencia de entrada/salida (por ejemplo: leer-escribir-leer) entre las partes.

Las clases que más interesan en está sección y donde se centra el desarrollo del protocolo, son el grupo de módulos que se encuentran encerrados en el rectángulo dibujado con línea punteada (ver figura 4.1), que son: *eccMet*, *packet_struct* y *packet*. En el resto de esta sección se explicará cada uno de los módulos mencionados.

### 4.3.1. Biblioteca packet_struct

Cualquier protocolo conocido, como TCP, IP, ICMP, etc., fueron diseñados como una estructura divida en varias partes, y cada parte tiene alguna finalidad. En nuestro caso no fue la excepción. En la figura 4.3 se puede apreciar la manera en que esta conformado el protocolo.

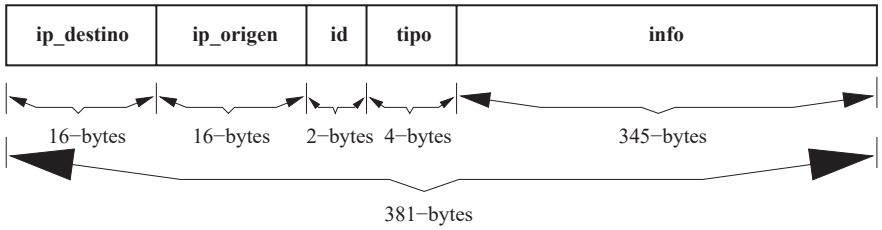



Figura 4.3: Representación gráfica del paquete.

Vamos a explicar cada una de las partes:

- **ip_destino**: Corresponde a la dirección IP del nodo destino.
- **ip_origen**: Campo donde se incluirá la dirección IP del nodo origen.
- **id**: En este campo se puede usar para llevar a cabo un control de secuencia de paquete[6].
- **tipo**: Lleva el número de proceso que deberá efectuar dicho nodo.
- **info**: En este campo, se incluirá el nombre de la curva elíptica que será usada, así como las claves públicas que generaron.

La declaración de estos campos se hace definiendo una estructura llamada *paquete* y quedaría como sigue:

```
struct paquete
{
 char  dest_addr[16]; /* Dirección IP Destino. */
 char  orig_addr[16]; /* Dirección IP Origen. */
 short id;            /* PID del proceso que se le asigno . */
 int   tipo;          /* Tipo de información incluida. */
 char  info[345];     /* Información incluida el paquete. */
};
```

Respecto al campo *tipo* se puede hacer uso de cualquiera de las siguientes opciones que se muestran en la tabla 4.1:

[6]para el propósito que se tiene, este campo siempre valdrá cero.

| Nombre tipo | Valor | Descripción |
|---|---|---|
| CURVE_REQ | 1 | Solicitud de la curva. |
| CURVE_ACK | 2 | Respuesta de solicitud. |
| GEN_KEY_DH | 3 | Generar clave compartida. |
| END | 4 | Finalizar acuerdo. |

Cuadro 4.1: Valores que puede asignarse al campo *tipo*.

### 4.3.2. Clase Packet

Este es uno de los módulos donde recae gran parte de la implementación del protocolo de comunicación. A grandes rasgos, este es el encargado de formar un paquete y extraer información del paquete, también de determinar que acciones se tienen que efectuar de acuerdo al valor que contenga el campo *tipo* en el paquete.

Una de las cosas importantes en la formación del paquete es utilizar un delimitador, el cual determina el límite entre dos campos. Se hace uso de dos caracteres que a partir de este momento se les llamará *caracteres especiales*, que son: | y #.

El primer carácter de uso especial (|) es el encargado de delimitar los campos del paquete y el segundo carácter especial (#) es usado para delimitar los datos que se incluirán en el campo *info*. En la figura 4.4 se muestra un paquete formado, en donde se puede observar el uso de estos caracteres.

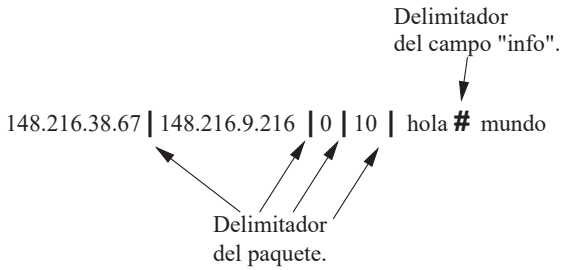



Figura 4.4: Representación gráfica del paquete con los caracteres especiales.

Así pues, estos caracteres se definen como sigue:

```
const char DIV_CHAR = '|';
const char DIV_CHAR_KEY = '#';
```

El módulo *packet* consta de tres métodos fundamentales, que son: la creación del paquete, la división (o separación) del paquete y la acción que se va a tomar.

### Creación del paquete

Este método consiste en la concatenación de los campos que conforman el paquete en una sola cadena de caracteres, junto con el respectivo carácter especial. El siguiente ejemplo ilustra la conformación de un paquete.

**Ejemplo 4.5** *Creación de paquete*. Supóngase que se tiene los datos que se muestran en la figura 4.4.

- **dest_addr** = 148.216.38.67
- **orig_addr** = 148.216.9.216
- **id** = 0
- **tipo** = 10
- **info** = hola#mundo

En seguida se declara una variable (*cad*) de tipo cadena de caracteres. Estando esta vacía se le va a ir asignando uno a uno los campos a la variable, es decir, se le asigna primero el campo *dest_addr* (**cad="148.216.38.67"**), inmediatamente después se le concatena el carácter de uso especial para el paquete (**cad="148.216.38.67|"**). Así se continua, concatenando los valores de los campos junto con el carácter especial hasta tener la cadena completa (**"148.216.38.67|148.216.9.216|0|10|hola#mundo"**)

La definición de este método es la siguiente:

```
char* creat_packet(void);
```

el cual no recibe de entrada ningún parámetro, pero devuelve un puntero de tipo *char*, que es la dirección donde se encuentra la cadena que se formó.

La implementación de la operación de crear el paquete se muestra a continuación:

```
char *dgram;

...

/* Agregar a "dgram" la IP_DST. */
strcat ( dgram, paquete.dest_addr );
```

```
dgram[strlen( dgram )] = DIV_CHAR;

/* Agregar a "dgram" IP_ORIG */
strcat ( dgram, paquete.orig_addr );
dgram[strlen( dgram )] = DIV_CHAR;

/*
 * El valor entero largo de PID se convierte
 * primero a (char *)
 * enseguida se pasa a la cadena "dgram"
 */
ltoa ( pid, paquete.pid );
strcat ( dgram, pid );
dgram[strlen( dgram )] = DIV_CHAR;

/*
 * El valor del entero type se convierte a tipo (char *)
 * enseguida se pasa a la cadena "dgram".
 */
itoa ( type, paquete.type );
strcat ( dgram, type );
dgram[strlen( dgram )] = DIV_CHAR;

/* Por último se agrega el contenido a INFO */
strcat ( dgram, paquete.info );
dgram[strlen( dgram )] = '\0';

return ( dgram );
```

### Dividir paquete

Este método realiza la operación de separar cada uno de los campos que vienen en la cadena. La definición de este método es como sigue:

```
bool divide_packet(string s);
```

Tiene como parámetro de entrada un tipo de dato *string* que es la cadena que representa al paquete. Esta cadena la recorre hasta encontrar un carácter igual a | y como se definió la forma del paquete se considera que el primer valor que se obtenga hasta antes del carácter especial corresponde a la dirección IP del nodo destino, el siguiente valor que se obtenga hasta antes del segundo carácter especial corresponderá a la dirección IP del nodo origen, y así sucesivamente hasta el largo de la cadena *s*.

Para implementar está operación, se uso la función para concatenar cadenas de caracteres, definidas en el lenguaje C++, como a continuación se muestra:

```
char id[10];
char type[5];

/*
 * Se convierte los valores id y type a los
 * respectivos tipos en los que esta defini-
```

```
 * dos paquete.id y paquete.inf
 */

strcpy ( paquete.des_addr, strtok ( s, "|") );
strcpy ( paquete.orig_addr, strtok (NULL, "|") );
strcpy ( id, strtok (NULL, "|") );
strcpy ( type, strok (NULL, "|") );
strcpy ( paquete.info, strok (NULL, "|") );

 ...
```

### Acción a tomar

La definición de este método es como sigue:

```
void action ( void );
```

No recibe ni retorna parámetro alguno, y consiste en determinar la operación que tiene que realizar el nodo. Este método esta muy ligado con el valor del campo *type*. Dependiendo del valor que tenga el campo *type* es la acción que se va a tomar. A continuación se muestra la implementación de este método:

```
void action(void)
{
 switch ( paquete.type )
 {

  .
  .
  .


  /*
   * Cuando llega solicitud de curva, se tiene que elegir el nombre
   * del archivo de la curva a usar.
   *
   * Del origen se recibe la petición y se procesa.
   *
   */

   case CURVE_REQ: /* Solicitud de Curva */
   {
     /* Cuando se entra a este caso, se debe a que
      * el origen solicitó un de acuerdo de claves
      * ("key aggrement"), llevando a cabo el si-
      * guiente proceso:
      *  1.- Se determina la curva que se va a emplear.
      *  2.- Se calcula la clave publica y privada, con la curva
      * que se va a compartir con el nodo que solicita ese acuerdo.
      *  3.- Al campo TYPE se le asigna el valor de CURVE_ACK.
      *  4.- Al campo INFO se le agrega el nombre de la curva que
      * se empleo para generar la claves y que deberá de usar el
      * otro nodo y además se le incluye el valor de la clave públi-
      * ca que se generó.
      */
```

```
    .
    .
    .
}//fin_case_CURVE_REQ
break;

case CURVE_ACK: /* Respuesta de Curva a usar */
{
  /*
   * En este caso, sucede lo siguiente, en el modulo de "INFO" se
   * encuentra, además de el nombre de la curva a usar, las partes
   * que conforman la clave pública del nodo que envió la respuesta
   * de la solicitud de la curva. Por lo que tiene el siguiente
   * proceso:
   *  1.- Separa las partes que conforman el modulo "INFO":
   * curve_name, key_foreing_pub_x, y key_foreing_pub_y.
   *  2.- Crear un archivo ("key<ip_envió>.pub") en donde se almace-
   * nará tanto: key_foreing_pub_x y key_foreing_pub_y.
   *  3.- Con el nombre de la curva ("curve_name"), se calcula
   * la clave publica y la privada. (keypairgen).
   *  4.- Generar clave compartida, con el valor de la clave
   * pública foránea (key_foreing_pub_<x,y>).
   *  5.- Asigna el valor de GEN_KEY_DH a TYPE.
   *  6.- Agrega el resultado de la clave pública que generó al
   * campo INFO.
   *
   */

   .
   .
   .


 }//fin_case_CURVE_ACK.
 break;

 case GEN_KEY_DH:/* Generar la clave compartida */
 {
   /*
    * Cuando entra en este caso, significa que el otro nodo
    * fue capaz de generar sus respectivas claves, así como
    * la clave compartida. Entonces queda de generar la lla-
    * ve compartida para este nodo, ya que viene en el campo
    * INFO la clave pública del otro nodo, pera ello se sigue
    * el siguiente procedimiento:
    *  1.- Separa las partes que conforman el modulo "INFO":
    * key_foreing_pub_<x,y>.
    *  2.- Crear un archivo ("key<ip_envió>.pub") en donde se almace-
    * nará tanto: key_pub_<x,y>.
    *  3.- Generar clave compartida, con clave publica forenea.
    *  4.- Agregar el valor de END a TYPE.
    *  NOTA: El campo INFO se queda vacío.
    *
    */

    .
    .
    .


 }//fin_case_GEN_KEY_DH.
```

```
    break;

    case END:/* Terminación de procesos */
    {
        cout << "\n\tFinalizó acuerdo de claves entre dos nodos.....!!!!";
    }//fin_END
    break;

    .
    .
    .
 }//fin_switch
}//fin_action
```

Vamos a suponer que existen dos nodos (A y B) que intentan ponerse de acuerdo en un clave para ser usada entre ellos. El nodo B forma un paquete con el valor del campo *type* igual al valor de la constante CURVE_REQ y se lo envía al nodo A. El nodo A recibe el paquete y revisa el valor de *type*. Como el valor es igual al de CURVE_REQ, lo que hace es seleccionar los valores de la curva (esta se encuentran en archivos[7]) que se va a emplear.

Inmediatamente después genera su clave pública y la privada. Enseguida forma un paquete donde le asigna el valor de CURVE_ACK al campo *type*, además el campo *info* va a contener el nombre de la curva que usó y los valores de la clave pública.

El paquete lo recibe el nodo B y verifica que el valor del campo *type* corresponde a CURVE_ACK, por lo que tomará los datos que vienen en el campo *info* para generar su clave pública y su clave privada; pero además puede hacer la operación de generar la clave compartida, ya que el campo *info* contiene también el valor de la clave pública del nodo A.

[7]Estos archivos contienen los valores de los parámetros que definió NIST, es decir, los valores de la curva de Koblitz de 163 se encuentran físicamente en un archivo con el nombre K-163 como sigue:

163
7
6
3
0
0x1
0x1
0x8eee49c5e5d6e4ed397d70aaca11cbb7350c31ef2
0x9d3aadcc835d635008e2f12385ff83d50bf070982
5846006549323611672814741753598448348329118574063

donde las primeras cinco líneas corresponden a los valores del polinomio irreducible ($x^163 + x^7 + x^6 + x^3 + 1$), las siguientes dos líneas son los valores de los parámetros de la ecuación de la curva elíptica (a y b, $y^2 + xy = x^3 + ax^2 + b$); la antepenúltima y penúltima línea corresponde a los valores $(x, y)$ del punto generador $(G)$; y la última línea es el valor del orden $(n)$ del punto generador.

Al concluir el nodo B de hacer esta operación, es necesario enviarle un paquete al nodo A para que éste forme su clave compartida. Para esto se le asigna el valor de GEN_KEY_DH al campo *type* y el resultado de su clave pública al campo *info*.

El nodo A recibe el paquete y al verificar que se trata del valor GEN_KEY_DH, lo que hace es generar la clave compartida con el valor de la clave pública que envió el nodo B en el campo *info*.

Finalmente, el nodo A envía un último paquete al nodo B con el valor 4, que corresponde a END, en el campo *type*. De esta manera termina la comunicación entre los dos nodos para el acuerdo de claves.

## 4.4. Clase eccMet

La finalidad de está clase es implementar métodos, para realizar operaciones tales como: generar la claves, tanto pública como privada, así como la clave compartida. Para instanciar un objeto con está clase, se le pasa un parámetro de tipo cadena de caracteres (*string*). De la siguiente forma queda el constructor:

```
eccMet::eccMet( char *filename )
```

Ese parámetro que recibe el constructor, representa al nombre del archivo de la curva que se quiere usar (p.e. K-163). Como se comentó en la sección anterior, cada línea del archivo contiene cierta información relevante acerca de la curva elíptica que se pretende usar.

