UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE UN SISTEMA DE CIFRADO DE LLAVE PÚBLICA BASADO EN EL PROBLEMA DEL LOGARITMO DISCRETO PARA CURVAS ELÍPTICAS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A:

MOISÉS BAUTISTA OSORNO

DIRECTOR DE LA TESIS: DR. GERARDO VEGA HERNÁNDEZ

MÉXICO, D.F. 2002.

Para, por y de Olivia

Quiero externar mi agradecimiento a mis sinodales, quienes participaron en la revisión de este trabajo:

Dr. Alberto Alonso y Coria
Dr. Pablo Barrera Sánchez
Dr. Enrique Daltabuit Godas
Dr. Horacio Tapia Recillas
Dr. Gerardo Vega Hernández

Un agradecimiento especial para el Dr. Gerardo Vega Hernández por su ayuda, paciencia y sobre todo por compartir incondicionalmente su conocimiento. Asimismo, quiero expresar mi profunda gratitud al Dr. Neal Koblitz quien amablemente contestó a todas y cada una de mis dudas.

Por último, no puedo dejar de mencionar el generoso apoyo del Dr. Homero Ríos, así como el del Ing. Juvenal Guzmán.

Índice general

Prefacio

En la última década y lo que va de ésta, el manejo y transmisión electrónica de la información ha llegado a adquirir gran relevancia. Instituciones de gobierno, educativas, comerciales e incluso los hogares manejan grandes cantidades de datos mediante mecanismos electrónicos. Las ventajas de este tipo de manejo son muchas, entre otras, la capacidad de transmisión instantánea y acceso remoto; también es posible realizar operaciones financieras, comerciales y de servicios por una gran cantidad de personas de manera remota y simultánea. Sin embargo, con la revolución tecnológica alcanzada hasta hoy, los riesgos que este manejo trae consigo han aumentado. A diferencia de la documentación impresa, los datos que viajan y se almacenan por medios electrónicos como pueden ser el correo electrónico, telefonía, transmisiones de televisión, etcétera, pueden ser robados, manipulados y corrompidos desde una localidad distante. Por lo anterior, se han ideado diversos mecanismos para proteger el contenido de la información mientras ésta se almacena o viaja hacia su destino.

La búsqueda de seguridad en la información es casi tan vieja como la información misma. Siempre que la gente desarrolle nuevos métodos para almacenar, grabar o transmitir datos, es necesario desarrollar métodos que protejan la integridad y confidencialidad de los mismos. Dentro de los métodos que existen para tal fin están los criptosistemas. Un criptosistema mapea unidades de texto ordinario llamadas unidades de texto en claro (cada una consistiendo de una o más letras) en unidades de texto codificado llamadas unidades de texto cifrado. La seguridad que provee un criptosistema, se basa en el hecho de que al tratar de romper el cifrado se debe resolver un problema matemático del cual resulta *difícil* tener una solución. En este contexto, cuando se dice que un problema es *difícil,* lo que se quiere decir es que la solución a dicho problema tomará un tiempo muy grande en

términos prácticos.

Los criptosistemas se dividen básicamente en dos: los de clave secreta y los de clave pública. En los primeros se tiene una clave que nos permite cifrar y descifrar, mientras que para los segundos se emplea un par de claves. Ahora bien, de manera informal, un criptosistema de clave secreta es aquél en el cual se comparte "un secreto", en este caso el secreto es la clave. En un criptosistema de clave pública, una de las dos claves, la llamada clave pública, está al alcance de cualquiera, pero resultará prácticamente imposible deducir la otra clave (la clave privada) a partir del conocimiento de la clave pública o del texto cifrado.

Este trabajo trata sobre la implementación de un criptosistema de clave pública. El criptosistema basa su seguridad en la dificultad computacional de resolver el **Problema del Logaritmo Discreto para Curvas Elípticas (PLDCE)**. Actualmente este tipo de sistemas está ganando la aceptación a nivel mundial, debido a que el tamaño de las claves que utilizan es aproximadamente un sexto de aquél que emplean los mejores y más populares sistemas de cifrado y firma digital, como por ejemplo, DSA (*Digital Signature Algorithm*) y RSA, Koblitz *et al* [22]. Esto impacta directamente en el tipo de aplicaciones que se pueden beneficiar de esta tecnología, ya que se tienen ahorros sustanciales por ejemplo en el uso de memoria o espacio de almacenamiento, y en general en entornos donde los recursos son restringidos, como aplicaciones de telefonía celular, computadoras de bolsillo, tarjetas inteligentes, etcétera.

El objetivo principal de este trabajo es diseñar e implementar un esquema real de cifrado y firma digital de clave pública basado en el Problema del Logaritmo Discreto para Curvas Elípticas. Otro objetivo, no menos importante, es el de sentar un precedente para futuros desarrollos de este tipo de aplicaciones, y con esto motivar a todo aquél que se interese en general por la criptografía y en particular por la de clave pública.

Como resultado de este trabajo se generó una biblioteca de rutinas criptográficas, la cual cuenta con rutinas de generación de llaves, cifrado, descifrado, firma digital, verificación de firma, ensobretado y apertura de sobre. Para la implementación de un criptosistema basado en el PLDCE, son tres los principales componentes; la ecuación de una curva elíptica, un campo

finito sobre el que se define la curva y un punto racional de la curva sobre éste campo. Cabe señalar, que la curva, el campo y el punto con los que finalmente se desarrolló el criptosistema, pertenecen a la lista de sugerencias del Departamento de Comercio de E.U., las cuales se publican a través del Instituto Nacional de Estándares y Tecnología (NIST), FIPS 186-2 [4].

El trabajo consta de 4 capítulos y un anexo:

- Capítulo 1. Fundamentos Teóricos.
- Capítulo 2. Elementos de un sistema de cifrado.
- Capítulo 3. Implementación de un sistema de cifrado.
- Capítulo 4. Conclusiones y futuras líneas de trabajo.
- Anexo. Manual de uso de la biblioteca.

En el capítulo 1 se abordan los fundamentos de la teoría de curvas elípticas sobre campos finitos, esto es, el conocimiento matemático necesario para comprender y desarrollar un criptosistema basado en el PLDCE. En el capítulo 2, se describen los elementos o componentes necesarios para implemetar un sistema de cifrado seguro y eficiente, lo cual se hace generando instancias del PLDCE. El capítulo 3 describe las rutinas y algoritmos esenciales de la implementación, las cuales fueron desarrolladas en el lenguaje de programación C. El último capítulo describe los resultados obtenidos y las líneas de trabajo que se pudiesen derivar. Finalmente, el anexo que se incorpora contiene un manual que explica como utilizar las diferentes rutinas que conforman a la biblioteca.

Capítulo 1

Fundamentos Teóricos

El uso de la criptografía hoy en día resulta imprescindible en diversos ámbitos del quehacer humano, hasta hace un par de décadas su uso estaba reservado a aspectos militares y diplomáticos. La historia de la criptografía es muy antigua y fascinante, para una revisión histórica no técnica de la criptografía puede consultarse Kahn D. [16], esta obra es una de las referencias más completas en su tipo.

Un parteaguas vino en 1976 cuando dos investigadores de la universidad de Stanford, Whitfield Diffie y Martin Hellman [6], publicaron el artículo clásico "*New Directions in Cryptography*". Desde la aparición de éste artículo muchos criptosistemas de clave pública han sido propuestos. Cada uno de estos se basa en un problema matemático difícil de resolver. Formalemente no se ha probado que ninguno no pueda resolverse de una manera eficiente. Sin embargo, existe una creencia generalizada de ser muy difíciles porque durante varios años de investigación intensa, matemáticos y científicos de la computación no han podido encontrar hasta ahora algoritmos eficientes para resolverlos. Además, en la práctica, dichos problemas siguen siendo difíciles de resolver con la tecnología computacional actual. Mientras más tiempo se tome derivar la clave privada con el mejor algoritmo conocido para un cierto problema, más seguro es el criptosistema basado en ese problema.

Actualmente, existen sólamente tres clases de criptosistemas de clave pública, Menezes *et al* [29], los cuales son considerados seguros y eficientes. Están clasificados de acuerdo al problema matemático sobre el que están basados.

- Sistemas de factorización entera.
- Sistemas de logaritmo discreto.
- Sistemas de logaritmo discreto para curvas elípticas.

Las curvas elípticas son curvas algebraicas que han sido estudiadas por muchos matemáticos desde el siglo XVII. En 1985 Victor Miller [27] y Neal Koblitz [17] de manera independiente propusieron criptosistemas de clave pública utilizando el conjunto de puntos de una curva elíptica definida sobre un campo finito.

Los criptosistemas de curvas elípticas ofrecen la mayor seguridad por bit que cualquier criptosistema de clave pública porque actualmente parece que el PLDCE resulta más difícil de resolver que los problemas de factorización entera y del logaritmo discreto, Koblitz *et al* [22]. Debido a esto es posible utilizar una curva elíptica de tal manera que los tamaños de las claves sean más pequeñas, sin sacrificar el nivel de seguridad.

Por lo anterior, los criptosistemas basados en curvas elípticas resultan adecuados para emplearse en aplicaciones donde los dispositivos de almacenamiento y memoria están restringidos, como pueden ser el uso de tarjetas inteligentes, telefonía celular, computadoras de bolsillo, etcétera. Empresas como Certicom Corporation han implementado este tipo de sistemas con fines comerciales, se puede visitar el sitio de esta empresa en http://www.certicom.com/, en esta dirección podrán encontrarse diversos aspectos de la criptografía de curvas elípticas.

En este capítulo se describen los fundamentos teóricos sobre los que se construye un sistema de cifrado basado en el problema de hallar el logaritmo discreto para curvas elípticas. El desarrollo inicia con una introducción de términos y conceptos básicos que deberán precisarse. La exposición se enmarca fundamentalmente en tres aspectos , el primero hace referencia a los elementos necesarios para abordar la teoría de las curvas elípticas, a saber, los grupos, anillos y campos finitos. El segundo trata lo referente a las curvas elípticas sobre campos finitos, señalando aquellos aspectos que resultan de utilidad práctica. La parte final del capítulo se dedica a examinar las posibilidades de implementación con los elementos descritos previamente para el caso particular de los campos finitos de característica

2. Este trabajo no pretende ser una referencia teórica sobre los aspectos matemáticos relacionados con la criptografía de curvas elípticas porque el objetivo principal es desarrollar una implementación real de cifrado. Es por ello, que los resultados o teoremas que aquí se enuncian no serán probados, para un tratamiento más exhaustivo sobre grupos, campos y curvas elípticas se proporcionan las referencias correspondientes.