Lo primero que se obtiene son los coeficientes del polinomio irreducible. Cada uno de esos valores, se van asignando al polinomio con la sentencia *SetCoeff(Phi, i)*.

```
eccMet::eccMet( char *filename )
{
  .
  .
  .
  long i;
  do {
  if ( ( buffer = ( char * )malloc( 5 * sizeof( char ) ) ) == NULL )
    cerr << "\nERROR: mal manejo de espacio de memoria.";

  fileread.getline ( buffer, 5 );
  cout << "\n" << buffer;
  i = atol ( buffer );
  SetCoeff(Phi,i);
  free ( buffer ); // Liberar espacio de memoria.
   } while(i);
   .
```

```
   .
   .
}//fin_constructor.
```

A continuación, se leen e inicializan los parámetros de la curva. También, se instancia un objeto ($G$) a la clase *ec_point*, que representará al punto generador de la curva, para obtener las coordenadas $x$ y $y$ del punto $G : (G_x, G_y)$. Por último, se lee el orden del punto generador ($n$). Esto se puede observar en el siguiente código que también se localiza en el constructor.

```
eccMet::eccMet( char *filename )
{
   .
   .
   .
   GF2E a, b; // Parámetros de la curva

   fileread >> a;
   fileread >> b;

   G = new ec_point();

   ec_point::init(a,b); // Inicializar la curva

   G->infinity = 0;

   fileread >> G->x;
   fileread >> G->y;

    fileread >> n; // Orden de G

}//fin_constructor.
```

El método *keypairgen* tiene como función generar las claves pública y privada. Como parámetros de entrada, se definen dos variables de tipo cadena de caracteres, que es donde se almacenarán las claves. Se declara una variable ($Q$), que es un objeto de la clase *ec_point*, la cual va ha representar a la clave pública. Después se inicializa el generador de números aleatorios y se selecciona un valor ($d$, esté representa a la clave privada) que debe encontrarse entre 0 y $n$. Finalmente, se calcula la clave pública ($Q = d * G$).

```
void eccMet::keypairgen( char *filename_pub, char *filename_priv )
{
   .
   .
   .

   ec_point Q; // Clave publica

   SetSeed(to_ZZ((long)time(NULL))); // Inicializar el
                                                                  // generador de
// números aleatorios.
```

```
   RandomBnd(d,n); // Seleccionar un número aleatorio d, 0< d < n

   multiply(Q,d,*G);   // Calcular Q=dG
   .
   .
   .
}
```

El último procedimiento, es el que genera la clave compartida. Recibe como parámetros dos cadenas de caracteres, *file_keypubforeing* y *file_keydiffiehellman*. La primera variable de entrada, es el nombre del archivo que tiene almacenado el valor de la clave pública, que fue enviada por la entidad vecina. El contenido es una cadena de caracteres que puede ser vista como un valor entero grande y es representado como una base hexadecimal. La segunda variable, servirá para almacenar la clave que se calcule, y que será la clave compartida.

En el método *keydiffie_hellman* se declaran dos objetos de tipo *ec_point*, donde el primero ($Qf$), va ha definir el valor de la clave pública de la entidad vecina y el segundo ($C$) se le asignará el resultado del cálculo de la clave compartida. Así, después de declarar dichas variables, se le asigna los valores a las coordenadas de $Qf$ ($Qf_x, Qf_y$). Finalmente, se multiplica la clave privada ($d$) y el valor de $Qf$, dando como resultado la clave compartida ($C$).

```
void eccMet::keydiffie_hellman( char *file_keypubforeing,
                                                char *file_keydiffiehellman )
{

   ifstream filereadkeypubforeing ( file_keypubforeing );

   if ( ! filereadkeypubforeing.is_open() )
   {
      cerr << "\nERROR: No se puede abrir el archivo.";
      exit ( 1 );
   }//fin_if

   ec_point Qf; // Llave pública foránea.

   ec_point C;  // Llave que es compartida
                     // por el que envió la clave foránea.

   filereadkeypubforeing >> Qf.x;
   filereadkeypubforeing >> Qf.y;

   multiply( C, d, Qf ); // Calculo de  C=d(Qf); k_a(k_bG)

   .
   .
   .
}//fin_keydeiffie_hellman
```

## 4.5. Resumen

En este capítulo se ha presentado la forma como está integrado el protocolo de comunicación, así como la implementación de este. Antes de entrar de lleno en la implementación, se presenta el diseño del modelo utilizando UML. Se optó por hacer esto, ya que UML permite tener un panorama más claro de las distintas clases, con sus métodos y variables, y como es que pueden interactuar. En su momento se fueron dando explicaciones breves pero concisas de cada una de las clases.

La primera que se analizó fue, *ec_point*, que muestra los métodos que permiten realizar las operaciones aritméticas sobre puntos en la curva. En seguida, se describió como se crea el protocolo y el procedimiento para extraer los distinto campos de el. También se menciona un método que realiza las acciones de acuerdo al tipo de información que se envía de un nodo a otro.

Se expuso la clase que permite generar la clave pública y privada, donde se pudo observar el uso de las operaciones aritméticas sobre puntos en la curva elíptica. Finalmente, se expuso un método que permite generar la clave compartida para los nodos que intervienen en dicha acción.

Capítulo 5

# Implementación del protocolo en el dispositivo móvil

En el capítulo 4 se vio la manera como se encuentra organizado el protocolo de comunicación. Se hace mención de las bibliotecas más importantes, así como aquellos métodos usados para el desarrollo del proyecto.

Para todo organización o persona que diseñe aplicaciones, un paso muy importante y de gran transcendencia es evaluarlos en ambientes reales. Se pueden pasar toda la vida haciendo simulaciones, pero no hay como verlo funcionar en situaciones verdaderas. Allí es donde verdaderamente se puede determinar si funciona o no. No se quiere decir con esto que las simulaciones no sean importantes, pero hay que reconocer que los resultados que arrojan, son aproximaciones de entornos que se asemejan a la realidad, más no son la realidad.

Desarrollado el protocolo de comunicación, el siguiente paso fue exportarlo a un dispositivo móvil, cuyos recursos computacionales están por debajo de los sistemas de cómputo de escritorio (PC) actuales.

En este capítulo se explicará el proceso que se debió de realizar para tener en las mejores condiciones el equipo, que servirá para llevar a cabo las pruebas del protocolo de comunicación. En una primera parte se explicará el procedimiento que se debe de seguir para preparar la PC de escritorio. En seguida, se mencionará la puesta a punto del dispositivo móvil y por último se darán los tiempos que se obtuvieron al generar claves de distintas longitudes usando ECC, comparándolas con los tiempos que se obtuvieron al generar claves con RSA.

## 5.1. Configuración de la PC de escritorio

El equipo que se usó para el desarrollo del protocolo se encuentra en la familia de las computadoras personales de escritorio, con las siguientes características:

- **Marca**: HP Compaq.
- **Modelo**: d530 CMT.
- **Procesador**: Pentium IV con una velocidad de 2.6 GHz (HT).
- **Memoria**: 1 GB (RAM).
- **Disco duro**: 80 GB.

Esta PC cuenta con el sistema operativo **GNU/Linux i686**, con la versión 3,1 estable de Debian y el kernel $2{,}4{,}27-2-686$.

### 5.1.1. Necesidad de una PC de escritorio

La gran mayoría de las personas se han de preguntar, ¿Por qué se tiene que utilizar una PC de escritorio para un proyecto donde el equipo central va a ser uno con características completamente diferentes? La razón es muy simple y es que la gran mayoría de los dispositivos móviles, cuentan con capacidades limitadas que hacen la tarea del desarrollador más difícil.

Imagínese que se tiene que escribir más de mil líneas de código y se debe de hacer en un dispositivo que tiene como medio de escritura un teclado virtual (que se muestra en la pantalla del dispositivo), donde las teclas tienen una dimensión de a lo mucho $0{,}3 \times 0{,}3$ *cm* y como único medio de interactuar con ese teclado es el lápiz óptico, que sirve para pinchar el carácter que se desee. Resultaría sumamente extenuante llevar a cabo dicha tarea, además de otras tantas deficiencias físicas con las que hoy por hoy son muy notables en estos tipos de dispositivos. En cambio, para cualquier desarrollador, tener la libertad en las manos para la escritura y una visión clara de lo que se esta escribiendo es muy reconfortante y más para trabajos que requieren de horas de dedicación.

Enseguida, se va ha describir el procedimiento para tener la PC lista. El procedimiento también va a ser útil para la programación de aplicaciones que habitarán en arquitecturas diferentes.

### 5.1.2. Configuración para la programación

De manera general, cualquier PC deberá tener previamente instalado alguna distribución de Linux (preferentemente alguno que soporte paquetes *deb*) y deberá de contar con un kernel igual o superior a la versión 2.4.x.

También se debe de prever que el sistema cuente con una versión actual del compilador del lenguaje C++, conocido como GCC del proyecto GNU/Linux.

#### Instalación del compilador multi-plataforma (*cross-compiler*)

Un *compilador multi-plataforma* es un herramienta que se encarga de compilar un programa para una arquitectura diferente de la que se está haciendo uso para la compilación, es decir, ocurre cuando un compilador produce ejecutables para otro tipo de sistema diferente. Este tipo de herramientas son muy útiles cuando se tiene un sistema, el cual no cuenta con las herramientas necesarias para hacer la compilación o por el hecho de que el sistema donde se piensa hacer la compilación es mucho más rápido ó tiene mayores recursos.

Cuando se habló de un sistema diferente, se refiere a un sistema de cómputo de propósito general, el cual está completamente encapsulado por el dispositivo que lo controla. A estos tipos de sistemas se les conoce como *sistemas incrustados* (del inglés *."embedded system"*).

Un sistema incrustado tiene requerimientos específicos y lleva a cabo tareas predefinidas, a diferencia de una computadora personal que esta diseñado para uso general. Un sistema incrustado es un dispositivo con hardware programable, es decir, cuenta de manera general con un *chip* programable que es la *"materia prima"* y está programado con aplicaciones particulares del sistema. Por su parte en las computadoras de propósito general el software se carga de manera externa; es decir, el hardware de estos sistemas no tienen un propósito particular si no que se diseñan para poder realizar casi cualquier tarea. Por ello, el software que se diseña para estos sistemas, se puede instalar o cargar de manera externa.

Así pues, un sistema incrustado es una combinación de hardware y software, facilitando la producción en masa y con variedad de aplicaciones. Como ejemplo de estos sistemas, se puede mencionar a los dispositivos tales como: reproductores de música portable, organizadores, teléfonos inteligentes, sistemas de control en tiempo real, etc.

Un punto muy importante para aquel que va hacer uso de un compilador multiplataforma es conocer la arquitectura del dispositivo destino.

Existe una gran variedad de arquitecturas que usan diseños incrustados tales como: ARM, MIPS, Coldfire/68k, PowerPC, X86, PIC, 8051, AtmelAVR, H8, SH, V850, FR-V, M32R, etc.. A menudo estos sistemas no cuentan con un sistema operativo o algunos traen sistema operativos incrustados (del inglés *.embedded operating system"*) conocidos como *sistemas operativos en tiempo real* (RTOS del inglés *Real Time Operating System"*) o puede ser que el desarrollador porte alguno a este tipo de sistema. En el apéndice G, se muestra la gran variedad de sistemas operativos incrustados que existen, tanto los que son de licencia libre o gratuitos, como aquellos que no lo son. Éstos están clasificados por el tipo de dispositivo, el nombre del sistema operativo y el fabricante.

Muy bien, ahora hay que proseguir con la herramienta de compilación (del inglés *Toolchain*). Lo primero que uno debe de hacer, es conocer el tipo de arquitectura a la que se pretende desarrollar. Las herramientas de compilación contienen las utilerías necesarias para hacer una compilación multi-plataforma a un programa y poder con esto, crear un archivo binario o ejecutable.

Existen muchas opciones para decidir la herramienta a utilizar, una de las más utilizadas es **GCC**[1], la cual tienen licencia de libre distribución, por lo que se puede obtener de manera gratuita. También se puede construir la herramienta de compilación[2]. Unas de las ventajas que tiene el construir su propia herramienta de compilación, es que se puede personalizar de acuerdo a las necesidades del desarrollador. Las manera más factible y rápida es hacer uso de las herramientas ya previamente construidas[3] conocidas como (*PreBuilt-Toolchains*). Existe una gran variedad de este tipo de herramientas para la arquitectura que se desee.

Para aquellos que son usuarios del sistema operativo Linux y en especial a los que usan la distribución Debian, existe un procedimiento para instalar un compilador multi-plataforma que se puede adecuar para la arquitectura deseada y que es precisamente el que a continuación explica. En este proyecto, se instaló el compilador de *C* para una arquitectura **ARM**.

[1] http://gcc.gnu.org/
[2] http://www.handhelds.org/moin/moin.cgi/BuildCrossToolchainHowto
[3] http://www.handhelds.org/moin/moin.cgi/PrebuiltToolchains

### Preparación

Como notación vamos a considerar lo siguiente: cuando se use # se debe de hacer las operaciones como sesión de superusuario (*root*), en el caso donde se use **$** deberá de ser como un usuario normal.

Primero que todo, se instalarán los paquetes **toolchain-source**.

```
# apt-get install toolchain-source toolchain-source-gdb
```

Este paquete contiene el código fuente de *gcc* y las utilerías *binutils*, actuales. El paquete **toolchain-source-gdb** es muy útil para cuando se requiere depurar los programas.

A continuación se instala el **autoconf2.13**

```
# apt-get install autoconf2.13
```

Para la construcción del compilador multi-plataforma, muchas de las veces se requiere de programas como el **fakeroot** o el también conocido **sudo**. En caso de que no se tenga alguno de estos instalado es recomendable hacerlo.

En versiones anteriores a la del kernel 2.4.20, será necesario cambiar la línea *crossbase=/usr/local* por la *crossbase=/usr* del archivo **/etc/dpkg/cross-compile**.

### Establecimiento de binutils, libc y el compilador

Se agrega el usuario actual al grupo **src**.

```
# adduser <usuario> src
```

**binutils** Se desempaqueta y construye las utilerías **binutils** y el **compilador**.

```
$ cd /usr/src/; tpkg-make arm-linux
```

Se agrega *–enable-languages=ç,c++"* al archivo *gcc-arm-linux-3.4.3/debian/rules*.

Ahora se hace lo siguiente.

```
$ cd binutils-arm-linux-2.15/; debuild -uc -us
```

con esto se configura, compila y construye *binutils*. Para construir el compilador, se tiene que hacer lo siguiente.

```
# debi
```

**libc** El compilador necesita las bibliotecas C del sistema para el que se pretende usar.

```
# tpkg-install-libc arm-linux
```

Este comando descarga las bibliotecas necesarias para una plataforma *arm-linux*, desde algún repositorio de paquetes Debian.

### Creando paquetes clonados

Para poder crear un paquete clonado, es necesario instalar el paquete *equivs*.

```
# apt-get install equivs
```

En seguida se usa el siguiente archivo de ejecución (*script*) y construye el paquete clonado.

```
#!/bin/sh

EMAIL='<Nombre completo> <usuario@localdomain>'
TARGET=arm-linux

# long options not yet implemented
args=`getopt 'e:t:' $*`
for o
    do case "$o" in
        -e | --emai*) EMAIL="$2"; shift; shift;;
        -t | --targ*) TARGET="$2"; shift; shift;;
        --) shift; break;;
    esac
done

if [ -z "$1" ]; then echo "no package"; exit -1; fi

PACKAGE=$1
VERSION=`dpkg-query -W --showformat='${Version}' $PACKAGE | sed 's/-.*/-1/'`

echo "Section: devel
Priority: optional
Standards-Version: 3.6.1.0

Package: $PACKAGE-$TARGET
Version: $VERSION
Maintainer: $EMAIL
Depends: $PACKAGE
Architecture: all
Description: Dummy $PACKAGE for $TARGET" > $PACKAGE-$TARGET.$$

equivs-build $PACKAGE-$TARGET.$$ && rm -rf ./equivs/ $PACKAGE-$TARGET.$$

$ ./<nombre-script>.sh -t arm-linux pkg-config
```

Se crea el siguiente archivo de control *pkg-config-arm-linux*,y se le agrega lo siguiente:

```
Section: devel
Priority: optional
Standards-Version: 3.0.1

Package: pkg-config-arm-cross
Version: 0.15.0-2
Maintainer: <Nombre completo> <email>
Depends: pkg-config
Architecture: all
Description: Dummy pkg-config for arm-linux

$ equivs-build pkg-config-arm-linux
```

Se instala el paquete formado, como sigue:

```
# dpkg -i pkg-config-arm-cross_0.15.0-2_all.deb
```

#### gcc

Para instalar **gcc** tenemos que hacer lo siguiente.

```
# apt-get install autoconf2.13 automake1.4 dejagnu libgmp3-dev
```

Se construye

```
$ cd ../gcc-arm-linux-3.4.3/; debuild -uc -us
```

y se instala el paquete con el comando **debi**:

```
# debi
```

Con esto tenemos un entorno de desarrollo multi-plataforma listo para usarse. Además, este mismo procedimiento se puede emplear para la arquitectura que se desee.