1.1. Terminología básica y conceptos

A continuación se enuncian algunos conceptos y definiciones básicas las cuales serán manejadas a lo largo del trabajo. La mayor parte del material para este apartado se basó en Menezes *et al* [29] y para la parte de complejidad en Johnson [15].

Participantes en una comunicación

Dentro de un esquema de transmisión-recepción de información, existen diferentes elementos que lo componen, los principales son los siguientes:

1. Una *entidad* es algo o alguien que envía, recibe o manipula información. Una entidad puede ser una persona, una computadora, etc.

2. Un *emisor* es una entidad dentro de una comunicación de dos o más entidades la cual es transmisor de información.

3. Un *receptor* es una entidad dentro de una comunicación de dos o más entidades la cual recibe la información.

4. Un *adversario* es una entidad que intenta romper el esquema de seguridad que se ha establecido entre los emisores y los receptores.

Complejidad

La tarea primordial de la teoría de la complejidad es la de clasificar problemas computacionales en función de los recursos necesarios para resolverlos. Los recursos que pueden ser evaluados son el tiempo, el espacio de almacenamiento, el número de procesadores usados, etcétera. En general lo que se mide es el tiempo y el espacio de almacenamiento que requieren los problemas para ser resueltos.

Definición 1.1.1. Un *algoritmo* es un procedimiento computacional bien definido que toma como entrada una variable y se detiene produciendo una salida.
Por lo general, resulta de interés encontrar el algoritmo más eficiente (el más rápido y mejor estructurado) para resolver un problema computacional específico.

Definición 1.1.2. El *tamaño de la entrada* para un algoritmo, es el número total de bits que se requiere para representar la entrada en forma binaria utilizando un esquema de codificación apropiado.

Definición 1.1.3. El *tiempo de ejecución* de un algoritmo cualquiera, es el número de "pasos" realizados.

Es frecuente considerar un paso como una comparación, una instrucción de máquina ó un ciclo de reloj.

Definición 1.1.4. El *tiempo de ejecución del peor caso* para un algoritmo es la cota superior sobre el tiempo de ejecución para cualquier entrada.

Definición 1.1.5. Un algoritmo de *tiempo polinomial* es aquel para el cual el tiempo de ejecución para el peor caso es del orden de $O(n^k)$, donde n es el tamaño de la entrada y k es una constante. Cualquier algoritmo cuyo tiempo de ejecución no pueda acotarse se le llama *un algoritmo de tiempo exponencial.*

Definición 1.1.6. Un algoritmo de *tiempo subexponencial* es aquel en el que su peor tiempo de ejecución es del orden de $e^{o(n)}$, donde n es el tamaño de la entrada.

Criptología

La criptología se encarga del estudio y uso de métodos destinados a ocultar información. Esta se divide en criptografía y criptoanálisis.

1. *Criptografía* es el estudio de técnicas relacionadas con aspectos de seguridad en la información tales como confidencialidad, integridad en los datos, verificación de la entidad y verificación del origen de los datos.

2. *Criptoanálisis* es el estudio de técnicas para intentar frustrar las técnicas criptográficas y de manera más general los servicios de seguridad en la información.

3. Un *criptoanalista* es alguien que trabaja en criptoanálisis.

4. Un *criptógrafo* es alguien que trabaja en criptografía.

5. Un *algoritmo criptográfico* es un procedimiento computacional que provee servicios de seguridad en la información.

6. *Criptosistema* es un término que denota a un conjunto de algoritmos criptográficos.

Objetivos criptográficos

Cuando se habla de seguridad en la información se deberán cumplir los siguientes objetivos:

1. *Confidencialidad.* Esta permite el acceso a la información sólamente a quienes cuenten con la autorización debida.

2. *Integridad de la información.* Esta detecta la alteración no autorizada de la información.

3. *Autenticación.* Se relaciona con la identificación y es aplicable tanto a entidades como a la información.

4. *No-repudio.* Establece un compromiso entre los participantes de una comunicación. Por ejemplo, auxilia en disputas originadas a partir de que una entidad realiza o ejecuta ciertas acciones y posteriormente las niega.

Dominios e Imágenes de Cifrado

Para hablar de las funciones de cifrado-descifrado es necesario conocer los conjuntos sobre los que se definen.

1. *El alfabeto de definición* es un conjunto finito $\mathcal{A}$ de símblos. Por ejemplo, el alfabeto decimal, $\mathcal{A} = \{0, 1, ..., 9\}$, otro ejemplo sería el alfabeto castellano.

2. *El espacio de mensajes* es un conjunto $\mathcal{M}$ integrado por cadenas de símbolos que se forman a partir de un alfabeto de definición. A un elemento de $\mathcal{M}$ se le denomina mensaje en *texto plano o texto claro.*

3. *El espacio de texto cifrado* es un conjunto $\mathcal{C}$ integrado por cadenas de símbolos que se forman a partir de un alfabeto de definición, el cual podría diferir del alfabeto de definición para $\mathcal{M}$. A un elemento en $\mathcal{C}$ se le llama *texto cifrado.*

Funciones de Cifrado y Descifrado

Ahora se enuncian las definiciones de lo que son las funciones de cifrado y descifrado.

1. *El espacio de claves* es un conjunto no vacío $\mathcal{K}$ de cadenas (posiblemente binarias) de longitud finita. A cualquier elemento de $\mathcal{K}$ se le llama *clave.* Los elementos que están en $\mathcal{K}$ son cadenas de longitud finita.

2. Para cada elemento $e \in \mathcal{K}$ se determina una función inyectiva única que va de $\mathcal{M}$ a $\mathcal{C}$, la cual se denota por E_e. A la función $E_e : \mathcal{M} \to \mathcal{C}$ se le llama una *función o transformación de cifrado.*

3. Para cada $d \in \mathcal{K}$, D_d denotará la función inyectiva que va de $E_e(\mathcal{M}) \subseteq \mathcal{C}$ a $\mathcal{M}$. A la relación $D_d : E_e(\mathcal{M}) \to \mathcal{M}$ se le llama *función o transformación de descifrado.*

4. El proceso de aplicar la función E_e a un mensaje $m \in \mathcal{M}$ se conoce como *el cifrado* de m.

5. La aplicación de la transformación D_d a un texto cifrado $c \in E_e(\mathcal{M}) \subseteq \mathcal{C}$ se le conoce como *el descifrado* de c.

6. *Un sistema de cifrado* consiste de un conjunto $\{E_e : e \in \mathcal{K}\}$ de funciones de cifrado y el correspondiente conjunto $\{D_d : d \in \mathcal{K}\}$ de funciones de descifrado con la propiedad de que para cada $e \in K$ existe una clave única $d \in \mathcal{K}$ tal que $D_d = E_e^{-1}$, o lo que es lo mismo $D_d(E_e(m)) = m$ para toda $m \in \mathcal{M}$.

7. A las claves d y e que se mencionan en el inciso anterior se les llama *el par de claves* y se denota por (d, e).

Tipos de sistemas de cifrado

1. Un sistema de cifrado, el cual consiste de los conjuntos de transformación para cifrar y descifrar $\{E_e : e \in \mathcal{K}\}$ y $\{D_d : d \in \mathcal{K}\}$ respectivamente, se dice que es un *sistema asimétrico* o *de clave pública* si para cada par de claves (e, d), con $e \neq d$, la clave e está disponible públicamente, mientras que la clave d es mantenida en privado. Para que el esquema sea seguro, no debe ser computacionalmente factible determinar d a partir de e.

2. Un sistema de cifrado, con los conjuntos de transformación para cifrar y descifrar $\{E_e : e \in \mathcal{K}\}$ y $\{D_d : d \in \mathcal{K}\}$ respectivamente, se dice que es un *sistema simétrico* o *de clave secreta* si existe únicamente una clave e, la cual nos permite cifrar y descifrar, es decir, $d = e$.

Objetivos del adversario

Uno de los objetivos de un adversario consiste en recuperar texto plano a partir del cifrado de manera sistemática. Si el adversario logra su cometido se dice que el sistema ha sido *roto*. Un objetivo que resulta más ambicioso es el de recuperar la clave secreta, para el caso de criptosistemas simétricos; y en el caso de los asimétricos la clave privada; si se logra esto, se dice que el sistema ha sido *totalmente roto*, ya que el adversario podrá descifrar cualquier documento e incluso generar sus propios mensajes cifrados.

La puesta en práctica de la criptografía de clave pública

Los sistemas de cifrado/descifrado más rápidos son los de clave secreta, sin embargo la debilidad que tienen es que la clave que utilizan debe ser compartida por quienes participan en una comunicación basada en este tipo de sistemas, en este sentido se debe asegurar que la clave no sea robada por un adversario al momento de compartirla, a este último problema se le conoce como "el problema de la distribución de la clave".

Por otra parte, los sistemas asimétricos resuelven el problema mencionado en el párrafo anterior, ya que por definición una de las claves es pública y es precisamente con ésta con la que se cifran los mensajes y únicamente el poseedor de la clave privada correspondiente podrá descifrar dichos mensajes. Sin embargo, estos sistemas a su vez tienen un problema llamado "el problema

de la certificación de la clave pública". Para resolver este último se emplea lo que se conoce como *Infraestructura de Clave Pública* (PKI por sus siglas en Inglés). Como se verá con más claridad en el capítulo 3, el objetivo de una PKI es proporcionar *confianza* entre los participantes de una comunicación basada en criptografía asimétrica.

En la práctica se utiliza una combinación de ambos tipos de sistemas de cifrado, por una parte aprovechando la velocidad de los sistemas simétricos y por el otro la posibilidad de usar un canal de transmisión inseguro (suceptible de ser interceptado) empleando los sistemas asimétricos.

1.2. Grupos, anillos y campos

El orden que se seguirá para esta sección es ascendente con respecto a las riqueza algebraica de las estructuras que se presentan, se inicia con la definición y propiedades básicas de los grupos, se continúa con los anillos y se termina con los campos.

1.2.1. Grupos

Definición 1.2.1. Un conjunto no vacío de elementos $\mathcal{G}$ se dice que forma *un grupo* si en $\mathcal{G}$ está definida una operación binaria "$*$" y es tal que:

1. Si $a, b \in \mathcal{G}$, implica que $a * b \in \mathcal{G}$ (cerradura).
2. Si $a, b, c \in \mathcal{G}$, implica que $a * (b * c) = (a * b) * c$ (ley asociativa).
3. Existe un elemento $e \in \mathcal{G}$ tal que $a * e = e * a = a$ para todo $a \in \mathcal{G}$ (existencia del neutro).
4. Para todo $a \in \mathcal{G}$ existe un elemento $a^{-1} \in \mathcal{G}$ tal que $a * a^{-1} = a^{-1} * a = e$ (existencia del inverso).

Como se verá más adelante, los puntos racionales de una curva elíptica determinan un grupo a partir de una operación que se define entre ellos. Esta última propiedad tiene una gran trascendencia para la construcción de criptosistemas que se basan en el PLDCE.

Un tipo especial de grupo $\mathcal{G}$ es aquel en el cual la operación definida además de cumplir las propiedades mencionadas también es conmutativa, es decir, $a * b = b * a$, para todo $a, b \in \mathcal{G}$. A este tipo de grupo se le llama *abeliano* o *conmutativo*.

Una propiedad importante de un grupo $\mathcal{G}$ es el número de elementos con que cuenta. A esta propiedad se le conoce como *el orden del grupo* y se le denota por $o(\mathcal{G})$. Cuando el grupo tiene orden finito, se dice que el grupo es finito.

Ejemplo 1.2.1. Sea $\mathbb{Z} = \{0, \pm 1, \pm 2, ...\}$ el conjunto de los enteros con la operación "$*$" definida como la suma usual entre enteros, es decir, $a*b = a+b$. Este grupo es abeliano e infinito, donde el número 0 es el elemento neutro y a^{-1} es $-a$.

Es frecuente encontrar en la literatura sobre grupos la notación con el uso de exponentes, por ejemplo, se dice que el elemento $a^i \in \mathcal{G}$, con esto lo que se quiere decir es que la operación "$*$" definida sobre G se aplica $i-1$ veces, es decir, $a^2 = a * a$, $a^3 = a * a * a$, etcétera.

Definición 1.2.2. Un grupo $\mathcal{G}$ es *cíclico* si existe un elemento $\alpha \in \mathcal{G}$ tal que para cada $b \in \mathcal{G}$ existe un número entero i con $b = \alpha^i$. Al elemento α se le llama un *generador* de $\mathcal{G}$.

Definición 1.2.3. Un subconjunto $\mathcal{H}$ de un grupo $\mathcal{G}$ se dice que es un *subgrupo* de $\mathcal{G}$ si respecto a la operación definida en $\mathcal{G}$, él mismo forma un grupo.

Ejemplo 1.2.2. Sea nuevamente $\mathcal{G} = \mathbb{Z}$ el conjunto de los enteros, con la operación "$*$" definida como la suma usual entre enteros. Sea $\mathcal{H} = 2\mathbb{Z} = \{0, \pm 2, \pm 4, ...\}$ el subconjunto de $\mathcal{G}$ definido por los enteros pares y el cero; con la misma operación "$*$" definida en $\mathcal{G}$. El subconjunto $\mathcal{H}$ es un subgrupo de $\mathcal{G}$.

El siguiente es un tipo especial de subgrupo, el cual es importante para este trabajo.

Ejemplo 1.2.3. Sea $\mathcal{G}$ un grupo cualquiera y $a \in \mathcal{G}$. Sea $\langle a \rangle = \{a^i | \, i = 0, \pm 1, \pm 2, \ldots\}$. Se puede mostrar fácilmente que el conjunto $\langle a \rangle$ es un subgrupo de $\mathcal{G}$, al cual se le conoce como *el subgrupo cíclico generado por* a.

Definición 1.2.4. Si $\mathcal{G}$ es un grupo y $a \in \mathcal{G}$, el orden de a es el mínimo entero positivo m tal que $a^m = e$, es decir, $o(\langle a \rangle) = m$.

Ahora se enuncia un teorema importante de la teoría de grupos, a saber, el teorema de Lagrange.

Teorema 1.2.1. *Si $\mathcal{G}$ es un grupo finito y $\mathcal{H}$ es un subgrupo de $\mathcal{G}$ entonces $o(\mathcal{H})$ es un divisor de $o(\mathcal{G})$.*

Como una consecuencia directa del teorema anterior se tiene el siguiente resultado.

Corolario 1.2.2. *Si $\mathcal{G}$ es un grupo finito cuyo orden es un número primo p, entonces $\mathcal{G}$ es un grupo cíclico.*

Si se desea ahondar sobre la teoría de grupos y anillos, el lector puede consultar, por ejemplo Fraleigh [7] y Herstein [13].

1.2.2. Anillos

A diferencia de los grupos, en los anillos se definen dos operaciones, a las cuales comúnmente se les identifica con la suma y el producto.

Definición 1.2.5. Un conjunto no vacío $\mathcal{R}$ se dice que es *un anillo asociativo* (o simplemente anillo), si en $\mathcal{R}$ están definidas dos operaciones, denotadas por "+" y "$*$" tales que:

1. $(\mathcal{R}, +)$ es un grupo conmutativo.
2. $\forall\ a, b \in \mathcal{R}$, $a * b \in \mathcal{R}$ (cerradura).
3. $\forall\ a, b, c \in \mathcal{R}$, $a * (b * c) = (a * b) * c$ (asociatividad).
4. $\forall\ a, b, c \in \mathcal{R}$, $a * (b + c) = a * b + a * c$ y $(b + c) * a = b * a + c * a$ (distributividad).

Las propiedades 2 y 3 indican que la operación producto es cerrada y asociativa. La última propiedad relaciona ambas operaciones y comúnmente se dice que el producto se distribuye respecto a la suma. Es posible que exista un elemento $1 \in \mathcal{R}$, con la propiedad de que $a * 1 = 1 * a = a$ para todo $a \in \mathcal{R}$; si tal elemento existe se dice que $\mathcal{R}$ es *un anillo con elemento unitario.* Por otro lado, si el producto en $\mathcal{R}$ tiene la propiedad de que $a * b = b * a$ para todo $a, b \in \mathcal{R}$, entonces se dice que $\mathcal{R}$ es un *anillo conmutativo con elemento unitario.*

Ejemplo 1.2.4. Sea $\mathcal{R} = \mathbb{Z}$, el conjunto de los enteros con las operaciones suma y producto habituales, $\mathcal{R}$ resulta ser un anillo conmutativo con elemento unitario.

Definición 1.2.6. Un anillo se dice que es *un anillo con división* si sus elementos diferentes de cero forman un grupo bajo el producto.

La definición anterior permite definir lo que es un campo:

Definición 1.2.7. Un *campo* es un anillo conmutativo con división.

1.2.3. Campos

La sección anterior termina con la definición de lo que es un campo, a saber, un campo es un anillo conmutativo en el que podemos dividir por cualquier elemento diferente de cero. Si el número de elementos en el campo es finito, se dice que es un campo finito. La mayor parte de los resultados que aquí se presentan se refieren a campos finitos, ya que sobre este tipo de campos se realiza la implementación del criptosistema que es materia de este trabajo. De aquí que resulta importante centrar la atención sobre los campos finitos, la representación de sus elementos y la implementación de la aritmética sobre ellos. Un buen trabajo sobre estos aspectos repercute directamente en la eficiencia del criptosistema.

Existe un teorema muy fuerte e importante, el cual dice que dado un campo finito $\mathbb{F}$, el número de elementos de $\mathbb{F}$ es de la forma p^n, para algún número primo p y cierto entero $n \geq 1$, Lidl y Niederreiter [24], MacWilliams y Sloane [26]. Entonces, el número de elementos de un campo finito $\mathbb{F}$ es potencia de algún número primo p, es decir, la cardinalidad de un campo finito $\mathbb{F}$ es p^n, con $n \geq 1$, al número primo p se le conoce como *la característica del campo* $\mathbb{F}$. A los campos finitos se les denota por $\mathbb{F}_{p^n}$ o $GF(p^n)$, la última notación proviene de *Galois Fields*. Serán de interés para este trabajo los campos del tipo $\mathbb{F}_{2^n}$ o $GF(2^n)$, es decir, aquéllos que tengan característica 2.

Ejemplo 1.2.5. El campo $\mathbb{F}_2 = \{0, 1\}$ es el campo finito más pequeño. Las tablas de suma y producto son las siguientes.

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

Definición 1.2.8. Un subconjunto $\mathbb{K}$ de un campo $\mathbb{F}$ se dice que es un subcampo de $\mathbb{F}$, si bajo las mismas operaciones definidas en $\mathbb{F}$, $\mathbb{K}$ resulta ser también un campo.

Definición 1.2.9. Se dice que un campo $\mathbb{F}$ es un campo de extensión de $\mathbb{K}$, siempre que $\mathbb{K}$ sea un subcampo de $\mathbb{F}$.

Si $\mathbb{F}$ es una extensión de $\mathbb{K}$, entonces, bajo las operaciones definidas en $\mathbb{K}$, el campo $\mathbb{F}$ puede verse como un espacio vectorial sobre $\mathbb{K}$, Herstein [13]. En este sentido, se puede hablar de dependencia lineal, dimensión, bases, etcétera, en $\mathbb{F}$ respecto a $\mathbb{K}$.

Ejemplo 1.2.6. Considérese el campo $\mathbb{C}$ de los números complejos, el cual contiene al campo $\mathbb{R}$ de los números reales. El campo $\mathbb{C}$ puede verse como un espacio vectorial sobre $\mathbb{R}$, la dimensión del espacio vectorial es 2 y una base sería $\{1, i\}$. Para aclarar ideas, los elementos en $\mathbb{C}$ son los "vectores", mientras que los elementos en $\mathbb{R}$ son los escalares. Entonces, la suma vectorial de $u, v \in \mathbb{C}$ es la suma de elementos en $\mathbb{C}$. Asimismo se define el producto de un escalar $\alpha \in \mathbb{R}$ por un vector $v \in \mathbb{C}$, como el producto αv, visto también como el producto de elementos en el campo $\mathbb{C}$. Con estas operaciones, todos los axiomas que definen al campo $\mathbb{C}$ como un espacio vectorial sobre el campo $\mathbb{R}$ se cumplen.

Definición 1.2.10. El *anillo de polinomios* sobre un campo $\mathbb{F}$, denotado por $\mathbb{F}[x]$, consiste de todas las sumas finitas de productos de potencias de x con coeficientes en $\mathbb{F}$. La suma de dos elementos en $\mathbb{F}[x]$ se realiza sumando los respectivos coeficientes, mientras que la multiplicación utiliza la ley distributiva en $\mathbb{F}$ y las reglas $ax = xa$ para toda $a \in \mathbb{F}$ y $x^k x^l = x^{k+l}$ para todos los números naturales k y l.