#### Una pequeña prueba

Vamos a probar el compilador multi-plataforma que se instaló. Se creará el clásico programa de *Hola Mundo*:

```
#include <iostream.h>

int main()
{
  cout << "Hola Mundo!!!";
  return 0;
}
```

para compilarlo, hay que hacer lo siguiente:

```
$ arm-linux-g++ -c hola_mundo hola_mundo.cpp
```

con esto se tendrá el archivo binario *./hola_mundo* y se puede comprobar con el comando **file** el tipo de archivo que se formó.

```
$ file hola_mundo
hola_mundo: ELF 32-bit LSB executable, ARM, version 1 (ARM),
for GNU/Linux 2.2.0, dynamically linked (uses shared libs),
not stripped.
```

### 5.1.3. Instalación de las bibliotecas

Para que la biblioteca ECC funcione adecuadamente se necesita instalar primero la biblioteca NTL.

### Instalación de NTL

La versión que se uso de NTL fue la 5,4. Después de descomprimir el archivo se configura y compila con el compilador nativo de **gcc**.

```
$ cd ntl_5.4/src; ./configure; make
```

Con lo anterior se crean varios archivos, de los cuales *mach_desc.h*, *gmp_aux.h* y *lip_gmp_aux_impl.h* se respaldan.

```
$ cp ../include/NTL/mach_desc.h OtraRuta/
$ cp ../include/NTL/gmp_aux.h OtraRuta/
$ cp lip_gmp_aux_impl.h OtraRuta/
```

Se borran los archivos creados con los siguiente comandos:

```
$ make clean
$ make clobber
```

Se edita el archivo **DoConfig** en las siguientes líneas de la manera que a continuación se muestra:

```
...
Wizard = off
...
CC=arm-linux-gcc
CXX=arm-linux-g++
AR=arm-linux-ar
RANLIB=arm-linux-ranlib
...
```

lo anterior, es para establecer el compilador multiplataforma se va a usar para compilar la biblioteca.

Los archivos que se respaldaron se tienen que copiar a su localidad original. En seguida se ejecuta el *script* **configure**.

```
$ ./configure
```

Después, se tiene que editar el archivo **makefile**, comentando las siguientes líneas:

```
setup1:
       ...
       #./MakeDesc
       ...
setup3:
       ...
       #./gen_lip_gmp_aux > lip_gmp_aux_impl.h
       ...
       #./gen_lip_gmp_aux > ../include/NTL/gmp_aux.h
       ...
```

Se ejecuta con el comando **make** para compilar la biblioteca e inmediatamente después como super usuario se instala.

```
# make install
```

### Instalación de ECC

Después de descargar y descomprimir el archivo, dentro de la carpeta *ecc-0.3/src* se localiza un archivo nombrado *makefile*, en donde se fija el compilador cruzado, editando las siguientes líneas.

```
CC=arm-linux-g++
...
AR=arm-linux-ar
RANLIB=arm-linux-ranlib
...
```

Al archivo *ecc-0.3/include/ecc.h* se le agrega el nombre de la macro que pertenece a la biblioteca de NTL: **NTL_CLIENT**.

Tanto al archivo *ecc-0.3/include/ecc.h* como a *ecc-0.3/src/ecc.c* se le cambia el nombre al método miembro que tiene como nombre *negate*, ya que en la versión 5,4 de NTL existe un método con el mismo nombre.

A continuación se compila

```
$ make all
```

e instala la biblioteca

```
# make install
```

## 5.1.4. Compilación de la aplicación del protocolo de comunicación

Para cada uno de los archivos **.cpp* se crea su módulo objeto de la manera que sigue:

```
arm-linux-g++  -O0 -g3 -Wall -c -fmessage-length=0 nombre_archivo.cpp
```

y cada uno de los archivos objetos (*.o) creados se enlazan tomando en cuenta las bibliotecas **lntl** y **lecc**. Además hay que destacar que el enlace se hace de manera estática, ya que se desea que los archivos binarios de las funciones se incorporen al código binario de nuestro ejecutable.

Todo esto se incluye dentro de un archivo llamado *makefile*:

```
makefile:

CC=arm-linux-g++
CFLAGS=-O2

AR=arm-linux-ar
ARFLAGS=ruv
RANLIB=arm-linux-ranlib
```

```
LDLIBS=-lm
NTL_LIBS=-lntl
ECC_LIBS=-lecc

PREFIX=/usr/local
LIBDIR=$(PREFIX)/lib
INCLUDEDIR=$(PREFIX)/include


INCLUDE=-I$(INCLUDEDIR)
LIB=-L$(LIBDIR)
COMPILE=$(CC) $(INCLUDE) $(CFLAGS) -c
LINK=$(CC) $(LIB) $(CFLAGS) -o

TEST_FLAGS= $(LDLIBS) $(NTL_LIBS) $(ECC_LIBS)


all: Prot-Com

ecc.o:  $(SRC)
        $(COMPILE) $(SRC)

ecc.d:  $(OBJ)
        $(AR) $(ARFLAGS) ecc.d $(OBJ)
        - $(RANLIB) ecc.d

eccMet.o: eccMet.cpp
        $(COMPILE) eccMet.cpp

eccMet.d: eccMet.o
        $(AR) $(ARFLAGS) eccMet.d eccMet.o
        - $(RANLIB) eccMet.d

Packet.o: Packet.cpp
        $(COMPILE) Packet.cpp

ServerSocket.o: ServerSocket.cpp
        $(COMPILE) ServerSocket.cpp

Socket.o: Socket.cpp
        $(COMPILE) Socket.cpp

simple_server_main.o: simple_server_main.cpp
        $(COMPILE) simple_server_main.cpp

Prot-Com: ecc.d eccMet.d DES.d Packet.o ServerSocket.o Socket.o
          simple_server_main.o
        $(LINK) Prot-Com  ecc.o eccMet.o DES.o Packet.o ServerSocket.o
        Socket.o simple_server_main.o $(TEST_FLAGS)

clean:
        - rm *.o
        - rm *.d
        - rm Prot-Com
```

## 5.2. Seleccionando la PDA

Existe una gran variedad de dispositivos móviles en el mercado, pero para seleccionar alguno se tuvo que tomar en cuenta dos aspectos

1.- Soporte para Linux.

2.- Costo.

En la actualidad las grandes corporaciones se han estado preocupando por crear equipos que tengan soporte para Linux y hay otros que sacan al mercado dispositivos con su propia distribución de Linux, entre los que podemos encontrar a los celulares con IP móvil, *gateways*, puntos de acceso (*access point*), dispositivos de audio/vídeo, PCs tipo *tablet*, etc.

De esta gran gama de equipos se ha elegido a los asistentes personales digitales (PDA, *Personal Digital Assistant*), debido a que son los dispositivos que más se asemejan a una PC de escritorio y cuentan con un gran soporte para Linux.

En el apéndice H se muestra una lista de las PDAs que cuentan con Linux precargado (desde fabrica) o con el soporte para ser instalado.

### 5.2.1. Dell Axim X5

Como se observa en la tabla H.1 del apéndice H, existe una gran variedad de PDAs que soportan Linux. Algunas ya lo traen integrado desde fabrica, como es el caso de la Sharp Zaurus.

La razón por la que se optó en seleccionar la PDA **Dell Axim X5**, fue que se tenia disponible.

Las características físicas con las que cuenta la PDA Dell Axim X5 (ver figura 5.1) son:

- **Dimensiones**: $128x81x18mm$.
- **Peso**: $196g$.
- **Velocidad del procesador**: $300MHz$.
- **Memoria**: $36MB$.
- **Slots de expansión**: Compact Flash y Secure Digital.
- **Pantalla**: 3,5", $320x240$pixels.

Figura 5.1: Dell Axim X5.

**Instalación de Linux sobre Dell Axim X5**

Como se mencionó anteriormente, la PDA Dell Axim X5 no tiene Linux de manera nativa y para esto se tiene que seguir el procedimiento que aquí se explica. Para poder realizarlo es necesario tener una memoria *Compact Flash* de al menos $64MB$ y un lector de memoria *Compact Flash*.

Todo el proceso se hará en una PC de escritorio con Linux. La versión de Windows que trae la PDA Axim X5 (Windows 2002 Mobile) se tiene que actualizar a Windows 2003 Mobile.

Se descargan al disco duro de la PC los siguientes archivos :

```
wince-loader-110404.tar.bZ2
zImage
axim-initrd-110404.tar.bZ2
axim-rootfs-110404.tar.bZ2
```

estos archivos se descomprimen con el comando: **tar xvfj**.

Con el paso anterior, se crea un directorio el cual cuenta con un archivo llamado *axim* que genera la imagen *initrd*. Para esto, es necesario ejecutar el archivo binario de la siguiente manera:

```
$ ./axim
```

Ahora se tiene que particionar la memoria Compact Flash. Se necesitarán dos particiones: una **FAT16** ó **FAT32** con un mínimo de $3MB$; y la otra de tipo **ext2** con el resto del espacio, pero con un mínimo de $40MB$.

Se relee la tabla de particiones de la memoria.

```
# hdparm -z /dev/sdx
```

y se usa el comando **fdisk** para realizar las particiones a la memoria. Teniendo las dos particiones, se necesitan crear el sistema de ficheros de cada una.

Para crear la partición **FAT16**

```
# mkdosfs -F16 /dev/sdx1
```

Para crear la partición **ext2**

```
# mke2fs /dev/sdx2
```

Ahora, se recomienda crear dos carpetas (*flash_fat16* y *flash_ext2*) en */mnt* donde se montarán cada una de las particiones creadas.

```
# mount /dev/sdx1 /mnt/flash_fat16
# mount /dev/sdx2 /mnt/flash_ext2
```

Se copia el archivo del kernel *zImage*, la imagen del cargador *initrd* y el cargador *winCe* a la partición **FAT16**:

```
# cp wince-loader/* initrd zImage /mnt/flash_fat16
```

Los archivos de lo que sería Linux se pasan a la partición **ext2**:

```
# cp -a axim-rootfs/ axim-rootfs/. /mnt/flash_ext2
```

Se desmonta la memoria y se coloca en la PDA Dell Axim X5. Para lanzar Linux, es necesario ubicar el archivo llamado **haret.exe** dentro de la memoria Compact Flash y ejecutarlo. Nos mostrará una ventana con el nombre del archivo script (*default.txt*) y se presiona el botón *Run* y de inmediato comienza la carga de Linux.

Teniendo preparado el dispositivo móvil (PDA) y habiendo compilado el programa, resta probar el programa y así determinar los tiempos que tardan en ser generadas las claves.

## 5.3. Resultados

Una vez que se tiene instalado y configurado Linux en la PDA, se compila el archivo en la PC y se pasa a la PDA. Al ejecutar el archivo binario, este funciona muy bien.

Decir que "funciona muy bien", puede no resultar una conclusión contundente para decidir si ECC es tan eficiente como se dice. Es por ello que se optó por computar lo

que tarda la PDA, en un ciclo de tiempo, para generar tanto la clave pública y privada, así como la clave compartida usando ECC.

En el primer experimento, se calcula el tiempo que toma en generar la clave pública y la privada entre la PC y la PDA usando ECC. La tabla 5.1 muestra los tiempos que arrojó la PC en generar las claves, usando ECC con curvas de Koblitz y las aleatorias sobre $\mathbb{F}_{2^m}$. Por su parte, en la tabla 5.2 se puede observar la cantidad que tuvo que invertir la PDA en generar la claves en un ciclo de tiempo. Examinando los tiempos de la tabla 5.1 y

| **Curva** | **Tiempo de generación de clave pública y privada (milisegundos).** | **Tiempo para generar clave compartida (milisegundos).** |
|---|---|---|
| K-163 | 0.005 | 0.005 |
| K-233 | 0.009 | 0.009 |
| K-283 | 0.014 | 0.014 |
| K-409 | 0.036 | 0.038 |
| K-571 | 0.083 | 0.081 |
| B-163 | 0.004 | 0.004 |
| B-233 | 0.009 | 0.009 |
| B-283 | 0.015 | 0.015 |
| B-409 | 0.04 | 0.039 |
| B-571 | 0.084 | 0.082 |

Cuadro 5.1: Tiempos en milisegundos para generar claves (pública, privada y compartida) con curvas de *Koblitz* y $F_{2^m}$ en 1 iteración, sobre una PC.

la tabla 5.2, se puede apreciar la deferencia de poder de cómputo entre la PC y la PDA. Es decir, si se utiliza una curva elíptica de Koblitz de 571 bits de longitud, la PC tarda un tiempo de 0,081 milisegundos en generar una clave compartida, en una iteración. En cambio la PDA, usando la misma curva, invierte 10,916 milisegundos más que la PC. Esto significa, que la PDA demora 11,0 milisegundos en formar la misma clave en una repetición (véase la figura 5.2(b) para mayor claridad).

En el segundo experimento, se examina el modelo criptográfico RSA[4], donde se evalúan los tiempos que tarda la PC, así como la PDA en generar una clave con longitudes que varían

[4]RSA es un sistema criptográfico, que es ampliamente usado en todo el mundo. Si desea conocer un poco más sobre éste modelo, véase el apéndice A

| Curva | Tiempo de generación de clave pública y privada (milisegundos). | Tiempo par generar clave compartida (milisegundos). |
|---|---|---|
| K-163 | 0.8 | 0.8 |
| K-233 | 1.7 | 1.6 |
| K-283 | 2.3 | 2.2 |
| K-409 | 5.1 | 4.9 |
| K-571 | 10.9 | 10.7 |
| B-163 | 0.8 | 0.8 |
| B-233 | 1.6 | 1.6 |
| B-283 | 2.2 | 2.2 |
| B-409 | 5.0 | 4.9 |
| B-571 | 11.0 | 11.0 |

Cuadro 5.2: Tiempos en milisegundos para generar claves (pública, privada y compartida) con curvas de *Koblitz* y $F_{2^m}$ en 1 iteración sobre una PDA Dell Axim X5.

desde los 1024 bits, hasta 15360 bits (véase la tabla 1.1, donde se muestra las equivalencias entre tamaños de claves, entre RSA y ECC). En éste experimento, también se consideró una iteración para cada equipo.