Definición 1.2.11. Sea $f = a_0 + a_1 x + \cdots + a_n x^n \in \mathbb{F}[x]$, si $f \neq 0$ y $a_n \neq 0$, entonces el grado de f es n.

Definición 1.2.12. Un campo $\mathbb{F}$ se dice que es *algebraicamente cerrado* si para todo polinomio $f \in \mathbb{F}[x]$, siendo el grado de $f \geq 1$, f tiene una raíz en $\mathbb{F}$.

Por ejemplo, el campo $\mathbb{R}$ de los números reales no es algebraicamente cerrado, porque el polinomio $x^2 + 1$ no tiene raíces en $\mathbb{R}$. En contraste, el campo de los números complejos $\mathbb{C}$ es algebraicamente cerrado, lo cual

se sabe por el *teorema fundamental del álgebra*, Herstein [13]. El teorema fundamental del álgebra dice que todo ponomio de grado n con coeficientes en $\mathbb{C}$ tiene exactamente n raíces, pudiendo ser múltiples algunas.

Definición 1.2.13. La *cerradura algebraica* de un campo $\mathbb{K}$ es una extensión de $\mathbb{K}$ la cual es algebraicamente cerrada. La cerradura algebraica de $\mathbb{K}$ se denota por $\overline{\mathbb{K}}$.

Ejemplo 1.2.7. La cerradura algebraica del campo de los números reales $\mathbb{R}$ es el campo de los números complejos $\mathbb{C}$.

Definición 1.2.14. Sea $\mathbb{F}$ un campo finito y sea $f \in \mathbb{F}[x]$. Una extensión finita $\mathbb{E}$ de $\mathbb{F}$ se dice que es un campo de descomposición de f sobre $\mathbb{F}$ si f puede ser descompuesto en un producto de factores lineales sobre $\mathbb{E}$, pero no así en ningún subcampo propio de $\mathbb{E}$.

Definición 1.2.15. Se dice que un polinomio π con coeficientes en un campo $\mathbb{F}$ es irreducible en $\mathbb{F}$, si no se puede factorizar como el producto de dos polinomios de grado menor a π con coeficientes en $\mathbb{F}$. También se dice que π no tiene raíces en $\mathbb{F}$.

Los polinomios irreducibles son el equivalente en $\mathbb{F}[x]$ a los números primos para los enteros.

Teorema 1.2.3. *Sea π un polinomio irreducible en $\mathbb{F}_p[x]$ de grado n. Entonces el conjunto de todos los polinomios en x de grado menor a n y coeficientes en $\mathbb{F}_p$, con operaciones de suma y producto entre polinomios en $\mathbb{F}_p[x]$ realizadas módulo π, forma un campo de orden p^n. A este campo se le denota por $\mathbb{F}_{p^n} = \mathbb{F}_p[x]/ < \pi >$.*

El campo $\mathbb{F}_{p^n}$ está formado por todos los polinomios en x de grado menor que n, con coeficientes en $\mathbb{F}_p$. Un elemento en $\mathbb{F}_{p^n}$ es de la forma $a_0 + a_1x + ... + a_{n-1}x^{n-1}$, donde $a_i \in \mathbb{F}_p$.

Ejemplo 1.2.8. Tómese al campo $\mathbb{F}_2 = \{0, 1\}$ y al polinomio irreducible en $\mathbb{F}_2[x]$, $\pi(x) = x^2 + x + 1$. Por el teorema 1.2.3, $\mathbb{F}_2[x]/ < (x^2 + x + 1) >$ forma un campo con 4 elementos, es decir $\mathbb{F}_{2^2}$ o $GF(4)$. Se presentan las tablas de suma y producto para el campo.

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$*$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Las suma y producto en $\mathbb{F}_2[x]/ < (x^2+x+1) >$

Para el caso de la tabla de multiplicar, se observa que los elementos diferentes de cero forman un grupo cíclico, este grupo se denota como $\mathbb{F}_{2^2}^* = \{1, x, x+1\}$. Cuando se trata de generar un campo más grande, existen tablas de polinomios irreducibles para construir campos, estas tablas se pueden encontrar en Zierler [40] y [41] y Zivkovic [42].

El campo $\mathbb{F}_{p^n} = \mathbb{F}_p[x]/ < \pi >$ (para cierto polinomio irreducible π con coeficientes en $\mathbb{F}_p$) es una extensión de $\mathbb{F}_p$, por lo cual es posible ver a la extensión como un espacio vectorial sobre el campo base. Una base es $\{1, x, x^2 \ldots, x^{n-1}\}$.

Ejemplo 1.2.9. El campo $\mathbb{F}_2[x]/ < (x^2+x+1) >$ que tiene cuatro elementos puede verse como espacio vectorial sobre $\mathbb{F}_2$ de la siguiente manera. Los elementos en $\mathbb{F}_2[x]/ < (x^2+x+1) >$ pueden verse como parejas ordenadas $v = (e_1, e_2)$ con $e_1, e_2 \in \mathbb{F}_2 \times \mathbb{F}_2$, porque cada una representa la combinación lineal respecto a la base. Ahora bien, los elementos del campo son $\{0, 1, x, x+1\}$ los cuales pueden ser vistos como el conjunto de tuplas $\{(0,0), (1,0), (0,1), (1,1)\}$ respectivamente. Entonces, $\{(1,0), (0,1)\} = \{1, x\}$, es una base porque no existen $a_0, a_1 \in \mathbb{F}_2$ tales que $a_0(1,0) + a_1(0,1) = (0,0)$ y $(a_0, a_1) \neq (0,0)$ y este conjunto genera a $\mathbb{F}_2[x]/ < (x^2+x+1) >$, ya que

$$\begin{aligned} 0 = (0,0) &= 0(1,0) + 0(0,1), \\ 1 = (1,0) &= 1(1,0) + 0(0,1), \\ x = (0,1) &= 0(1,0) + 1(0,1), \\ x+1 = (1,1) &= 1(1,0) + 1(0,1). \end{aligned}$$

De lo anterior se observa que $\mathbb{F}_2[x]/ < (x^2+x+1) >$ es un espacio vectorial sobre $\mathbb{F}_2$, la dimensión del espacio vectorial es 2.

Si $\mathbb{F}_{p^n}$ es visto como un espacio vectorial sobre el campo base $\mathbb{F}_p$, entonces todos los elementos del campo estarán representados como una combinación

lineal de cualquier base. En la práctica, la mayoría de las aplicaciones utilizan dos tipos de bases; normal y estándar.

En una base *estándar* o polinomial los elementos del campo se representan como polinomios de la forma $a_0 + a_1 + ... + a_{n-1}x^{n-1}$, donde $a_i \in \mathbb{F}_p$, esta representación es precisamente la que se señaló en el teorema 1.2.3. Como se ha dicho, las operaciones en el campo consisten de operaciones entre polinomios. A un producto entre polinomios le sigue inmediatamente una reducción módulo el polinomio irreducible de grado n. Para este trabajo se usarán bases del tipo estándar, por lo cual se centrará la atención en éstas. Respecto a las representaciones con bases normales basta decir que se orientan más hacia aplicaciones de hardware y las mejores bases de este tipo son las llamadas bases normales óptimas (**BNO**), sin embargo, las implementaciones más rápidas en software que se han reportado, Schroeppel [33] y López y Dahab [25], utilizan bases del tipo estándar.

Una vez que se ha decidido utilizar bases del tipo estándar, resulta necesario hallar un cierto polinomio irreducible para realizar la aritmética en el campo que se haya elegido.

1.3. Curvas elípticas

Como se ha dicho, las curvas elípticas han sido ampliamente estudiadas durante mucho tiempo, en áreas como variable compleja, teoría de números, geometría algebraica y existe una inmensa cantidad de literatura sobre el tema, algunas referencias clásicas son Semple J. G. y Roth L. [34], Walker R. J. [38] y Fulton W. [8], otras buenas referencias son por ejemplo Housemöller [14], Koblitz [21] y Silverman [35]. Estas también figuran de manera prominente en la prueba reciente del último Teorema de Fermat. Originalmente su estudio fue puramente teórico, pero recientemente han sido utilizadas en la creación de algoritmos de factorización de enteros, pruebas de primalidad y por supuesto en criptografía de clave pública.

Una de las razones principales por las cuales surge el interés de basar los sistemas de cifrado sobre curvas elípticas obedece a que estas curvas son fuente de un considerable número de grupos abelianos.

Como se verá a continuación, el conjunto de puntos racionales de una curva elíptica tienen estructura de grupo abeliano a partir de que se define una cierta operación entre ellos. Usualmente cuando se tiene un grupo abeliano la operación se denota por "+" y se le llama "suma".

Una vez que se haya definido dicha operación entre los puntos que están sobre una curva elíptica, estaremos en condiciones de enunciar el PLDCE.

Definición 1.3.1. Una curva elíptica E sobre un campo $\mathbb{F}$ es una curva que está dada por una ecuación (llamada forma normal de Weierstrass) de la forma

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_i \in \mathbb{F},$$

unión un elemento denotado por $\mathcal{O}$ (del cual se justificará su presencia enseguida) y donde las siguientes dos ecuaciones

$$a_1y = 3x^2 + 2a_2x + a_4, \quad 2y + a_1x + a_3 = 0$$

que representan las derivadas parciales respecto a x y y respectivamente, no pueden simultáneamente ser satisfechas por ningún punto (x, y) sobre la curva E.

Se denotará por $E(\mathbb{F})$ al conjunto de puntos $(x, y) \in \mathbb{F}^2$ que satisfacen a esta ecuación, junto con un "punto al infinito" denotado por $\mathcal{O}$. Si $\mathbb{K}$ es una extensión de $\mathbb{F}$, entonces $E(\mathbb{K})$ denota al conjunto de parejas $(x, y) \in \mathbb{K}^2$ que satisfacen la ecuación mencionada.

El llamado **j-invariante de la forma normal de Weierstrass** clasifica al conjunto de curvas elípticas y resulta útil para descartar dentro de todo el conjunto de curvas, muchas que no son útiles para los propósitos de generar instancias del PLDCE que permitan evitar los principales ataques conocidos a criptosistemas que basan su seguridad en la dificultad de resolver el PLDCE. El **j-invariante de la forma normal de Weierstrass** se define en términos del **discriminante de la forma normal de Weierstrass**.