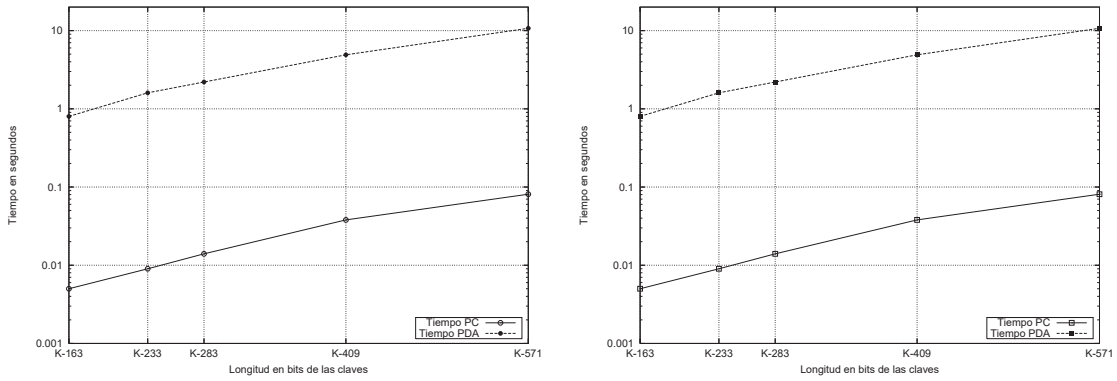
En la tabla 5.3 se puede apreciar los tiempos, en segundos, que la PC emplea para generar una clave. Como es de esperarse, el tiempo para crear una clave es proporcional a la longitud. Ahora bien, el tiempo que se tuvo que esperar para generar una clave, utilizando

| Longitud (bits) | Tiempo (segundos). |
|---|---|
| 1024 | 0.01 |
| 3072 | 0.39 |
| 7680 | 20.14 |
| 15360 | 185.00 |

Cuadro 5.3: Tiempos en segundos para generar claves con RSA en la PC en una iteración.

RSA, en la PDA fue más notorio. En la tabla 5.4, se puede observar que para formar una clave de 1024 bits se tardó 23 segundos, 22,99 segundos más que la PC. Pero donde se puede originar mayor asombro, es para establecer una clave de 15360, ya que la PDA, demoró 271873 segundos, que equivalen a 3 días aproximadamente.

Tomando en cuenta la tabla de equivalencias 1.1 del capítulo 1. Se puede comparar la eficiencia de ECC frente a RSA. Según la tabla 1.1 una clave de 163 bits de longitud, es




(a) Tiempo de generación de la clave pública y privada

(b) Tiempo de generación de clave compartida

Figura 5.2: Comparación de tiempos para generar la clave compartida entre la PC y la PDA, con diferentes longitudes.

| **Longitud (bits)** | **Tiempo de generación de clave (segundos).** |
|---|---|
| 1024 | 23 |
| 3072 | 1053 |
| 7680 | 95983 |
| 15360 | 271873 |

Cuadro 5.4: Tiempos en segundos para generar una clave con RSA sobre la PDA en una iteración.

equivalente a la de 1024 bits de RSA y así sucesivamente. Entonces, comparando los valores de la tabla 5.2, que muestra los tiempos sobre la PDA utilizando ECC y los de la tabla 5.4, se nota la enorme diferencia entre ECC y RSA.

En la figura 5.3 se muestra gráficamente, la diferencia entre RSA y ECC ejecutándose en la PDA. Véase la enorme diferencia, en tiempos y tamaños de las claves, entre los sistemas criptográficos ECC y RSA. Mientras que para generar una clave de 163-bits de longitud utilizando ECC, nos llevaría menos de un segundo, usando RSA nos demoraría un poco más de 20 segundos, con una longitud de clave de 1024-bits y con lo que obtendríamos el mismo nivel de seguridad.Ahora bien, observando las longitudes más grandes de los dos sistemas criptográficos (15360-bits para RSA y 571-bits para ECC), se aprecia que para generar una clave de 571-bits de longitud, usando ECC, tarda un poco más de 10 segundos,

en cambio para generar una clave de 15360-bits utilizando RSA, tenemos que esperar tres días y unas cuantas horas, solo para generar una clave de esa longitud.

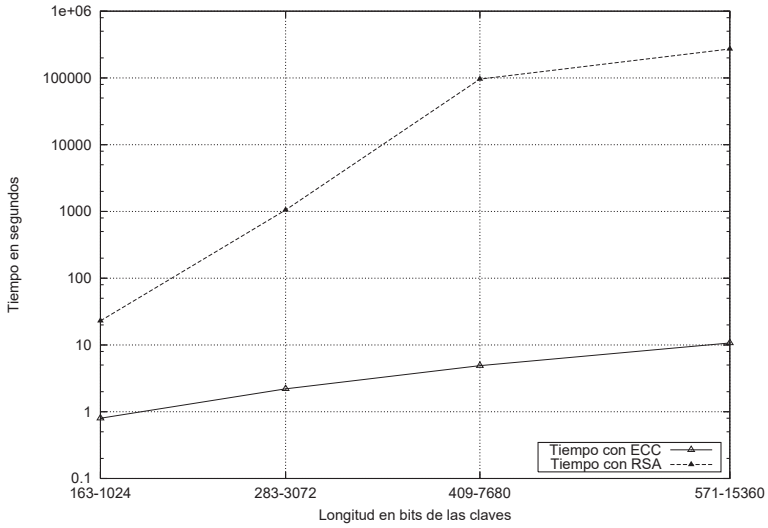



Figura 5.3: Comparación de tiempos de generación de claves sobre la PDA, usando ECC y RSA.

Aún comparando el tiempo en la PDA para generar una clave usando ECC y el tiempo que se lleva la PC en generar una clave utilizando RSA (ver figura 5.4), se nota una leve diferencia entre un sistema criptográfico y el otro.

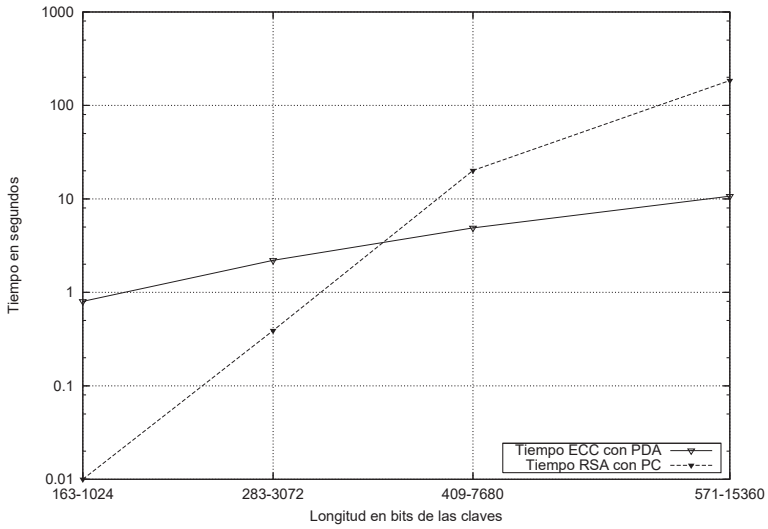



Figura 5.4: Comparación de tiempos de generación de claves sobre la PDA usando ECC y la PC utilizando RSA.

## 5.4. Resumen

Se ha explicado cada uno de los procedimientos para tener una PC lista, para compilar aplicaciones que van a ser probadas o residirán en arquitecturas con características diferentes. También, se discutió la razón por la que se eligió un dispositivo móvil, como la PDA Axim, para probar la aplicación y los pasos que se deben de seguir para instalar Linux. Finalmente, se muestran los resultados que se obtuvieron al generar claves de distintas longitudes, utilizando dos sistemas criptográficos: ECC y RSA, observando el gran rendimiento que tiene ECC respecto a RSA en la PDA.

Capítulo 6

# Conclusiones y trabajos futuros

En está tesis se ha mencionado algunos de los problemas que presentan las redes inalámbricas móviles ad hoc, así como los dispositivos que la integran a posibles ataques. El enfoque principal, fue el problema de recursos que presentan los dispositivos inalámbricos móviles respecto a la autenticación. En este capítulo se dan las conclusiones más relevantes y las sugerencias a trabajos futuros que se desprenden de está tesis.

## 6.1. Conclusiones

En este trabajo se mencionaron los problemas que tienen las redes inalámbricas móviles ad hoc, frente a posibles ataques. Así también se indicó los inconvenientes que presentan los dispositivos inalámbricos móviles, los cuales integran a dichas redes, utilizando modelos criptográficos tradicionales para la autenticación. Por lo que se desarrolló un protocolo de autenticación, basándose en ECC.

En el capítulo 2 se explicó, de manera breve, aspectos importantes acerca de la teoría de curvas elípticas. Describiendo lo que era un grupo abeliano. Luego lo que son los campos finitos, enfocándose principalmente en los campos finitos primos y los campos con característica dos. Con lo anterior es posible adentrarse en la definición de la curva elíptica, las características importantes que se tienen que considerar y las operaciones aritméticas que se pueden realizar.

Se presentó el diseño estructurado de un algoritmo para el protocolo de autenticación. En este, se toman en cuenta dos entidades que desean generar un clave que puedan compartir.

En está tesis, también se describe la manera como se tiene que preparar una PC para tener un compilador multiplataforma. Así como la instalación de una distribución de Linux sobre una PDA.

La parte experimental se presentó en el capítulo 6, de dichos experimentos se desprenden las siguientes conclusiones:

- Tanto la librería NTL como ECC, nunca habían sido probadas en un dispositivo que tuviera restricciones de recursos lo que permitió que con la integración de estas librerías al protocolo de comunicación se adaptarán para su buen funcionamiento.
- Se implementó un protocolo de comunicación de acuerdo de clave compartida, haciendo uso de la criptografía de curvas elípticas.
- Se demostró la eficiencia en la generación de claves del criptosistema de curvas elípticas, contra el ya tradicional método de cifrado-descifrado RSA.
- Se puede constatar como existe una gran diferencia entre una arquitectura sin tantas limitaciones de recursos, como es el caso de la PC de escritorio, y un dispositivo que cuenta con recursos más mínimos (PDA).

## 6.2. Sugerencias para trabajos futuros

Aquí se proporcionan algunas sugerencias para trabajos futuros que pueden extender el presente trabajo. Dichas sugerencias son la siguientes:

- Desarrollar un protocolo de enrutamiento para autenticación, usando criptografía de curvas elípticas para dispositivos móviles que integren una red inalámbrica *ad-hoc*.
- Desarrollar el algoritmo de firma digital.
- Se puede crear una entidad generadora de certificado digitales, usando criptografía de curvas elípticas.
- Implementar el protocolo utilizando el lenguaje de programación JAVA.

# Apéndice A

# RSA

## A.1. Método de cifrado y descifrado RSA

El método de cifrado y descifrado de un mensaje $M$ con el criptosistema RSA[1], procede de la siguiente manera [Rivest75].

Primero, se representa el mensaje $M$ como un entero positivo que se encuentra entre 0 y $(n-1)$ ($n$ es un entero positivo).

Para cifrar el mensaje se hace elevando el mensaje a la $e$-sima potencia módulo $n$. Esto es, el resultado (que se representa como $C$ al texto cifrado) es el residuo cuando $M^e$ es dividido por $n$.

$$C \equiv E(M) \equiv M^e (\text{mód} n)$$

Para descifrar el texto cifrado, se toma el residuo potencia $d$ módulo $n$.

$$D(C) \equiv C^d (\text{mód} n)$$

En el método de cifrado, este no incrementa el tamaño de un mensaje; tanto el mensaje, como el texto cifrado son enteros en el rango 0 y $n-1$.

La *clave de cifrado*, conocida también como *clave pública*, es el par de enteros positivos $(e, n)$. Similarmente, la *clave de descifrado*, conocida como *clave privada*, es el par de enteros positivos $(d, n)$. Cada usuario hace pública su clave de cifrado, y almacena la clave de descifrado.

[1] RSA, son las siglas de sus creadores: Rivest Ronald, Shamir Adi y Adleman Leonard

Para escoger las claves de descifrado como cifrado; primero se calcula el valor de $n$, como el producto de dos números primos $p$ y $q$:

$$n = p \cdot q$$

Esos números primos, son escogidos aleatoriamente y deben de ser muy grandes. Aunque es publico el valor de $n$, los valores de $p$ y $q$ se vuelven muy difícil de calcular, debido a la dificultad de factorizar el valor de $n$.

El valor de $d$ es un entero largo, escogido aleatoriamente. Se debe de verificar que realmente el valor de $d$ es un primo escogido aleatoriamente, considerando la siguiente condición:

$$MCD(d, (p-1) \cdot (q-1))) = 1$$

El entero $e$ es finalmente calculado desde $p$, $q$ y $d$, aplicando la inversa multiplicativa de $d$, módulo $(p-1) \cdot (q-1)$. Esto es:

$$e \cdot d \equiv 1(\text{mód}(p-1) \cdot (q-1))$$

Apéndice B

# Curvas recomendadas por NIST

A continuación se presentan algunas recomendaciones, por parte de NIST, para usar en los campos; tanto aquellos campos finitos primos ($\mathbb{F}_q$), como los campos binarios:

1.- Para el campo primo $\mathbb{F}_q$, tenemos:

$$\begin{array}{ll} p = 2^{194} - 2^{64} - 1 & p = 2^{254} - 2^{96} + 1 \\ p = 2^{384} - 2^{198} - 2^{96} - 2^{32} - 1 & p = 2^{521} - 1 \\ p = 2^{256} - 2^{192} + 2^{96} - 1 & \end{array}$$

2.- Para los campos binarios:

$$\begin{array}{ll} F_{2^{163}} & F_{2^{233}} \\ F_{2^{283}} & F_{2^{409}} \\ F_{2^{571}} & \end{array}$$

La correspondencia entre la longitud de la clave simétrica y el tamaño del campo se puede apreciar en la tabla B.1. Allí se observan distintos algoritmos de cifrado-descifrado, cuyos tamaños de longitud de los campos es recomendada para uso del gobierno de los Estados Unidos. Ahora bien, para las curvas elípticas existen también recomendaciones. Hay tres tipos de curvas elípticas que son:

1.- Curvas elípticas aleatorias sobre $\mathbb{F}_q$ (ver B.1).

2.- Curvas elípticas de Koblitz sobre $\mathbb{F}_{2^m}$ (ver B.2).

3.- Curvas elípticas aleatorias sobre $\mathbb{F}_{2^m}$ (ver B.3).

| **Longitud de clave** | **Algoritmo** | **Longitud de $\mathbb{F}_q$** | **Dimensión de m ($\mathbb{F}_{2^m}$)** |
|---|---|---|---|
| 80 | SKIP JACK | 192 | 163 |
| 112 | Triple-DES | 224 | 233 |
| 128 | Pequeño AES | 256 | 283 |
| 192 | Mediano AES | 384 | 409 |
| 256 | Grande AES | 521 | 571 |

Cuadro B.1: Tamaño de los campos recomendados para el uso del gobierno de U.S.

## B.1. Curvas elípticas aleatorias sobre $\mathbb{F}_q$

Los siguientes parámetros son dados para cada curva elíptica:

- **p:** El orden del valor primo $F_p$.
- **seedE:** Es usado para generar aleatoriamente los coeficientes de la curva elíptica.
- **r:** La salida de la función hash (SHA-1).
- **a ,b:** Los coeficientes de la curva elíptica ($E : y^2 = x^3 + ax + b$).
- $x_P$**,** $y_P$**:** las coordenadas del punto $P$.
- **n:** El orden de $P$.
- **h:** El cofactor.

### B.1.1. Curva P-192

| | |
|---|---|
| p | 6277101735386680763835789423207666416083908700390324961279 |
| seedE | 0x3045ae6f c8422f64 ed579528 d38120ea e12196d5 |
| r | 0x3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65 |
| a | -3 |
| b | 0x64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1 |
| $x_P$ | 0x188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012 |
| $y_P$ | 0x07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811 |
| n | 6277101735386680763835789423176059013767194773182842284081 |
| h | 1 |

### B.1.2. Curva P-224

| | |
|---|---|
| | 18291fb |
| a | -3 |
| b | 0xb4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2 355ffb4 |
| $x_P$ | 0xb70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 1 15c1d21 |
| $y_P$ | 0xbd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199 8 5007e34 |
| n | 26959946667150639794667015087019625940457807714424391721682722368061 |
| h | 1 |

### B.1.3. Curva P-256

| | |
|---|---|
| p | 1157920892103562487626974469494075735300861434152903141955336313088670978539 51 |
| seedE | 0x c49d3608 86e70493 6a6678e1 139d26b7 819f7e90 |
| r | 0x 7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af 317768 0104fa0d |
| a | -3 |
| b | 0x 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3b ce3c3e 27d2604b |
| $x_P$ | 0x 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4 a13945 d898c296 |
| $y_P$ | 0x 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cb b64068 37bf51f5 |
| n | 1157920892103562487626974469494075735299969552241357603424222590610685120443 69 |
| h | 1 |

### B.1.4. Curva P-384

| | |
|---|---|
| p | 39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319 |
| seedE | 0x a335926a a319a27a 1d00896a 6773a482 7acdac73 |
| r | 0x 79d1e655 f868f02f ff48dcde e14151dd b80643c1 406d0ca1 0dfe 6fc5 2009540a 495e8042 ea5f744f 6e184667 cc722483 |
| a | -3 |
| b | 0x b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314 088f 5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef |
| $x_P$ | 0x aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f7 41e0 82542a38 5502f25d bf55296c 3a545e38 72760ab7 |
| $y_P$ | 0x 3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c e9da 3113 b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f |
| n | 39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643 |
| h | 1 |

### B.1.5. Curva P-521

| | |
|---|---|
| p | 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559 640661454554977296311391480858037121987999716643812574028291115057151 |
| seedE | 0x d09e8800 291cb853 96cc6717 393284aa a0da64ba |
| r | 0x 0b4 8bfa5f42 0a349495 39d2bdfc 264eeeeb 077688e4 4fbf0ad8 f6d 0edb3 7bd6b533 28100051 8e19f1b9 ffbe0fe9 ed8a3c22 00b8f875 e523 868c 70c1e5bf 55bad637 |
| a | -3 |
| b | 051 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3 b8b489 91 8ef109e1 56193951 ec7e937b 1652c0bd 3bb1bf07 3573df88 3d2c34f 1 ef451fd4 6b503f00 |
| $x_P$ | 0x c6 858e06b7 0404e9cd 9e3ecb66 2395b442 9c648139 053fb521 f828 |

| | |
|---|---|
| | af60 6b4d3dba a14b5e77 efe75928 fe1dc127 a2ffa8de 3348b3c1 856a4 29b f97e7e31 c2e5bd66 |
| $y_P$ | 0x 118 39296a78 9a3bc004 5c8a5fb4 2c7d1bd9 98f54449 579b4468 17a fbd17 273e662c 97ee7299 5ef42640 c550b901 3fad0761 353c7086 a272 c240 88be9476 9fd16650 |
| n | 68647976601306097149819007990813932172694353001433054093944634591855431833976553942450577463332171975329639963713633211138647686124403803403728088927070054491 |
| h | 1 |

## B.2. Curvas elípticas de Koblitz sobre $\mathbb{F}_{2^m}$

Los parámetros de la curva de Koblitz y el punto base esta dado por la representación de la base normal (FR) y la representación de la base polinomial (FR2). A continuación se darán los parámetros para las curvas de Koblitz.

- **m:** El grado de extensión del campo binario $F_{2^m}$.
- **FR:** Un indicador de la representación usada por los elementos de $F_{2^m}$ de acuerdo con ANSI X9.62.
- **r:** La salida de la función hash (SHA-1).
- **a ,b:** Los coeficientes de la curva elíptica ($E : y^2 + xy = x^3 + ax^2 + b$).
- $x_P$**,** $y_P$**:** las coordenadas del punto $P$.
- **n:** El orden de $P$.
- **h:** El cofactor.
- **FR2:** Un indicador de la segunda representación usada por los elementos de $F_{2^m}$ de acuerdo con ANSI X9.62.
- **a2, b2:** Los coeficientes de la misma curva pero representados por FR2.
- $x_P2$**,** $y_P2$**:** Las coordenadas del punto base $P$ para FR2.

### B.2.1. Curva K-163

| | |
|---|---|
| m | 163 |
| FR | BNG, T=4 |
| a | 1 |
| b | 1 |
| $x_P$ | 0x 0 5679b353 caa46825 fea2d371 3ba450da 0c2a4541 |
| $y_P$ | 0x 2 35b7c671 00506899 06bac3d9 dec76a83 5591edb2 |
| n | 5846006549323611672814741753598448348329118574063 |
| h | 2 |
| FR2 | $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ |
| a2 | 1 |
| b2 | 1 |
| $x_P2$ | 0x 2 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8 |
| $y_P2$ | 0x 2 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9 |

### B.2.2. Curva K-233

| | |
|---|---|
| m | 233 |
| FR | BNG, T=2 |
| a | 0 |
| b | 1 |
| $x_P$ | 0x 0fd e76d9dcd 26e643ac 26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3 |
| $y_P$ | 0x 064 3e317633 155c9e04 47ba8020 a3c43177 450ee036 d6335014 34cac978 |
| n | 3450873173395281893717377931138512760570940988862252126328087024741343 |
| h | 4 |
| FR2 | $f(x) = x^{233} + x^{74} + 1$ |
| a2 | 0 |
| b2 | 1 |

| | |
|---|---|
| $x_P2$ | 0x 172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126 |
| $y_P2$ | 0x 1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3 |

### B.2.3. Curva K-283

| | |
|---|---|
| m | 283 |
| FR | BNG, T=6 |
| a | 0 |
| b | 1 |
| $x_P$ | 3ab9593 f8db09fc 188f1d7c 4ac9fcc3 e57fcd3b db150 24b 212c7022 9de5fcd9 2eb0ea60 |
| $y_P$ | 0x 2118c47 55e7345c d8f603ef 93b98b10 6fe8854f fe b9a3b3 04634cc8 3a0e759f 0c2686b1 |
| n | 3885337784451458141838923813647037813284811733793 061324295874997529815829704422603873 |
| h | 4 |
| FR2 | $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ |
| a2 | 0 |
| b2 | 1 |
| $x_P2$ | 0x 503213f 78ca4488 3f1a3b81 62f188e5 53cd265f 23 c1567a 16876913 b0c2ac24 58492836 |
| $y_P2$ | 0x 1ccda38 0f1c9e31 8d90f95d 07e5426f e87e45c0 e8 184698 e4596236 4e341161 77dd2259 |

### B.2.4. Curva K-409

| | |
|---|---|
| m | 409 |
| FR | BNG, T=4 |

| | |
|---|---|
| a | 0 |
| b | 1 |
| $x_P$ | 0x 1b559c7 cba2422e 3affe133 43e808b5 5e012d72 6c a0b7e6 a63aeafb c1e3a98e 10ca0fcf 98350c3b 7f89a9 75 4a8e1dc0 713cec4a |
| $y_P$ | 0x 1e36905 0b7c4e42 acba1dac bf04299c 3460782f 91 8ea427 e6325165 e9ea10e3 da5f6c42 e9c55215 aa9ca2 7a 5863ec48 d8e0286b |
| n | 3305279843951242994759576540163855199142023414821 4060964232439502288071128924919105067325845777745 80140963665906177313586 71 |
| h | 4 |
| FR2 | $f(x) = x^{409} + x^{87} + 1$ |
| a2 | 0 |
| b2 | 1 |
| $x_P2$ | 0x 060f05f 658f49c1 ad3ab189 0f718421 0efd0987 e3 07c84c 27accfb8 f9f67cc2 c460189e b5aaaa62 ee222e b1 b35540cf e9023746 |
| $y_P2$ | 0x 1e36905 0b7c4e42 acba1dac bf04299c 3460782f 91 8ea427 e6325165 e9ea10e3 da5f6c42 e9c55215 aa9ca2 7a 5863ec48 d8e0286b |

### B.2.5. Curva K-571

| | |
|---|---|
| m | 571 |
| FR | BNG, T=10 |
| a | 0 |
| b | 1 |
| $x_P$ | 0x 04bb2db a418d0db 107adae0 03427e5d 7cc139ac b4 65e593 4f0bea2a b2f3622b c29b3d5b 9aa7a1fd fd5d8b e6 6057c100 8e71e484 bcd98f22 bf847642 37673674 2 |

| | |
|---|---|
| | 9ef2ec5 bc3ebcf7 |
| $y_P$ | 0x 44cbb57 de20788d 2c952d7b 56cf39bd 3e89b189 84 bd124e 751ceff4 369dd8da c6a59e6e 745df44d 8220ce 22 aa2c852c fcbbef49 ebaa98bd 2483e331 80e04286 f eaa2530 50caff60 |
| n | 193226876150862917234767594546599367214946366485321749932861762572575957114478021226813397852270671183470671280082535146127367497406661731192968242161709250355n 5733685276673 |
| h | 4 |
| FR2 | $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$ |
| a2 | 0 |
| b2 | 1 |
| $x_P2$ | 0x 26eb7a8 59923fbc 82189631 f8103fe4 ac9ca297 00 12d5d4 60248048 01841ca4 43709584 93b205e6 47da30 4d b4ceb08c bbd1ba39 494776fb 988b4717 4dca88c7 e 2945283 a01c8972 |
| $y_P2$ | 0x 349dc80 7f4fbf37 4f4aeade 3bca9531 4dd58cec 9f 307a54 ffc61efc 006d8a2c 9d4979c0 ac44aea7 4fbebb b9 f772aedc b620b01a 7ba7af1b 320430c8 591984f6 0 1cd4c14 3ef1c7a3 |

## B.3. Curvas elípticas aleatorias sobre $\mathbb{F}_{2^m}$

Los parámetros que incluye este tipo de curvas son casi los mismos, vistos en parte de arriba, con excepción de que se incluye el valor *seedE*, además de que se usa una base normal Gaussiana (BNG). Estos parámetros quedarían como sigue:

- **m:** El grado de extensión del campo binario $F_{2^m}$.
- **FR:** Un indicador de la representación usada por los elementos de $F_{2^m}$ de acuerdo con ANSI X9.62.

- **seedE:** Este valor es usado para generar los coeficientes de la curva elíptica.
- **a ,b:** Los coeficientes de la curva elíptica ($E : y^2 + xy = x^3 + ax^2 + b$).
- $x_P$**,** $y_P$**:** las coordenadas del punto $P$.
- **n:** El orden de $P$.
- **h:** El cofactor.
- **FR2:** Un indicador de la segunda representación usada por los elementos de $F_{2^m}$ de acuerdo con ANSI X9.62.
- **a2, b2:** Los coeficientes de la misma curva pero representados por FR2.
- $x_P2$**,** $y_P2$**:** Las coordenadas del punto base $P$ para FR2.

### B.3.1. Curva B-163

| | |
|---|---|
| m | 163 |
| FR | BNG, T=4 |
| seedE | 0x 85e25bfe 5c86226c db12016f 7553f9d0 e693a268 |
| a | 1 |
| b | 0x 6 645f3cac f1638e13 9c6cd13e f61734fb c9e3d9fb |
| $x_P$ | 0x 0 311103c1 7167564a ce77ccb0 9c681f88 6ba54ee8 |
| $y_P$ | 0x 3 33ac13c6 447f2e67 613bf700 9daf98c8 7bb50c7f |
| n | 5846006549323611672814742442876390689256843201587 |
| h | 2 |
| FR2 | $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ |
| a2 | 1 |
| b2 | 0x 2 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd |
| $x_P2$ | 0x 3 f0eba162 86a2d57e a0991168 d4994637 e8343e36 |
| $y_P2$ | 0x 0 d51fbc6c 71a0094f a2cdd545 b11c5c0c 797324f1 |

### B.3.2. Curva B-233

| | |
|---|---|
| m | 233 |
| FR | BNG, T=2 |
| seedE | 0x 74d59ff0 7f6b413d 0ea14b34 4b20a2db 049b50c3 |
| a | 1 |
| b | 0x 1a0 03e0962d 4f9a8e40 7c904a95 38163adb 82521260 0c7752ad 52233279 |
| $x_P$ | 0x 18b 863524b3 cdfefb94 f2784e0b 116faac5 4404bc91 62a363ba b84a14c5 |
| $y_P$ | 0x 049 25df77bd 8b8ff1a5 ff519417 822bfedf 2bbd7526 44292c98 c7af6e02 |
| n | 6901746346790563787434755862277025555839812737345013555379383634485463 |
| h | 2 |
| FR2 | $f(x) = x^{233} + x^{74} + 1$ |
| a2 | 0 |
| b2 | 0x 066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad |
| $x_P2$ | 0x 0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b |
| $y_P2$ | 0x 100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052 |

### B.3.3. Curva B-283

| | |
|---|---|
| m | 283 |
| FR | BNG, T=6 |
| seedE | 0x 77e2b073 70eb0f83 2a6dd5b6 2dfc88cd 06bb84be |
| a | 1 |
| b | 0x 157261b 894739fb 5a13503f 55f0b3f1 0c560116 66 331022 01138cc1 80c0206b dafbc951 |
| $x_P$ | 0x 749468e 464ee468 634b21f7 f61cb700 701817e6 bc |

| | |
|---|---|
| | 36a236 4cb8906e 940948ea a463c35d |
| $y_P$ | 0x 62968bd 3b489ac5 c9b859da 68475c31 5bafcdc4 cc |
| | d0dc90 5b70f624 46f49c05 2f49c08c |
| n | 7770675568902916283677847627294075626569625924376 |
| | 904889109196526770044277787378692871 |
| h | 2 |
| FR2 | $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ |
| a2 | 0 |
| b2 | 0x 27b680a c8b8596d a5a4af8a 19a0303f ca97fd76 45 |
| | 309fa2 a581485a f6263e31 3b79a2f5 |
| $x_P2$ | 0x 5f93925 8db7dd90 e1934f8c 70b0dfec 2eed25b8 55 |
| | 7eac9c 80e2e198 f8cdbecd 86b12053 |
| $y_P2$ | 0x 3676854 fe24141c b98fe6d4 b20d02b4 516ff702 35 |
| | 0eddb0 826779c8 13f0df45 be8112f4 |

### B.3.4. Curva B-409

| | |
|---|---|
| m | 409 |
| FR | BNG, T=4 |
| seedE | 0x 4099b5a4 57f9d69f 79213d09 4c4bcd4d 4262210b |
| a | 1 |
| b | 0x 124d065 1c3d3772 f7f5a1fe 6e715559 e2129bdf a0 |
| | 4d52f7 b6ac7c53 2cf0ed06 f610072d 88ad2fdc c50c6f |
| | de 72843670 f8b3742a |
| $x_P$ | 0x 0ceacbc 9f475767 d8e69f3b 5dfab398 13685262 bc |
| | acf22b 84c7b6dd 981899e7 318c96f0 761f77c6 02c016 |
| | ce d7c548de 830d708f |
| $y_P$ | 0x 199d64b a8f089c6 db0e0b61 e80bb959 34afd0ca f2 |
| | e8be76 d1c5e9af fc7476df 49142691 ad303902 88aa09 |
| | bc c59c1573 aa3c009a |
| n | 661055968790248598951915308032771039828404682964 |

| | |
|---|---|
| | 81219284648798304157774827374805208143723762179110965979 8n 67288366567526771 |
| h | 2 |
| FR2 | $f(x) = x^{409} + x^{87} + 1$ |
| a2 | 0 |
| b2 | 0x 021a5c2 c8ee9feb 5c4b9a75 3b7b476b 7fd6422e f1f3dd67 4761fa99 d6ac27c8 a9a197b2 72822f6c d57a55aa 4f50ae31 7b13545f |
| $x_P$2 | 0x 15d4860 d088ddb3 496b0c60 64756260 441cde4a f1771d4d b01ffe5b 34e59703 dc255a86 8a118051 5603aeab 60794e54 bb7996a7 |
| $y_P$2 | 0x 061b1cf ab6be5f3 2bbfa783 24ed106a 7636b9c5 a7bd198d 0158aa4f 5488d08f 38514f1f df4b4f40 d2181b36 81c364ba 0273c706 |