El **discriminante de la forma normal de Weierstrass** es

$$\Delta = -b_2^2 b_8 - 8b_4^3 - 27b_6^2 + 9b_2 b_4 b_6.$$

Donde $b_2 = a_1^2 + 4a_2$, $b_4 = a_1 a_3 + 2a_4$, $b_6 = a_3^2 + 4a_6$ y $b_8 = a_1^2 a_6 - a_1 a_3 a_4 + 4a_2 a_6 + a_2 a_3^2 - a_4^2$.

Si $\Delta \neq 0$, **el j-invariante** de la forma normal de Wierestrass se define por $j(E) = \frac{a_1^{12}}{\Delta}$.

De esta manera las curvas se clasifican en dos tipos:

1. si $j(E) = 0$, entonces $a_1 = 0$, la curva recibe el nombre de *supersingular*.
2. si $j(E) \neq 0$, entonces $a_1 \neq 0$, la curva recibe el nombre de *no-supersingular*.

Es a partir de esta clasificación que las curvas que serán apropiadas son las no-supersingulares, debido a que por el momento, para un tipo especial de estas curvas los mejores ataques resultan imprácticos (Koblitz [18]), de esto se hablará más adelante en este capítulo.

1.4. El grupo de puntos racionales sobre una curva elíptica

Un punto racional sobre una curva definida por una ecuación $f(x, y) = 0$ sobre un campo $\mathbb{F}$ es un punto (x, y) en $\mathbb{F}^2$. Por ejemplo, un punto racional en el campo $\mathbb{Q}$ de los números racionales, es un punto (x, y) que satisface la ecuación mencionada, donde (x, y) está en $\mathbb{Q}^2$.

Ahora se mostrará como es la ecuación de una curva elíptica E definida para campos que no tienen característica 2 o 3 y aquéllos de característica 2. El objetivo es obtener la forma más simple de la ecuación de la forma normal de Weierstrass. Los correspondientes cambios de variable serán dados para cada caso.

Hablamos de las curvas definidas para campos que no tienen característica 2 o 3 porque como se verá enseguida, resulta más simple explicar sobre este

tipo de campos cómo se define la operación sobre la que se construye la estructura de grupo entre los puntos racionales de una curva elíptica. Las siguientes transformaciones en los distintos tipos de campos obedecen a la búsqueda de una forma más simple de la forma normal de Weierstrass.

1. $\text{car}(\mathbb{F}) \neq 2, 3$, si $\text{car}(\mathbb{F}) \neq 2$, el correspondiente cambio de variables es $(x, y) \rightarrow (x, y - \frac{a_1 x + a_3}{2})$, obteniéndose la forma

$$y^2 = x^3 + a_2' x^2 + a_4' x + a_6', \qquad a_2', a_4', a_6' \in \mathbb{F},$$

ahora, si $\text{car}(\mathbb{F}) \neq 3$, el cambio de variables a partir de la ecuación previa es $(x, y) \rightarrow (x - \frac{a_2'}{3}, y)$, lo cual nos lleva a la siguiente forma

$$y^2 = x^3 + a_4'' x^2 + a_6'', \qquad a_4'', a_6'' \in \mathbb{F},$$

2. $\text{car}(\mathbb{F}) = 3$, primero se realiza el cambio $(x, y) \rightarrow (x, y - \frac{a_1 x + a_3}{2})$, con lo cual se transforma la forma normal de Weierstrass en $y^2 = x^3 + a_2' x^2 + a_4' x + a_6'$. A continuación si $a_2' \neq 0$ mediante el cambio $(x, y) \rightarrow (x + \frac{a_4'}{a_2'}, y)$ se transforma la ecuación anterior en

$$y^2 = x^3 + a_2'' x^2 + a_6'', \qquad a_2'', a_6'' \in \mathbb{F},$$

si $a_2' = 0$ se tiene la forma

$$y^2 = x^3 + a_4' x + a_6', \qquad a_4', a_6' \in \mathbb{F},$$

3. $\text{car}(\mathbb{F}) = 2$, para este caso se tienen dos subcasos.

   - *El caso supersingular*, $a_1 = 0$;
     La sustitución $(x, y) \rightarrow (x + a_2, y)$, elimina el término x^2 de la forma normal de Weierstrass

$$y^2 + a_3' y = x^3 + a_4' x + a_6', \qquad a_3', a_4', a_6' \in GF(2^n).$$

   - *El caso no-supersingular*, $a_1 \neq 0$;
     La sustitución $(x, y) \rightarrow (a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3})$, transforma la forma normal de Weierstrass en

$$y^2 + xy = x^3 + a_2'x^2 + a_6', \qquad a_2', a_6' \in GF(2^n).$$

En la implementación que se hizo en este trabajo se empleó una curva no-supersingular sobre un campo finito de característica 2, la representación gráfica de este tipo de curvas es un conjunto finito de puntos sobre el plano.

1.4.1. La ley de grupo de una curva elíptica

A continuación se presenta la explicación de cómo es la operación que define la estructura de grupo sobre los puntos racionales de una curva elíptica, por simplicidad se presenta para el campo de los números reales $\mathbb{R}$. Sin embargo, la misma exposición también puede aplicarse para las curvas definidas sobre campos de característica 2.

1. Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ con $x_i, y_i \in \mathbb{R}$ y $x_1 \neq x_2$, dos puntos distintos cualesquiera sobre la curva, ver la figura 1.1. Observe que la figura de la curva consta de dos partes, esto es así porque entre ambas partes de la gráfica no existe solución a la ecuación para las abscisas correspondientes. Por ejemplo, la ecuación $y^2 = x^3 - 7x$ sobre los reales $\mathbb{R}$, genera una gráfica como la que se presenta en la figura mencionada, donde para los valores de $x \in (0, \sqrt{7})$ no hay solución, de aquí que la gráfica sea discontinua.

   La regla para la suma de P y Q se define de la siguiente manera. Primero, trácese una línea recta que pase por P y Q, esta línea como veremos enseguida, intersecta a la curva elíptica en un tercer punto R', ver figura. Sustituyendo el valor $y = \lambda x + \gamma$ (esta es la ecuación de la recta que pasa por P y Q) en la ecuación de la curva, resulta

$$(\lambda x + \gamma)^2 = x^3 + ax + b, \quad a, b \in \mathbb{R}.$$

   La ecuación anterior tiene tres raíces, dos de ellas son x_1 y x_2, ahora lo que resta es mostrar que existe un tercer punto sobre la curva y sobre la recta que pasa por P y Q, y es precisamente la tercera raíz de la ecuación. La recta que pasa por P y Q es de la forma $y = \lambda x + \gamma$, donde $\lambda = (y_2 - y_1)/(x_2 - x_1)$ y $\gamma = y_1 - mx_1$. Un punto $(x, \lambda x + \gamma)$ sobre la recta está también en la curva si y sólo si $(\lambda x + \gamma)^2 = x^3 + ax + b$.

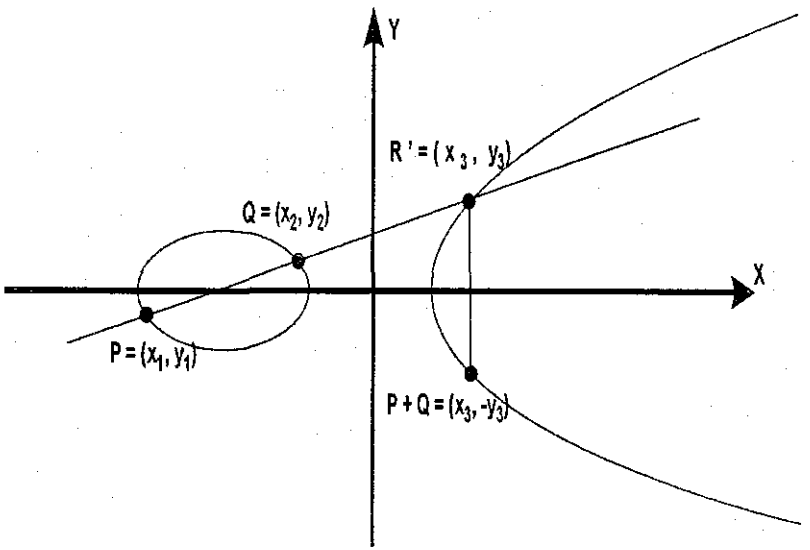

Figura 1.1: Descripción geométrica de la suma de dos puntos distintos.

Entonces, existe un punto de intersección para cada raíz de la ecuación cúbica $x^3 - (\lambda x + \gamma)^2 + ax + b$. Sabemos ya que existen dos raíces x_1 y x_2, porque $(x_1, \lambda x_1 + \gamma)$ y $(x_2, \lambda x_2 + \gamma)$ son los puntos P y Q respectivamente sobre la curva. Ahora bien, la suma de las raíces de un polinomio mónico es igual a menos el coeficiente de la segunda potencia más alta, por lo cual se concluye que la tercera raíz en este caso es de la forma $x_3 = \lambda^2 - x_1 - x_2$. Entonces, dado que $\gamma = y_1 - \lambda x_1$, deducimos el valor para y_3 si sustituimos el valor de x_3 en la ecuación de la recta. Una vez que se tiene el tercer punto de intersección $R' = (x_3, y_3)$, procedemos a reflejarlo sobre el eje de las abscisas,resultando el punto $R = (x_3, -y_3)$, este último punto se define como la suma de P y Q, es decir, $P + Q = R$. Esto nos lleva a encontrar una expresión para las coordenadas de R en términos de x_1, x_2, y_1, y_2:

$$x_3 = (\tfrac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \ ;$$
$$y_3 = -y_1 + (\tfrac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3).$$

2. Sea $P = (x_1, y_1)$ con $x_1, y_1 \in \mathbb{R}$, un punto sobre la curva, con la

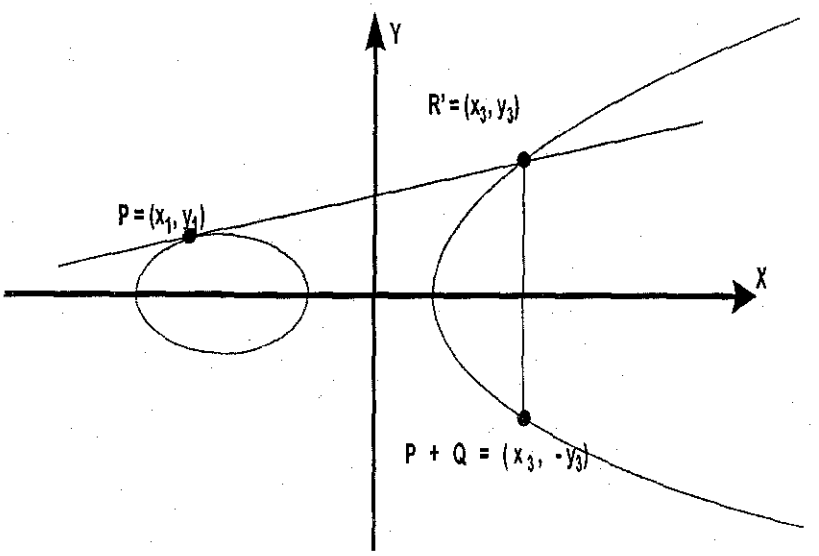

Figura 1.2: Descripción geométrica de la suma de un punto consigo mismo.

particularidad de que $y_1 \neq 0$.

A la suma de un punto P consigo mismo suele llamársele el *doble* de $P = (x_1, y_1)$, y se define a continuación. Se traza una línea recta tangente a la curva elíptica en el punto P, ver figura 1.2. Esta línea intersectará a la curva en un segundo punto $R' = (x_3, y_3)$, el argumento de la existencia de esta intersección es el mismo que para el caso anterior. De manera idéntica, la reflexión de este punto sobre el eje de las abscisas permite obtener otro punto, $R = (x_3, -y_3)$. Este punto se define como $2P$. La deducción de las fórmulas es semejante al caso anterior, excepto que ahora α es la derivada dy/dx en P, se tiene:

$$x_3 = (\tfrac{3x_1^2+a}{2y_1})^2 - 2x_1;$$
$$y_3 = -y_1 + (\tfrac{3x_1^2+a}{2y_1})(x_1 - x_3).$$

3. Sea $P = (x, y)$ un punto sobre la curva, observando la ecuación transformada resulta obvio que el punto $Q = (x, -y)$ también satisface la ecuación. El punto Q es precisamente el punto simétrico a P con respecto al eje de las abscisas, ver figura 1.3.

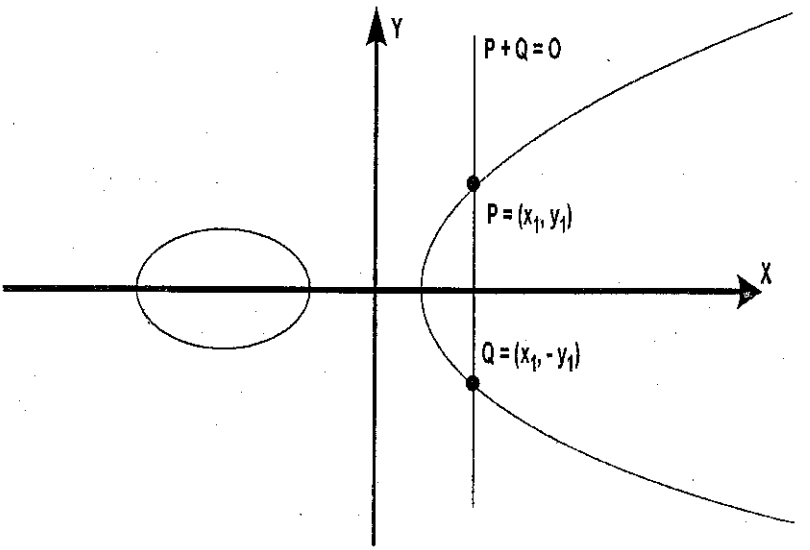

Figura 1.3: Descripción geométrica de la suma de P con -P.

Al reflejo de P sobre el eje de las abscisas se le denotará por $Q = -P$. Teniendo presente lo anterior, sean $P = (x_1, y_1)$ y $Q = (x_1, -y_1)$ dos puntos distintos sobre la curva elíptica E y sobre la recta $x = x_1$. El tercer punto de "intersección" de la recta con la curva se ha convenido en llamarlo el punto al infinito $\mathcal{O}$, ver figura 1.3, y la suma de estos puntos para este caso se define como $P + Q = \mathcal{O}$. De esto se justifican dos aspectos importantes sobre la curva y sus puntos con la operación que se describe, la existencia del elemento neutro y la existencia del inverso para cualquier punto de la curva. Para ver por qué el punto al infinito es el neutro obsérvese el punto P de la figura 1.3, supongamos que se realiza la suma $\mathcal{O} + P$, entonces, el tercer punto de intesección de la recta que pasa por $\mathcal{O}$ y P es $Q = -P$, por lo que el resultado de la suma será el punto simétrico a Q, el cual es precisamente P. En la práctica el punto al infinto se representa por el $(0, 0)$.

Para campos de característica distinta a 2 y 3 ya se presentaron las coordenadas de $P + Q$ en terminos de las de $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, lo que sigue es presentar las correspondientes expresiones para curvas elípticas definidas sobre campos de característica 2.

Sea $P = (x_1, y_1)$ un punto en $E(GF(2^n))$ con $P \neq \mathcal{O}$ y $P \neq -P$, es fácil ver que si P está en la curva entonces $-P = (x_1, x_1 + y_1)$ también lo está. Es decir, el punto $(x_1, x_1 + y_1)$ es el inverso del punto (x_1, y_1) en el caso de campos de característica 2. De manera análoga, para la ecuación de una curva definida sobre campos $GF(2^n)$, tenemos a la recta $y = \lambda x + \gamma$ que pasa por los puntos $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, con $x_1 \neq x_2$. Sustituyendo el valor $y = \lambda x + \gamma$ en la ecuación de la curva, utilizando el argumento de que la suma de las raíces es igual a menos el coeficiente de la segunda potencia más alta y considerando que se trabaja sobre un campo de característica 2, se tiene que $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2'$, donde a_2' es el coeficiente del término cuadrático de la ecuación de una curva elíptica no-supersingular definida en un campo de característica 2. Se sabe que $\gamma = y_1 + \lambda x_1$, entonces, sustituyendo este valor y el de x_3 en la ecuación de la recta, obtenemos $y_3' = \lambda(x_3 + x_1) + y_1$. Esta es la coordenada del tercer punto de la recta $y = \lambda x + \gamma$ que intersecta a la curva en R', lo que resta es reflejar el punto sobre el eje de las abscisas, como hemos visto si un punto (x, y) está en la curva, entonces $(x, x + y)$ también lo está, por lo que $y_3 = \lambda(x_3 + x_1) + x_3 + y_1$.

Entonces, las fórmulas para la suma $P + Q = (x_3, y_3)$ son:

$$\begin{gathered} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2', \\ y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \\ \text{con } \lambda = \tfrac{y_1 + y_2}{x_1 + x_2}. \end{gathered}$$

Para este caso $x_1 + x_2 \neq 0$ ya que tanto P como Q no son el punto al infinito y no tienen la misma abscisa.

Las fórmulas anteriores son válidas únicamente si $P \neq Q$; para $P = Q$, con $x_1 \neq 0$, son un poco diferentes:

$$\begin{gathered} x_3 = \lambda^2 + \lambda + a, \\ y_3 = x_1^2 + (\lambda + 1)x_3, \\ \text{con } \lambda = x_1 + \tfrac{y_1}{x_1}. \end{gathered}$$

La suma de P consigo mismo k veces, es decir, kP, para cualquier $k \in \mathbb{Z}$, puede calcularse repetidamente doblando y sumando puntos. Estas expresiones definen la operación o suma bajo la cual el conjunto de puntos sobre la curva elíptica (sobre un campo de caractarística 2) junto con el punto al infinito forman un grupo, Koblitz [19].

El lector puede consultar las referencias clásicas sobre la teoría de curvas elípticas y geometría algebraica en Semple J. G. y Roth L. [34], Walker R. J. [38] y Fulton W. [8], otras son Hartshorne R. [11], Griffits P. y Harris J. [10], Housemöller [14], Koblitz [21] y Silverman [35], [36]. Para conocer más sobre la historia del desarrollo de la geometría algebraica, puede verse Dieudonné J. [5].

El conjunto de puntos (x, y) que satisfacen la ecuación de la curva sobre un campo $GF(2^n)$ junto con el "punto al infinito" $\mathcal{O}$ será denotado por $E(\mathbb{F}_{2^n})$. La cardinalidad de este conjunto es finita (ver teorema 1.4.1 más adelante) y se representa como $\#E(\mathbb{F}_{2^n})$.

Como se ha mencionado, la seguridad del criptosistema que se desarrolla en este trabajo, se basa en la aparente dificultad de resolver el PLDCE. Antes de definir lo que es el PLDCE en primer lugar se definirá el problema del logaritmo discreto de un elemento β en un grupo $\mathcal{G}$ y posteriormente se define lo que es el *problema del logaritmo discreto* el cual se traslada o contextualiza sobre las curvas elípticas.

Definición 1.4.1. Sea $\mathcal{G}$ un grupo fininto cíclico de orden n. Sea α un generador de $\mathcal{G}$ y sea $\beta \in \mathcal{G}$. El *logaritmo discreto de β en base α* denotado por $\log_\alpha \beta$, es el entero x único, $0 \leq x \leq n-1$, tal que $\beta = \alpha^x$.

Definición 1.4.2. El *problema del logaritmo discreto* **(PLD)** es el siguiente: dado un grupo cíclico de orden n, un generador $\alpha \in \mathcal{G}$ y un elemento $\beta \in \mathcal{G}$, encontrar el entero x, $0 \leq x \leq n-1$, tal que $\beta = \alpha^x$.

Nótese que en las dos definiciones anteriores se asume notación multiplicativa, la cual también pudo haber sido aditiva.

Definición 1.4.3. El *problema del logaritmo discreto para curvas elípticas* **(PLDCE)** es el siguiente: dados un subgrupo cíclico $\mathcal{H}$ de orden n, del grupo de puntos $E(\mathbb{F})$, un punto P generador de $\mathcal{H}$ y un punto $Q \in \mathcal{H}$, encontrar el entero k único, $0 \leq k \leq n-1$, tal que $Q = kP$.

De esta última definición recuérdese que la operación sobre los puntos de la curva elíptica es aditiva.