### B.3.5. Curva B-571

| | |
|---|---|
| m | 571 |
| FR | BNG, T=10 |
| seedE | 0x 2aa058f7 3a0e33ab 486b0f61 0410c53a 7f132310 |
| a | 1 |
| b | 0x 3762d0d 47116006 179da356 88eeaccf 591a5cde a7500011 8d9608c5 9132d434 26101a1d fb377411 5f586623 f75f0000 1ce61198 3c1275fa 31f5bc9f 4be1a0f4 67f01ca8 85c74777 |
| $x_P$ | 0x 0735e03 5def5925 cc33173e b2a8ce77 67522b46 6d278b65 0a291612 7dfea9d2 d361089f 0a7a0247 a184e1c7 0d417866 e0fe0feb 0ff8f2f3 f9176418 f97d117e 624e2015 df1662a8 |
| $y_P$ | 0x 04a3642 0572616c df7e606f ccadaecf c3b76dab 0eb1248d d03fbdfc 9cd3242c 4726be57 9855e812 de7ec5 |

| | |
|---|---|
| | c5 00b4576a 24628048 b6a72d88 0062eed0 dd34b109 6 d3acbb6 b01a4a97 |
| n | 3864537523017258344695351890931987344298927329706 4349n 9865723525145151914228956042453614399938941 5773083133n 881121926944486246872462816813070234528288308333241139n 3191105285703 |
| h | 2 |
| FR2 | $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$ |
| a2 | 1 |
| b2 | 0x 2f40e7e 2221f295 de297117 b7f3d62f 5c6a97ff cb 8ceff1 cd6ba8ce 4a9a18ad 84ffabbd 8efa5933 2be7ad 67 56a66e29 4afd185a 78ff12aa 520e4de7 39baca0c 7 ffeff7f 2955727a |
| $x_P2$ | 0x 303001d 34b85629 6c16c0d4 0d3cd775 0a93d1d2 95 5fa80a a5f40fc8 db7b2abd bde53950 f4c0d293 cdd711 a3 5b67fb14 99ae6003 8614f139 4abfa3b4 c850d927 e 1e7769c 8eec2d19 |
| $y_P2$ | 0x 37bf273 42da639b 6dccfffe b73d69d7 8c6c27a6 00 9cbbca 1980f853 3921e8a6 84423e43 bab08a57 6291af 8f 461bb2a8 b3531d2f 0485c19b 16e2f151 6e23dd3c 1 a4827af 1b8ac15b |

Apéndice C

# Biblioteca NTL

**NTL** es una librería de alto rendimiento y portable, desarrollada en el lenguaje C++, proviendo estructuras de datos y algoritmos para enteros de longitud arbitraria; para vectores, matrices, y polinomios sobre enteros de precisión de punto flotante arbitrarios. NTL provee una alta calidad en la implementación de los algoritmos:

- Aritmética de enteros de longitud arbitraria y la aritmética de precisión de puntos flotantes arbitrarias.
- La aritmética polinomial sobre los enteros y campos finitos incluyendo la aritmética básica, factorización polinomial, pruebas de irreducibilidad, cálculos de los polinomios mínimos, trazas, normas, y más.
- Reducción de bases entramadas, incluyendo una muy robusta y rápida implementación de Schnorr-Eucher, la reducción de bloque de Kookie-Zolotarev, y la nueva heurística de podado (*pruning*) para el bloque Korkin-Zolotarev.
- Álgebra lineal básica sobre los enteros, campos finitos, y números de precisión de punto flotante arbitrario.

La aritmética polinomial que maneja la librería NTL, es una de las más rápidas disponibles y además a sido objeto de obtener un gran reconocimiento por alcanzar registros mundiales para factorizar polinomios, así como para determinar el orden de las curvas elípticas.

La reducción del código entramado de NTL es también de los mejores disponibles; esto en términos de velocidad y robustez. Así pues, NTL a servido o ha sido usado para

'romper' algunos criptosistemas.

Resulta prescindible mencionar que además de las grandes cualidades que muestra NTL sobre las operaciones que se pueden realizar, cabe destacar la gran portabilidad con la que fue diseñada, es decir, se puede instalar en una máquina que tenga UNIX o Linux así como aquellas corriendo Windows (con versiones superiores o iguales a 95) ó las PCs que tiene un sistema operativo Macintosh. Esa portabilidad que se le hace tanta alusión, se debe a que se trató de evitar las características esotéricas de C++ y también evitar el código ensamblado.

También NTL fue diseñado de tal manera que su interfaz (entiéndase por *interfaz* en el sentido estricto de la programación) proporcione una gran variedad de clases que representan a objetos matemáticos. Esto provee un buen entorno para así de manera fácil y rápida se implementen los nuevos algoritmos sin sacrificar el rendimiento.

### C.0.6. Interfaz de programación

La interfaz de programación de la librería NTL resulta ser muy variada. Vamos a tratar de mencionar de que se compone dicha interfaz.

#### Clases de anillos básicos.

Las clases de anillos básicos son:

- **zz:** Enteros grandes.
- **ZZ_p:** Enteros grandes módulo $p$.
- **zz_p:** Enteros módulo precisión simple $p$.
- **GF2:** Enteros módulo 2.
- **ZZX:** Polinomios univariantes sobre **zz**.
- **ZZ_pX:** Polinomios univariantes sobre **ZZ_p**.
- **zz_pX:** Polinomios univariantes sobre **zz_p**.
- **GF2X:** Polinomios sobre **GF2**.
- **ZZ_pE:** Campo/anillo extensión sobre **ZZ_p**.

- **zz_pE:** Campo/anillo extensión sobre **zz_p**.
- **GF2E:** Campo/anillo extensión sobre **GF2**.
- **ZZ_pEX:** Polinomio univariante sobre **ZZ_pE**.
- **zz_pEX:** Polinomio univariante sobre **zz_pE**.
- **GF2EX:** Polinomio univariante sobre **GF2E**.

Todas las clases soportan a los operadores aritméticos básicos, que son:

$$+, -, \text{(unary)-}, +=, -=, ++, --, *, *=, /, /=, \%, \%=$$

sin embargo, las operaciones de % y %= únicamente existen para las clases polinomial y los enteros, y no existen para las clases: ZZ_p, zz_p GF2, ZZ_PE, zz_pE, y GF2E.

Las clases polinomial y los enteros soportan los operadores de corrimiento, tanto por izquierda como por la derecha. Para la clase polinomial, eso significa la multiplicación o la división por una potencia de x.

### Clase punto flotante

NTL provee tres diferentes clases de punto flotante, que son:

- **xdouble**: Punto flotante de doble precisión con rango a exponente extendido, esto para números muy grandes.
- **quad_float**: punto flotante de cuádruple precisión.
- **RR**: Punto flotante de precisión arbitraria.

### Vectores y Matrices

Los vectores y las matrices se pueden encontrar dentro de: ZZ, ZZ_p, zz_p, GF2, ZZ_pE, zz_pE, GF2E, y RR, el cual soportan las operaciones de aritmética tradicional.

### La forma procedural y funcional

Las funciones que define la librería NTL, se pueden representar en dos formas: **forma procedural** y **forma funcional**. Con el siguiente ejemplo se mostrará el uso de las dos formas.

**Ejemplo C.1** *Forma procedural y funcional.* Considérese el siguiente código:

```
ZZ x, a, n;
x=InvMod(a,n);  //Forma funcional.
InvMod(x,a,n);  //Forma procedural.
```

El uso de la forma procedural puede ser muy eficiente ya que generalmente evita la creación de objetos temporales que van almacenando el resultado. Sin embargo, en ocasiones resulta conveniente evitar tales eficiencias. En cambio, la forma funcional es usualmente preferible debido a que el código resultante es usualmente fácil de entender.

### Conversiones y promociones

NTL no provee una manera de convertir tipos de datos de manera directa, es decir, de un tipo de dato primitivo tratarlo de convertir a un tipo de dato ZZ por ejemplo.

Aunque no existen las conversiones de manera automática en NTL, los programadores pueden todavía obtener más de los beneficios de NTL, ya que las operaciones de aritmética básica los operadores de comparación y asignación fueron sobrecargados para poder obtener el efecto de una promoción automática.

**Ejemplo C.2** *Promoción y conversión.* Considere el siguiente código:

```
ZZ x,a;
x=a+1;
if(x<0)
  mul(x,2,a);
else
  x=-1;
```

Existe la siguiente notación de la promoción para el operador de la suma (+).

```
ZZ operator+(const ZZ& a, const ZZ&b);
```

con lo anterior se tiene que en el ejemplo C.2 en la segunda línea 2 ($x = a + 1$) no se considera una operación directa de la suma de un número de tipo ZZ y otro numérico, si no que se hace uso de la notación anterior.

En el apéndice E, se muestra una tabla con todas las posibilidades de conversión entre los tipos de datos de la librería NTL.

Usualmente las conversiones y promociones son semánticamente equivalentes, sin embargo existen tres tipos de excepciones, que son:

1.- Conversión de punto flotante *double* a ZZ. Es decir, tenemos lo que sería un error de conversión:

```
ZZ a;
double x;

a=a+x;   //El compilador devuelve un error
         //de incompatibilidad de tipos.
```

la manera correcta sería como sigue:

```
a=a+to_ZZ(x);
```

.

2.- Conversión de tipo de dato *unsigned int* o *unsigned long* a ZZ. La manera segura de evitar este tipo de error es con el operador de conversión explícita.

3.- Puede ocurrir este tipo de excepción (tipos de datos incompatibles) por que la máquina en donde se esta ejecutando la aplicación es de 64-bits, y ocurre que este tipo de máquinas cuando convierten un tipo de dato *unsigned long* o *long* a tipos de punto flotante de precisión extendida de NTL (RR ó quad_float) le resulta imposible por el echo de que las arquitecturas para las que fue desarrollada es x86.

### C.0.7. Principales módulos de NTL

Así como la gran mayoría de las librerías, NTL consiste de un gran número de módulos de software, es decir, módulos que contienen una cabecera (*.h), un archivo de documentación (*.txt) y el código fuente (*.c). Además de las cabeceras que vienen con el compilador por omisión. En el apéndice F, se muestra un resumen de los principales módulos de NTL.

Todas las cabeceras de los archivos de las clases polinomiales (ZZ_pX, ZZX, etc), declaran los vectores genéricos: vec_ZZ_pX, vec_ZZX, etc.

### C.0.8. Algoritmos

NTL hace uso consistentemente de los algoritmos más rápidos. La multiplicación de enteros largos son implementados usando algoritmos clásicos, como son el llamado *Karatsuba* para números muy grandes. La división de enteros largos es actualmente solo implementado usando el algoritmo clásico.

La multiplicación y división polinomial es llevada acabo, usando una combinación de algoritmos clásicos, Karatsuba, el *FFT* usando primos pequeños, y el FFT usando la propuesta de *Schoenhagge-Strassen*. El algoritmo usado depende de el dominio de los coeficientes.

Apéndice D

# Biblioteca Socket

Una API (*'Application Program Interface'*) es una interfaz disponible para el programador o la persona que lo necesite. La disponibilidad de una API depende tanto del sistema operativo en el que será usado, como del lenguaje de programación. Las dos grandes APIs de comunicación que prevalecen en los sistemas UNIX [Stevens90] son los denominados **sockets Berkeley** y la interfaz de la capa de transporte **Sistema V** (*"System V Transport Layer Interface"*, TI). Ambas interfaces fueron implementadas en el lenguaje C. Esta sección se enfoca en tratar de dar una pequeña pero nutrida explicación del funcionamiento de la API de sockets Berkeley.

### D.0.9. Resumen sobre Sockets

La interfaz propuesta por Berkeley fue primero provista con el sistema 4.1cBSD por VAX, hacia 1982. Esta versión soporta los siguientes protocolos de comunicación:

- Dominio UNIX.
- Dominio de Internet (TCP/IP).
- Domino Xerox NS.

La figura D.1 muestra un esquema típico donde se puede apreciar el modo de operar del **protocolo orientado a conexión**. Desde el punto de vista de línea de tiempo se puede observar como primero inicia la parte del servidor y se queda ciclado hasta que un cliente intente conectarse. Por eso es que cualquier cliente no puede establecer conexión alguna con el servidor hasta que este no este en esa parte. En cuanto al cliente, inmediatamente

después que se inicializa intentará conectarse con el servidor, en seguida de esto le envía los datos (**write()**) al servidor el cual este los lee (**read()**), procesa esos datos y retorna la respuesta (**write()**) y el cliente los lee (**read()**).

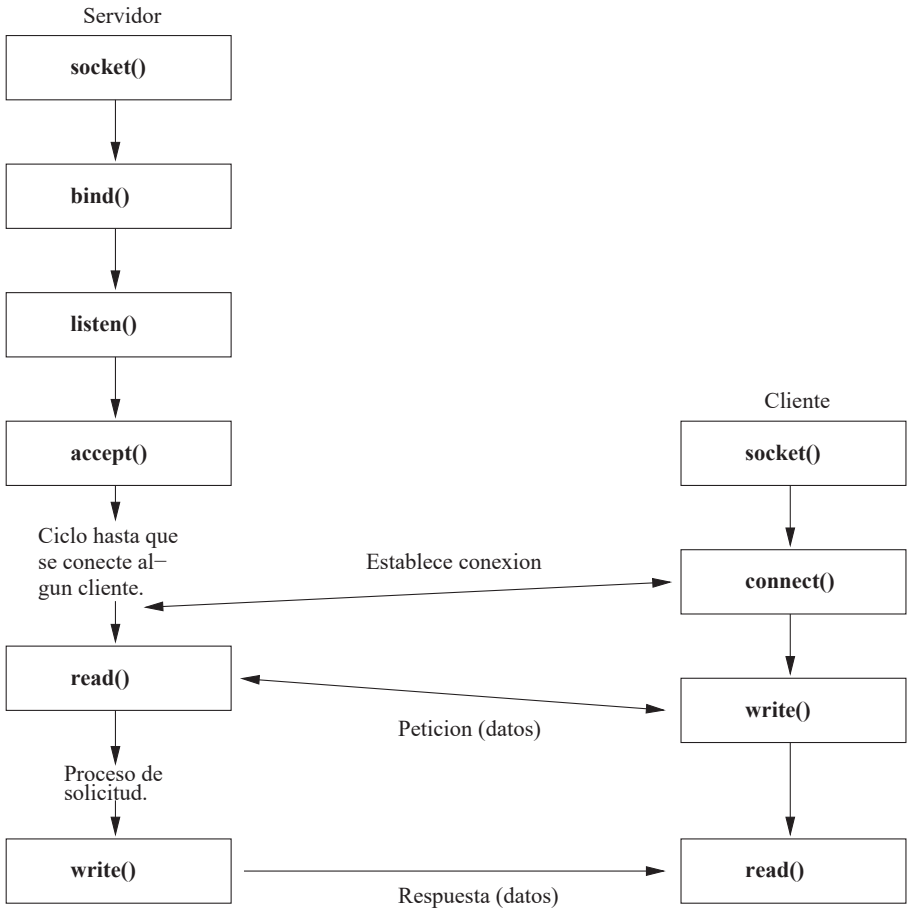



Figura D.1: Sistema de llamadas de socket para el protocolo orientado a conexión.