1.4.2. Las curvas supersingulares y no-supersingulares en los criptosistemas basados en el PLDCE

En esta sección se exponen las razones por las cuales las curvas supersingulares no resultan útiles desde el punto de vista de la seguridad, dado que resultan suceptibles de ataques eficientes. Dentro de las curvas no-supersingulares, existe un tipo especial de curvas que poseen buenas propiedades para la implementación de criptosistemas basados en el PLDCE. Además, no se conocen hasta ahora algoritmos eficientes que pongan en riesgo la seguridad del criptosistema que se desarrolle utilizando estas curvas, Koblitz *et al* [22].

Curvas supersingulares
Desde el punto de vista de la seguridad, este tipo de curvas no resultan ser satisfactorias y por tanto carecen de interés desde nuestra perspectiva, la razón es la siguiente. Se ha encontrado un algoritmo llamado MOV (en referencia a sus creadores Menezes, Okamoto y Vanstone [28]), el cual resuelve el PLDCE en tiempo subexponencial. A grandes rasgos, el algoritmo consiste en reducir el PLDCE sobre un campo F_{2^n} al problema del logaritmo discreto (PLD) sobre una extensión $F_{2^{kn}}$. Si se desea evitar el ataque MOV para este tipo de curvas lo conducente es trabajar en un campo lo "suficientemente" grande, donde el trabajo computacional para calcular el PLD es lento. Sin embargo, una de las consecuencias importantes que esto conlleva es que se gana en seguridad, pero se sacrifica eficiencia, ya que el trabajo computacional no sólo se incrementa para el criptoanalista, sino también para quien cifra/descifra.

Curvas no-supersingulares
El algoritmo MOV no resulta eficiente para las curvas no-supersingulares, por otra parte, los ataques mejor conocidos para el PLD se basan en la factorización del número $n = \#E(F_{2^n})$ de puntos en la curva, es decir, en los factores del orden del grupo. De esta manera, si se escoge una curva no-supersingular donde el número de puntos en la curva tenga un factor primo lo "suficientemente" grande, se podrá tener una seguridad aceptable. Actualmente, suponiendo un ataque con los mejores algoritmos conocidos trabajando en paralelo, se estima que un número $n = \#E(F_{2^n})$ será seguro siempre que tenga un factor primo de por lo menos 50 dígitos decimales, Koblitz *et al* [22].

Existe un tipo especial de curvas no-supersingulares las cuales poseen buenas propiedades criptográficas, dichas curvas son llamadas **anómalas** o **de Koblitz.** La curva que se usa en este trabajo es de este tipo, dichas curvas tienen las siguientes propiedades:

1. Son no-supersingulares (no puede usarse MOV eficientemenete).
2. El número de puntos sobre la curva tiene un factor primo grande (no pueden calcularse logaritmos discretos eficientemente con los mejores algoritmos).
3. Las curvas son fáciles de hallar.
4. El doblar puntos (sumar un punto consigo mismo) es eficiente.

El lector puede consultar con mayor detalle sobre este tipo de curvas en Koblitz [18]. Antes de continuar con la descripción de estas curvas se presentan un par de definiciones necesarias.

Definición 1.4.4. Sea $\mathbb{K} = \mathbb{F}_{p^n}$ un campo de extensión de $\mathbb{F}_p$. La *traza* de $\alpha \in \mathbb{K}$ está definida por la suma $Tr(\alpha) = \sum_{i=0}^{n=1} \alpha^{p^i}$, la cual define un mapeo con dominio $\mathbb{K}$ e imagen el campo base $\mathbb{F}_p$.

Se mostrará para el caso de campos de característica 2 por qué la traza mapea elementos de $\mathbb{F}_{2^n}$ en elementos de $\mathbb{F}_2$. Primero, sea $\alpha \in \mathbb{F}_{2^n}$, entonces

$$Tr(\alpha) = \sum_{i=0}^{n=1} \alpha^{2^i} = \alpha + \alpha^2 + \cdots + \alpha^{2^{n-1}},$$

por la propiedad para campos de característica 2 de que $(\alpha + \beta)^2 = \alpha^2 + \beta^2$, se tiene

$$(Tr(\alpha))^2 = \alpha^2 + \alpha^4 + \cdots + \alpha^{2^{n-1}} + \alpha^{2^n}.$$

Se sabe que si $\mathbb{F}_{2^n}$ es un campo con 2^n elementos, entonces todo elemento $x \in \mathbb{F}_{2^n}$ satisface la ecuación $x^{2^n} - x = 0$, es decir, $x^{2^n} = x$, Fraleigh [7]. Por lo anterior tenemos que $(Tr(\alpha))^2 = Tr(\alpha)$. Ahora bien $Tr(\alpha) = 0$ o $Tr(\alpha) \neq 0$, lo que resta mostrar es que si $Tr(\alpha) \neq 0$, entonces $Tr(\alpha) = 1$. De la igualdad $(Tr(\alpha))^2 = Tr(\alpha)$, se multiplica por el inverso de $Tr(\alpha)$ (existe porque $Tr(\alpha) \neq 0$) en cada lado de la igualdad, lo que nos lleva a $Tr(\alpha) = 1$,

con lo cual se muestra lo dicho en la definición.

Existe un resultado (Koblitz [19]) para el caso de campos $\mathbb{F}_{2^n}$, el cual dice que la mitad de los elementos para este tipo de campos tienen traza cero y la otra mitad traza uno. Una propiedad importante de la traza es que es un operador lineal, en particular, para cualesquiera $x, y \in K$, $Tr(x+y) = Tr(x) + Tr(y)$.

Conocer el número de puntos que posee una curva elíptica resulta crucial para la seguridad de un criptosistema basado en el PLDCE, pues es posible evitar los ataques que se basan en la descomposición en potencias de primos del número de puntos sobre la curva elíptica, Pollard [30]. La siguiente definición será útil con el propósito de calcular el número de puntos en curvas anómalas.

Definición 1.4.5. Dada la curva no-supersingular $E : y^2 + xy = x^3 + a_2x^2 + a_6$, definida sobre $\mathbb{F}_{2^n}$, se llama a la curva $\widetilde{E} : y^2 + xy = x^3 + (a_2 + D)x + a_6$ la pareja de E, donde $D \in \mathbb{F}_{2^n}$ y $Tr(D) = 1$.

La sucesión $A = \{a_n = a_{n-1} - pa_{n-2} \mid a_0 = 2, a_1 = 1\}$ donde p es un número primo, permite de manera sencilla y directa, calcular el número de puntos sobre una curva anómala, Koblitz [18]. Recuérdese que se buscan aquellas curvas elípticas cuyo número de puntos posea un factor primo "grande".

Dada la sucesión A, el número de puntos para una curva elíptica anómala E y su pareja $\widetilde{E}$ definidas ambas sobre el campo $\mathbb{F}$, está determinado por las siguientes expresiones $\#E = p^n + 1 - a_n$ y $\#\widetilde{E} = p^n + 1 + a_n$ respectivamente, donde a_n es el n-ésimo elemento de la sucesión A, Koblitz [18]. Para los ejemplos que se presentan, las ecuaciones de E y $\widetilde{E}$ son, $E : y^2 + xy = x^3 + x^2 + 1$ y $\widetilde{E} : y^2 + xy = x^3 + 1$, los cálculos se hicieron utilizando *Mathematica*.

Ejemplo 1.4.1. Para $n = 11$ se tiene $a_{11} = 67$, por lo que para la curva anómala E se tienen $\#E(\mathbb{F}_{2^{11}}) = 2^{11} + 1 - 67 = 1982$ puntos, y para su pareja $\widetilde{E}$ se tienen $\#\widetilde{E}(\mathbb{F}_{2^{11}}) = 2^{11} + 1 + 67 = 2116$ puntos.

Ejemplo 1.4.2. Para $n = 209$, $a_{209} = 729482828633575692274773046645 9$, factorizándose $\#E(\mathbb{F}_{2^{209}}) = 2^{209} + 1 - a_{209}$ en

$2 \times 991 \times 262543 \times 1581120611360106183591142862019334962897570878255147779,$

para su pareja $\#\widetilde{E}(\mathbb{F}_{2^{209}}) = 2^{209} + 1 + a_{209}$ se factoriza como $2^2 \times 23^2 \times 15467 \times 130873 \times 149384756497052748364495 1 \times 128585205571480139566670587$.

De este último ejemplo, se puede notar que para el caso de la curva E el número de puntos para el campo $\mathbb{F}_{2^{209}}$ cuenta con un factor primo de 55 dígitos. Esto convierte al campo $\mathbb{F}_{2^{209}}$ en un buen candidato para implementar un criptosistema. Sin embargo, en un trabajo publicado por Gaudry P. [9], se muestra que para campos del tipo $\mathbb{F}_{2^{mn}}$, donde m y n son dos enteros, la seguridad ya no es tan buena, pues se ha encontrado un algoritmo eficiente que resuelve el PLDCE para curvas definidas sobre este tipo de campos, por lo cual el campo $\mathbb{F}_{2^{209}} = \mathbb{F}_{2^{11\cdot 19}}$ se descarta. Los campos de característica 2 que resultan adecuados para proporcionar una seguridad aceptable en la actualidad son los del tipo $\mathbb{F}_{2^p}$, donde p es un número primo. Se presenta ahora un teorema muy importante, el cual se conoce generalmente como el "teorema de Hasse", que permite ubicar la cota del orden del grupo de puntos de una curva elíptica, Koblitz [19].

Teorema 1.4.1. *El número N de puntos racionales de una curva elíptica definida sobre un campo $\mathbb{F}_{p^n}$ está en el intervalo*

$$p^n + 1 - 2\sqrt{p^n} \leq N \leq p^n + 1 + 2\sqrt{p^n}.$$

Finalmente, se presenta para $n \leq 375$ la siguiente tabla, la cual muestra los números n para los que existe una buena implementación de un criptosistema basado en el PLDCE usando una base polinomial, siendo el polinomio irreducible que se utiliza de la forma $x^n + x^k + 1$, Angel [1]. Deberá ponerse atención en aquellos números n que sean primos.

n	k	n	k	n	k
3	1	81	4	209	6
5	2	89	38	231	26
9	1	95	11	233	74
11	2	105	4	239	36
23	5	113	9	273	23
29	2	119	8	281	93
33	10	135	11	303	1
35	2	155	62	329	50
41	3	183	56	359	68
65	18	191	9	375	16

Exponentes para el trinomio $x^n + x^k + 1$.