Para el caso donde se tiene un protocolo diferente al orientado a conexión en el modelo cliente-servidor, este es el llamado **protocolo no orientado a la conexión**. Es decir, el cliente (ver figura D.2) no establece una conexión con el servidor. Instantáneamente el cliente solo envía un datagrama al servidor usando el sistema de llamadas *"sendto()"*, el cual únicamente requiere de la dirección destino (que es la dirección que le pertenece al servidor) como parámetro. Similarmente, el servidor no tiene que aceptar una conexión desde el cliente, sino que instantáneamente, el servidor usa el sistema de llamadas *recvfrom()"* que espera hasta que reciba datos por parte del cliente. El módulo *recvfrom"* devuelve la

dirección de red que corresponde al cliente, además que con el datagrama de esta manera el servidor puede enviar su respuesta al cliente correctamente.

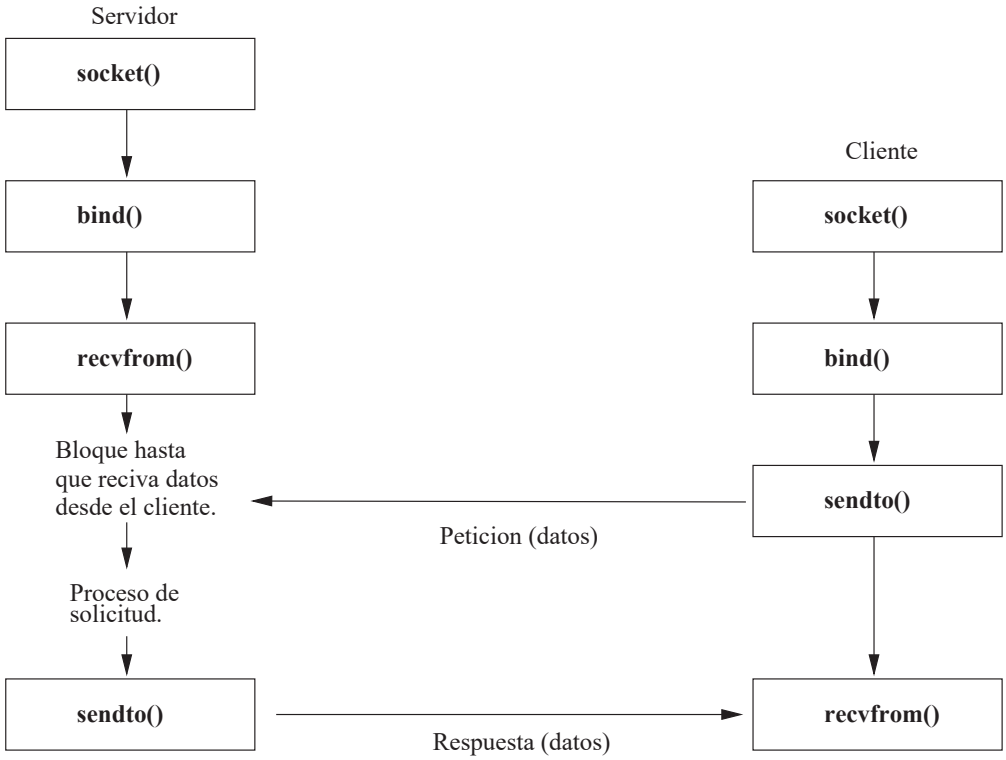



Figura D.2: Sistema de llamadas para el protocolo no orientado a conexión

### D.0.10. Dirección del Socket

Para los distintos protocolos de comunicación se usan diferentes estructuras o definiciones de estructuras. Pare el caso de la familia de Internet las siguientes estructuras son definidas dentro de <*netinet/in.h*>:

```
struct in_addr
{
  unsigned long s_addr;  /* 32-bit netid/hostid */
                         /* byte de red ordenado */
};

struct sockaddr_in
{
  short sin_family;        /*AF_INET*/
  unsigned short sin_port; /*16-bit numero de puerto*/
```

```
                          /*ordenado byte de red.*/
  struct in_addr sin_addr; /*32-bit netid/hostid*/
                          /*ordenado byte de red*/
  char sin_zero[8];       /*sin usar*/
};
```

De manera gráfica (ver figura D.3) se puede observar esta definición.

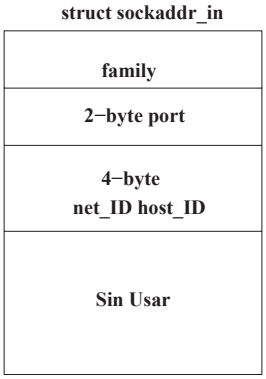



Figura D.3: Estructura de la dirección de red para el protocolo de Internet

El sistema de llamadas de red, tal como *connect()* y *bind()*, trabajan con dicha estructura. El sistema de llamadas de red toma dos parámetros: la dirección genérica de la estructura *sockaddr* y el tamaño actual de la estructura del protocolo específico.

**Ejemplo D.1** *Invocar a connect()*. Invocar al sistema de llamadas *connect()* por un socket de dominio de Internet.

```
struct sockaddr_in serv_addr;
...
connect(sockfd, (struct sockaddr *)& serv_addr,
        sizeof(serv_addr));
```

el segundo argumento es un puntero a la estructura del dominio de Internet y el tercero es su tamaño en bytes (16-bytes).

### D.0.11. Sistemas de llamadas de socket

Se tratará de describir el sistema de llamadas que se requiere para poder realizar la programación de los sockets.

| | |
|---|---|
| AF_UNIX | Protocolos internos de UNIX. |
| AF_INET | Protocolo de Internet. |
| AF_NS | Protocolo de XeroxNS. |
| AF_IMPLINK | Protocolo de enlace IMP. |

Cuadro D.1: Valores de las constantes del primer argumento *int family*.

| | |
|---|---|
| SOCK_STREAM | Socket para flujos (*stream*). |
| SOCK_DGRAM | Socket datagrama. |
| SOCK_RAW | Socket raw. |
| SOCK_SEQPACKET | Socket paquete de secuencia. |
| SOCK_RDM | Socket de mensaje de entrega confiable. |

Cuadro D.2: Valores que se pueden usar en el segundo argumento (*int type*).

**socket()**

Para establecer la comunicación de entrada y/o salida, la primer cosa que debemos hacer es llamar al sistema de llamadas **socket** especificando el tipo de protocolo de comunicación deseado (TCP, UDP, XNS, SPP, etc), para ello se debe de hacer uso de la siguiente definición:

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

Para el primer argumento (*int family*) se puede usar cualquiera de las constantes de se muestran en la tabla D.1.

El prefijo **AF** se establece debido a la frase *'Address Family'*.

Para el segundo argumento (*int type*) se puede utilizar cualquiera de los valores que se presentan en la tabla D.2.

No todas las combinaciones de los dos primeros parámetros (*family* y *type*) son válidos. En la tabla D.3 se muestran las combinaciones válidas, así como las que no lo son y en algunos casos se muestra el protocolo que puede resultar de dicha combinación.

El tercer parámetro (*int protocol*) del sistema de llamadas es fijo a 0 para muchas de las aplicaciones. Las combinaciones entre los tres parámetros se muestra en la tabla D.4

| | **AF_UNIX** | **AF_INET** | **AF_NS** |
|---|---|---|---|
| SOCK_STREAM | Si | TCP | SPP |
| SOCK_DGRAM | Si | UDP | IDP |
| SOCK_RAW | No | IP | Si |
| SOCK_SEQPACKET | No | No | SPP |

Cuadro D.3: Combinaciones válidas de los dos parámetros del sistema de llamadas *socket()*.

| **family** | **type** | **protocol** | **Protocolo actual** |
|---|---|---|---|
| AF_INET | SOCK_DGRAM | IPPROTO_UDP | UDP |
| AF_INET | SOCK_STREAM | IPPROTO_TCP | TCP |
| AF_INET | SOCK_RAW | IPPROTO_ICMP | ICMP |
| AF_INET | SOCK_RAW | IPPROTO_RAW | (raw) |
| AF_NS | SOCK_STREAM | NSPROTO_SPP | SPP |
| AF_NS | SOCK_SEQPACKET | NSPROTO_SPP | SPP |
| AF_NS | SOCK_RAW | NSPROTO_ERROR | Protocolo Ethernet |

Cuadro D.4: Combinaciones de los tres parámetros de entrada en *socket()*.

mostrando el protocolo (columna titulada *Protocolo actual*) que se podrá usar en caso de alguna de las combinaciones.

### bind()

El sistema de llamada *bind()* asigna un nombre a un socket sin que este hubiese sido nombrado. La manera como se define este sistema de llamadas es como sigue:

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

El segundo parámetro es un puntero a la dirección del protocolo específico y el tercer dato de entrada es el tamaño de la estructura de la dirección. Existen tres usos de este sistema de llamadas:

1.- El servidor registra su dirección bien conocida con el sistema. Tanto un servidor orientado a la conexión o como aquel no orientado a la conexión.

2.- Un cliente puede registrar una dirección específica para si misma.

3.- Un cliente que no esta orientado a la conexión, necesita asegurar que el sistema asigne alguna dirección única, para que el servidor devuelva una dirección válida en su respuesta.

**connect()**

Para que un cliente se pueda conectar con un servidor necesita hacer uso del sistema de llamadas *connect()*. El cliente se conecta con un descriptor socket (un *descriptor socket* ó *sockfd*, es una quíntupla formada con los siguiente elementos **{protocol, local_addr, local_process, foreing_addr, foreing_process}**) en seguida del sistema de llamadas *socket()*, estableciendo una conexión con el servidor. La definición de esta llamada se muestra a continuación:

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
```

El primer parámetro de entrada, *int sockfd()*, es un descriptor de socket que fue devuelto por el sistema de llamadas *socket()*. El segundo parámetro de entrada (*struct sockaddr *servaddr*) es un puntero a la dirección del socket y tercer parámetro (*int addrlen*) es el tamaño del mismo.

**listen()**

Este sistema de llamadas es usado por el servidor que es orientado a la conexión, el cual debe de esperar hasta que se establece una conexión. Su definición es como sigue:

```
int listen(int sockfd, int backlog);
```

Este es ejecutado justamente después del sistema de llamadas *socket()* y *bind()*, e inmediatamente antes del sistema de llamadas *accept()*. El parámetro *backlog* específica cuantas peticiones de solicitud pueden ser en listadas mientras el sistema espera a que el servidor ejecute el sistema de llamada *accept()*.

**accept()**

Después que el servidor orientado a la conexión ejecuta la llamada al sistema *listen()*, una conexión desde algún proceso cliente espera para ser atendido por el sistema de llamadas del servidor llamado *accept()*. Su definición se muestra a continuación:

| | |
|---|---|
| MSG_OOB | Envía o recibe datos fuera de banda (para cuando se usa mucho ancho de banda). |
| MSG_PEEK | Vistazo a los mensajes que llegan. Usado por *recv* y *recvfrom*. |
| MSG_DONTROUTE | Evitar el ruteo. Usado por *send* y *sendto*. |

Cuadro D.5: Posibles valores del parámetro *flags*.

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr* peer, int *addrlen);
```

Este sistema de llamadas toma la primera petición de conexión de la lista y crea un socket con las mismas propiedades con las que cuenta *sockfd*. Si no se encuentra otra petición de conexión pendiente en la lista, se bloquea hasta que haya otra petición de conexión.

### send(), sendto(), recv() y recvfrom()

Estos sistemas de llamadas son todos idénticos a las tradicionales llamadas de sistema, como son: *read* y *write*, pero con la pequeña diferencia de los parámetros de entrada. Las definiciones se representan como sigue:

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int sockfd, char *buff, int nbytes, int flags);

int sendto(int sockfd, char *buff, int nbytes, int flags,
           struct sockaddr *to, int addrlen);

int recv(int sockfd, char *buff, int nbytes, int flags);

int recvfrom(int sockfd, char *buff, int nbytes, int flags,
             struct sockaddr *from, int *addrlen);
```

El parámetro *buff* contendrá el mensaje que se desea enviar o que se recibe y *nbytes* contiene el tamaño de dicho mensaje en bytes. El parámetro *flags* puede ser cero o puede contener los valores de las constantes que se muestran en la tabla D.5.

El parámetro *to* del sistema de llamadas *sendto()* específica la dirección hacia donde los datos serán o son enviados. Ahora, el parámetro *from* del sistema de llamadas *recvfrom()* se fija la dirección de donde son enviados los datos. La longitud de la dirección es retornada en el parámetro *addrlen*.

**close()**

Este sistema de llamadas es el usado por el sistema operativo UNIX. Su definición se hace como sigue:

```
int close(int fd);
```

Si el socket a sido cerrado con un protocolo que tiene un envió confiable (TCP ó SPP), es decir, orientado a la conexión, el sistema debe de asegurar que algún dato dentro del kernel que todavía este transmitiendo debe de ser enviado.

Apéndice E

# Conversiones de tipos de NTL

Aquí se muestran, las diferentes conversiones que se pueden hacer, entre los distintos tipos de datos que ofrece la biblioteca NTL [Shoup].

| **Origen** | **Destino** |
|---|---|
| int | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR |
| long | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR |
| float | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR |
| double | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR |
| uint | ZZ |
| ulong | ZZ |
| xdouble | int, long, uint, ulong, ZZ, float, double, xdouble, RR, cstr |
| quad_float | int, long, uint, ulong, ZZ, float, double, quad_float, RR, cstr |
| RR | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR, cstr |
| ZZ | int, long, uint, ulong, ZZ, float, double, xdouble, quad_float, RR, cstr |
| ZZ_p | long, ZZ |
| vec_ZZ_p | vec_ZZ |
| ZZ_pX | long, ZZ_p, ZZ, ZZX, vec_ZZ_p |
| zz_p | long, ZZ |
| vec_zz_p | vec_ZZ |

Cuadro E.1: Conversión de tipos de datos en NTL. Parte I.

| **Origen** | **Destino** |
|---|---|
| ZZ_pX | long, zz_p, ZZ, ZZX, vec_zz_p |
| vec_ZZ | vec_ZZ_p, vec_zz_p |
| ZZX | long, ZZ, ZZ_pX, zz_pX |
| GF2 | long, ZZ |
| vec_GF2 | GF2X |
| GF2X | long, ZZ, GF2, vec_GF2 |
| GF2E | long, ZZ, GF2, GF2X |
| GF2EX | long, ZZ, GF2, GF2E, GF2X, vec_GF2E |
| mat_ZZ_p | mat_ZZ |
| mat_zz_p | mat_ZZ |
| ZZ_pE | long, ZZ, ZZ_p, ZZ_pX |
| ZZ_pEX | long, ZZ, ZZ_p, ZZ_pE, ZZ_pX |
| zz_pE | long, ZZ, zz_p, zz_pX |
| zz_pEX | long, ZZ, zz_p, zz_pE, zz_pX |

Cuadro E.2: Conversión de tipos de datos en NTL. Parte II.

Apéndice F

# Módulos principales de NTL

Existen muchas clases en la biblioteca de NTL que permiten llevar a cabo distintas operaciones. Aquí se describen las que creo más importantes [Shoup].

| **Módulo** | **Descripción** |
|---|---|
| GF2 | Clase GF2 para enteros módulo 2. |
| GF2X | Clase GF2X para polinomios sobre $F_2$. |
| GF2XFactoring | Rutinas para factorización de polinomios sobre $F_2$, también incluye rutinas para probar y construir polinomios irreducibles. |
| GF2XVec | Para vectores de longitud fija de GF2X de longitud fija menos flexible pero más eficiente que vec_GF2X. |
| GF2E | Extensión polinomial campo/anillo sobre $F_2$ implementado como $GF(2)[x]/(P)$.[1] |
| GF2EX | Polinomios sobre GF2E, incluyendo rutinas para aritmética polinomial módulo, composición polinomial, polinomios característicos y mínimos e interpolación. |
| GF2EXFactoring | Rutinas para factorización polinomial sobre GF2E, también incluye rutinas para probar y construir polinomios irreducibles[2] |

Cuadro F.1: Módulos principales de NTL. Parte I.

| **Módulo** | **Descripción** |
|---|---|
| RR | Números de punto flotante de precisión arbitraria. |
| ZZ | Enteros de longitud arbitraria incluye rutinas para GCDs, símbolos Jacobianos, aritmética modular; también incluye rutinas para generar números primos pequeños y rutinas en línea para aritmética modular de precisión simple[3].<br>números de precisión única. |
| ZZX | Polinomios sobre ZZ; incluye rutinas para GCDs polinomios característicos y mínimos, normas y trazas. |
| ZZ_pEX | Polinomios sobre ZZ_pE; incluye rutinas para aritmética de polinomios<br>modular, composición modular, polinomios característicos y mínimos, así como la interpolación. |
| ZZ_pEXFactoring | Rutinas para factorización polinomios sobre ZZ_pE; también incluye rutinas para probar y construir polinomios irreducibles. |
| ZZ_pX | Polinomios sobre ZZ_p; incluye rutinas para aritmética de polinomios modular, composición modular, polinomios característicos y mínimos, e interpolación. |

Cuadro F.2: Módulos principales de NTL. Parte II.

| **Módulo** | **Descripción** |
|---|---|
| ZZ_pXFactoring | Rutinas para factorizar polinomios sobre ZZ_pE; también incluye rutinas para probar y construir polinomios irreducibles. |
| matrix | Plantilla macro para arreglos bidimensionales de tamaño dinámico. |
| mat_GF2<br>mat_GF2E<br>mat_RR<br>mat_ZZ<br>mat_ZZ_p<br>mat_ZZ_pE<br>mat_lzz_p<br>mat_lzz_pE | Incluye operaciones aritméticas básicas para matrices, incluyendo calculo de matrices, inversa de una matriz , resolución de sistemas de ecuaciones lineales no singulares y en algunas otras, la eliminación Gaussiana. |
| pair | Plantilla de macro para pares. |
| quad_float | Números de punto flotante de cuádruple precisión. |
| tools | Algunos tipos básicos y rutinas de utilidad, incluyendo la función de tiempo (*GetTime()*), y varias versiones de sobrecarga de *min()* y *max()*. |
| vector | Macros de plantilla para vectores de dimensión dinámica, de esta también dependen las clases: vec_GF2E, vec_ZZ_p, vec_ZZ_pE, vec_lzz_p y vec_lzz_pE, con aritmética básica. |
| xdouble | Números de punto flotante de doble precisión con un rango de exponente extendido. |

Cuadro F.3: Módulos principales de NTL. Parte IV.

Apéndice G

# Sistemas operativos incrustados

Aquí se listan los distintos sistemas operativos incrustados y su fabricante, los cuales son clasificados de acuerdo al uso para el que son diseñados.

| **Sistema Operativo** | **Fabricante** |
|---|---|
| PalmOS | Palm Inc. |
| ÉPOCA | Pisón (UK), ahora llamado *Symbion OS*. |
| Windows CE (*Windows Compact Edition*) | Microsoft. |
| PocketPC | Variante de Windos CE. |
| Windows Mobile | Variante de Windows CE. |
| Linux | SharpZaurus y iPAQ. |
| DOS | Poet PC. |
| Newton OS | Applet Newton. |
| **Teléfonos inteligentes (*Saxophone*).** | |
| Windows CE | |
| Embedded Linux | Monetarist: Linux Motorola, A760, E680. |
| Mobiliary | Monetarist. |
| Symbion OS | |

Cuadro G.1: Sistemas operativos incrustados. Parte I.

| **Sistema Operativo** | **Fabricante** |
|---|---|
| **Routers**. | |
| ION (*Interneuron Operating System*) | Casco Systems. |
| ION-XI | Casco Systems. |
| Cat OS | Casco Systems. |
| PIX OS | Casco Systems. |
| JUNCOS | Juniper Networks. |
| ROSS | Ruggedly. |
| **Microordenadores; Sistema Operativo en Tiempo Real (RTOS).** | |
| Contigo | Escrito en lenguaje C. |
| Unix | Escrito en 6502. |
| eCOS | |
| FreeRTOS | |
| INTEGRITY | |
| Lynx OS | |
| OSEK | |
| Montavista Linux | |

Cuadro G.2: Sistemas operativos incrustados. Parte II.

| **Sistema Operativo** | **Fabricante** |
|---|---|
| Nucleus | |
| OS-9 | Microware. |
| QNX | |
| Rtems | |
| RT Linux | |
| Salvo | |
| TreadX | |
| TroN | |
| $\mu$CLinux | |
| VRTX | |
| VxWorks | |

Cuadro G.3: Sistemas operativos incrustados. Parte III.

Apéndice H

# PDAs con soporte para Linux

En seguida se muestran algunas de las PDAs que soportan Linux o por su parte tienen el sistema operativo Linux de manera nativa.

| **Modelo** | **Soporte Linux** | **Linux Nativo** |
|---|---|---|
| **Acer N series** | | |
| N10 | Si | No |
| N30 | Si | No |
| **Archos** | | |
| Pma400 | Si | Si |
| **Asus MyPal** | | |
| A730(W) | Si | No |
| A620/A620BT | No | No |
| A716/A712 | Si | No |
| **Dell Axim** | | |
| X3/X3i | | No |
| X30 | Si | No |
| X5 | Si | |
| X50 | Si | No |
| X51 | No | No |

Cuadro H.1: PDAs con Linux. Parte I.

| **Modelo** | **Soporte Linux** | **Linux Nativo** |
|---|---|---|
| **Fujitsu Siemens Loox** | | |
| Loox 410 / 420 | Si | No |
| Loox 600 | No | No |
| Loox 718 / 720 | Si | No |
| Simpad[1] | Si | No |
| **HP/Compaq iPAQ** | | |
| h3100 series | Si | No |
| h3600/h3700 series | Si | No |
| h3800 series | Si | No |
| h3900 series | Si | No |
| h5100/h5400/h5500 series | Si | No |
| h4100/h4300 series | Si | No |
| h2200 series | Si | No |
| h1910 series | Si | No |
| h1930/h1940 series | Si | No |
| rx3400 series | No | No |
| rx3700 series | Si | No |
| hx4700 series | Si | No |

Cuadro H.2: PDAs con Linux. Parte II.

| Modelo | Soporte Linux | Linux Nativo |
|---|---|---|
| **Fujitsu Siemens Loox** | | |
| h6300 series | Si | No |
| hx2100 series | No | No |
| hx2400 series | Si | No |
| hx2700 series | Si | No |
| rz1710 series | No | No |
| hw6500 series | No | No |
| hw6700 series | No | No |
| **HP Jornada** | | |
| 565/568 | Si | No |
| 620/660 & 680/690 | Si | No |
| 720 | Si | No |
| 820 | Si | No |
| **HTC Devices** | | |
| Wallaby (XDA) | Si | No |
| Himalaya (XDA II) | Si | No |
| Blueangel (XDA III) | Si | No |
| Universal (MDA Pro) | Si | No |
| Magician (MDA Mini) | No | No |
| **Sharp Zaurus** | | |
| SL-5500/SL-5600 | Si | Si |
| SL-6000 | Si | Si |
| SL-C750/SL-C760/SL-C860 | Si | Si |
| SL-C1000/SL-C3000/SL-C3100 | Si | Si |

Cuadro H.3: PDAs con Linux. Parte III.

# Bibliografía


[180-193] 180-1, F. P. SHA, secure hash standard, 1993.
URL `http://www.itl.nist.gov/fipspubs/fip180-1.htm`

[Al-Kalayi04] Al-Kalayi, A. K. M. Elliptic curve cryptography and smart cards, february 2004.

[anHuang03] an Huang, Y. y Lee, W. A cooperative intrusion detection system for ad hoc networks. págs. 135–147, 2003. Proceedings of the 1st. ACM workshop on security of ad-hoc and sensor networks. Fairfax, Virginia.

[Arbaugh] Arbaugh, W. A., Shankar, N., y Wan, Y. J. Your 802.11 wireless network has no clothes.
URL `citeseer.ist.psu.edu/arbaugh01your.html`

[Ash89] Ash, W. D., Blake, F. I., y Vanstone, S. A. *Low Complexity Normal Bases; Discrete Applied Mathematics*. 1989.

[Beachy96] Beachy, J. A. y Blair, W. D. *Abstract Algebra, Second Edition*. Waveland Press, 1996. Http://www.seguridata.com.

[Belingueres] Belingueres, G. Introducción a los criptosistemas de curva elíptica.
URL `http://www.geocities.com/belingueres`

[Brown00] Brown, M., Cheung, C., Hankerson, D., López, J., Kirkup, M., y Menezes, A. Pgp in constrained wireless devices. *En Proceedings of the 9th USENIX Securitu Symposium*. august 2000.

[Cagalj05] Cagalj, M., Capkun, S., y Hubaux, J.-P. Key agreement in peer-to-peer

wireless network. *En Proceedings of the IEEE (Special Issue on Security and Cryptography)*, págs. 1–11. 2005.

[Capkun03] Capkun, S., Butty, L., y Hubaux, J.-P. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):52–64, 2003.

[Charlap88] Charlap, L. S. y Robbins, D. P. An elementary introduction to elliptic curves. 1988. Center for Communications Research - Princeton.

[Diffie76] Diffie, W. y Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, november 1976.
URL `citeseer.ist.psu.edu/diffie76new.html`

[Frey94] Frey, G. y Rück, H. Aremak concerning m-divisibility and the discrete logartihm in the divisor class group of curves. 32:865–874, 1994.

[García03] García, J. M. G. *Tesis Doctoral Evaluación Paralela de Sumas de Puntos Relacionales Sobre Curvas Elípticas*. Tesis Doctoral, Centro de Investigación en Computación IPN, mar 2003.

[Hankerson04] Hankerson, D., Menezes, A. J., y Vanstone, S. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004.

[Hu02] Hu, Y.-C., Perrig, A., y Johnson, D. B. Ariadne: A secure on-demand routing protocol for ad hoc networks. *En Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking* (MobiCom 2002), págs. 12–23. sep 2002.
URL `citeseer.ist.psu.edu/article/hu02ariadne.html`

[iTomé04] i Tomé, S. B. y Ramiro, M. C. Gnupg implementation with elliptic curves, december 2004.

[Johnson01] Johnson, D., Menezes, A., y Vanstone, S. The elliptic curve digital signature algorithm (ecdsa). (Number 1):36–63, august 2001. Proceedings of the 1st. ACM workshop on security of ad-hoc and sensor networks. Fairfax, Virginia.

[Koblitz87] Koblitz, N. Elliptic curve cryptosystems. 48:203–209, 1987.

[Koblitz04] Koblitz, N. y Menezes, A. J. A survey of public-key cryptosystems. 46(4):599–634, 2004.

[Law03] Law, L., Menezes, A., Qu, M., Solinas, J., y Vanstone, S. A. An efficient protocol for authenticated key agreement. 28(2):119–134, march 2003.

[Lim98] Lim, C. H. y Lee, P. J. A study on the proposed korean digital signature algorithm. *En ASIACRYPT*, págs. 175–186. 1998.

[López05] López, M. J. L. *Criptografía y Seguridad en Computadores*, sept 2005. Http://wwwdi.ujaen.es/ mlucena/wiki/pmwiki.php?n=Main.LCripto.

[Memon05] Memon, N. y Ramkumar, M. An efficient key predistribution scheme for ad hoc network security. 23:611–621, march 2005.

[Menezes91] Menezes, A., Vanstone, S., y Okamoto, T. Reducing elliptic curve logarithms to logarithms in a finite field. *En STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, págs. 80–89. ACM Press, New York, NY, USA, 1991.

[Menezes96] Menezes, A. J., Oorsches, I. C., Paul, y Vanstone, S. A. *HandBook of Applied Cryptography*. CRC press, oct 1996. Http://www.seguridata.com.

[Pohlig78] Pohlig, S. y Hellman, M. An improved algorithm for computing logaritmo over gf(p) and its cryptographyc significance. 24:106–110, 1978.

[Pollard78] Pollard, J. Monte carlo index computation mod p. 32:918–924, 1978.

[Qizhi04] Qizhi, Q. y Qianxing, X. Research on elliptic curve cryptography. *En Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on Publication*, tomo 2, págs. 698–701. may 2004.

[Rivest75] Rivest, R. L., Shamir, A., y Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. 1975.

[Sanzgiri05] Sanzgiri, K., LaFlamel, D., Bridget, D., Levine, N. B., Shields, C., y Belding-Royer, E. M. Authenticated routing for ad hoc networks. 23(3):598–610, mar 2005.

[Semaev98] Semaev, I. Evaluation of discrete logarithm in a group of p-torsion points of an elliptic curve in characteristic p. 67:353–356, 1998.

[Shoup] Shoup, V. Conversions ntl.
URL `http://shoup.net/ntl`

[Silverman94] Silverman, J. *Advanced Topics in the Arithmetic of Elliptic Curves*, tomo 151 de *Graduate Texts in Mathematics*. Springer, 1994.

[Smart97] Smart, N. Announcement of an attack on the eclp for anomalous elliptic curves, 1997.

[Solinas00] Solinas, J. A. Efficient arithmetic on koblitz curves. 19:195–249, march 2000.

[Stevens90] Stevens, R. W. *Unix Network Programming*. Prentice Hall, 1990.

[Strangio05] Strangio, A. M. Efficient diffie-hellmann two-party key agreement protocols based on elliptic curves. *En SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, págs. 324–331. ACM Press, New York, NY, USA, 2005.

[Torii00] Torii, N. y Yokoyama, K. Elliptic curve cryptosystem. 36(2):140–146, dec 2000.

[Vigna04] Vigna, G., Gwalani, S., Srinivasan, K., Belding-Royer, E. M., y Kemmerer, R. A. An intrusion detection tool for aodv-based ad hoc wireless networks. *En ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, págs. 16–27. IEEE Computer Society, Washington, DC, USA, 2004.

[Wollinger04] Wollinger, T., Pelzl, J., Wittelsberger, V., y Paar, C. Elliptic and hyperelliptic curves on embedded microp. 3, august 2004.

[X9.6299] X9.62, A. The elliptic curve digital signature algorithm (ECDSA), 1999.

[X9.63rd] X9.63, A. Elliptic curve key agreement and transport protocols, Draft Standard.

[Yang02] Yang, H., Meng, X., y Lu, S. Self-organized network-layer security in mobile ad hoc networks. *En WiSE '02: Proceedings of the 3rd ACM workshop on Wireless security*, págs. 11–20. ACM Press, New York, NY, USA, 2002.

[Zhang03] Zhang, Y., Lee, W., y an Huang, Y. Intrusion detection techniques for mobile wireless networks. Volume 9, Issue 5:545–556, sep 2003.