De todo lo anterior se puede concluir hasta aquí, que las curvas elípticas que son de interés para su uso en criptografía son las no-supersingulares ($j \neq 0$), y dentro de estas curvas se escogen las llamadas anómalas, ya que se facilita el cálculo del número de puntos, son no-supersingulares, el número de puntos sobre la curva tiene un factor primo grande (no pueden calcularse logaritmos discretos eficientemente con los mejores algoritmos) y las curvas son fáciles de hallar. Asimismo, los campos binarios que resultan ser adecuados son los del tipo $GF(2^p)$ siendo p un número primo.

Capítulo 2

Elementos de un sistema de cifrado basado en el PLDCE

El propósito fundamental de este trabajo es construir un sistema de cifrado de clave pública basando su seguridad en la dificultad de resolver el PLDCE. La descripción del PLDCE es la siguiente: se tiene una curva elíptica E definida sobre un campo finito F, se fija un punto P sobre la curva y un número entero positivo $k \in F$. Se calcula kP (se suma P consigo mismo k veces), lo cual arroja otro punto Q sobre la curva. El problema es, ¿quién es k dados F, E, P y Q? Para este problema no se conoce hasta ahora un algoritmo eficiente que proporcione una respuesta en tiempo polinomial cuando la cardinalidad del grupo de los puntos racionales de la curva elíptica tiene un factor primo de tamaño "suficientemente" grande y la curva posee las características deseables descritas en el capítulo anterior.

En este capítulo se exponen los elementos que conforman un sistema de cifrado basado en el PLDCE, el campo en consideración es de característica 2 y tiene una base de tipo polinomial o estándar. Tales elementos son: (1) la ecuación de una curva elíptica, (2) un polinomio irreducible en el campo en el que se define a la curva, (3) el campo sobre el que se define la curva, (4) un punto de la curva elíptica y (5) el orden de dicho punto, el cual debe ser un número primo "grande".

Durante el desarrollo del trabajo se hicieron un par de aproximaciones previas a la implementación final variando el campo y el polinomio utilizados. La referencia sobre qué campo y qué polinomio utilizar fue el documento FIPS

186-2 [4] y la tabla del final del capítulo anterior.

En primera instancia se trabajó con la curva anómala $y^2+xy = x^3+x^2+1$ sobre dos campos y sus respectivos polinomios de la tabla mencionada, a saber, $F_{2^{113}}$ y $F_{2^{209}}$. El campo $F_{2^{113}}$ marcó la pauta para trabajar con campos de proporciones "respetables" que resultaran eficientes en la implementación de su aritmética. Sin embargo, la implementación no era segura debido a que el número de puntos sobre la curva para dicho campo no posee un factor primo de por lo menos 50 dígitos decimales. Para este caso el número de puntos sobre la curva es 10384593717069655255793407666935014, el cual posee sólamente un factor de 34 dígitos decimales.

La implementación correspondiente al campo $F_{2^{209}}$ sí cumple con los criterios de seguridad que se han mencionado, ya que la descomposición como producto de potencias de primos del número de puntos sobre la curva cuenta con un factor primo de 55 dígitos decimales, el número de puntos es 822752278660603021077484591278667957663081597059867183943838054. Sin embargo, la última versión del criptosistema corresponde a otra curva y a otro campo debido a que hacia el final del trabajo se decidió utilizar los elementos (la curva, el campo, el polinomio y el punto) que recomienda el Departamento de Comercio de E.U. y que publica el Instituto Nacional de Estándares y Tecnología (NIST), por medio del documento FIPS 186-2 (Federal Information Proccesing Standards Publication) [4]. En dicho documento se señalan de manera explícita los algoritmos, campos, números primos, curvas y puntos que resultan seguros hoy en día para ser implementados en aplicaciones que requieran la generación de firmas digitales. La firma digital de una entidad A sobre un mensaje M, es una transformación que asocia la identidad de A y el mensaje M con una cadena (posiblemente binaria) de longitud fija. La transformación es mantenida en secreto por parte de A, mientras que la verificación de dicha transformación deberá ser de dominio público. A diferencia de una firma autógrafa, una firma digital está completamente ligada al documento que se firma, es decir, la firma asocia la identidad de una entidad junto con el documento, de tal manera que si el documento sufriera un cambio, por mínimo que sea el cambio la firma digital ya no será la misma, además de constatar la identidad del firmante.

En el FIPS 186-2 se recomiendan tres tipos de algoritmos para la firma

digital: el primero es DSA, el segundo es el algoritmo de RSA y el tercero es el **algoritmo de firma digital para curvas elípticas, ECDSA** (por sus siglas en inglés). Para este último se prescriben específicamente diferentes campos, las correspondientes curvas y polinomios irreducibles, así como un punto específico sobre la curva.

Los elementos prescritos en el FIPS 186-2, se refieren al empleo de los mismos por parte de cualquier empresa comercial o entidad de gobierno que desee generar firmas digitales, es por ello que en este trabajo se decidió emplear para el caso de ECDSA, la opción del campo binario $\mathbb{F}_{2^{233}}$ junto con la curva anómala cuya ecuación es $y^2 + xy = x^3 + 1$ y el polinomio irreducible $x^{233} + x^{74} + 1$ en $\mathbb{F}_2[x]$. Sin embargo, se presenta un algoritmo para hallar soluciones no triviales a ecuaciones definidas sobre campos binarios, esto con el fin de no limitarnos únicamente a los elementos que están contenidos en el FIPS 186-2.

Como se verá en los diferentes procesos (cifrado/descifrado, firma/verificación, etcétera) que se llevan a cabo en un criptosistema basado en el PLDCE, es necesario sumar recurrentemente puntos sobre la curva, por lo que resulta natural buscar un procedimiento óptimo para realizar la suma de puntos. Dado que los puntos son parejas ordenadas de elementos en el campo F, al llevar a cabo la adición de puntos, lo que subyace de fondo es la aritmética en F.
Entonces, el primer paso es escoger un campo y después trabajar en la manera de representar sus elementos, de tal forma que se tenga una aritmética eficiente en F.

2.1. Elementos del sistema

Los elementos de la versión final del criptosistema motivo de este trabajo se tomaron del conjunto de recomendaciones emitidas en el FIPS 186-2, [4] cuando se utiliza el algoritmo de firmas digitales basado en curvas elípticas (ECDSA). A continuación se listan los elementos con los que se construyó finalmente el criptosistema.

1. *La curva* anómala (también llamada de Koblitz) $y^2 + xy = x^3 + 1$.
2. El polinomio irreducible $x^{233} + x^{74} + 1$.

3. *El campo* $F_{2^{233}}$.

4. *Un punto* $P_0 = (X, Y)$ sobre la curva, del cual se dan las coordenadas en formato hexadecimal, tal como aparecen en el FIPS 186-2

   X =17232ba853a7e731af129f22ff4149563a419c26bf50a4c9d6eefad6126,
   Y =1db537dece819b7f70f555a67c427a8cd9bf18aeb9b56e0c11056fae6a3.

5. *El orden* de P_0 es el número primo

   n = 3450873173395281893717377931138512760570940988862252126 − 328087024741343. Se comprobó que $nP_0 = \mathcal{O}$ mediante rutinas que implementaron la suma de puntos sobre $E(\mathbb{F}_{2^{233}})$.

La curva $y^2+xy = x^3+1$ sobre el campo $F_{2^{233}}$ cumple con todas las características deseables que se describieron al final del capítulo anterior. En particular, el orden del grupo de puntos sobre la curva satisface el objetivo de tener un factor mayor a 45 dígitos decimales, en este caso se tiene un factor primo n de 70 dígitos. Explícitamente, el orden del grupo de puntos es $\#E(F_{2^{233}}) = 13803492693581127574869511724554051042283763955449008505312348 09 - 8965372$, cuya descomposición como producto de potencias de primos es $2^2 \times n$.

A pesar de que en el FIPS 186-2 se especifica un punto sobre la curva, es deseable contar con un algoritmo que permita hallar puntos no triviales en una curva elíptica definida sobre un campo finito de característica 2. Esto posibilita trabajar con una gama más amplia de curvas y campos finitos de característica 2 y no únicamente restringirse a los que el FIPS prescribe.

2.2. Algoritmo para hallar puntos sobre una curva no supersingular

En esta sección se describe el algoritmo probabilístico para encontrar soluciones a la ecuación de una curva no supersingular definida en un campo F_{2^n} de característica 2, Koblitz [19]. La ecuación general de una curva no supersingular definida sobre un campo de característica 2 es

$$y^2 + xy = x^3 + a_2x^2 + a_6, \qquad a_2, a_6 \in F_{2^n}.$$

El algoritmo para encontrar soluciones no triviales a la ecuación $y^2+xy = x^3 + a_2x^2 + a_6$ sobre un campo F_{2^n} es el siguiente.

1. Elegir de manera aleatoria $x \in F_{2^n}$, con $x \neq 0$.

2. Calcular $z = x + a_2 + a_6x^{-2}$ y $Tr(z) = \sum_{i=0}^{n-1} z^{2^i}$. Si $Tr(z) = 1$, regresar al paso anterior.

3. Si $Tr(z) = 0$, encontrar $u \in F_{2^n}$, que satisfaga $u^2 + u = z$. Dado que el mapeo $u \rightarrow u^2 + u$ es 2 a 1 (Koblitz [19]), sabemos que su imagen es la mitad de $\mathbb{F}_2$; de aquí que la imagen consiste de todos los z tales que $Tr(z) = 0$, con lo cual se asegura que la ecuación mencionada tiene solución en $\mathbb{F}_2$. Observe que para una base de F_{2^n} sobre F_2, la ecuación anterior se puede reescribir como el sistema de ecuaciones lineales dados por $(M + I)\mu = z$, donde M es la matriz de evaluación al cuadrado (aquélla que mapea a toda μ en μ^2), es decir, $M\mu = \mu^2$, I es la matriz identidad de $n \times n$, μ es el vector $(1, x, \ldots, x^{n-1})$ y z es un vector con respecto a la base.

4. El punto $P = (x, xu)$ satisface la ecuación y por tanto está sobre la curva.

A continuación se muestra que el punto $P = (x, xu)$ efectivamente es un punto sobre la curva definida por la ecuación $y^2 + xy = x^3 + a_2x^2 + a_6$.

Si se sustituye la pareja ordenada $(x, y) = (x, xu)$ en el lado izquierdo de la ecuación se tiene

$$y^2+xy = (xu)^2+x\,(xu) = x^2\,(u^2 + u) = x^2z = x^2\,(x + a_2 + a_6x^{-2}) = x^3+a_2x^2+a_6.$$

De la igualdad anterior queda demostrado que la pareja ordenada satisface la ecuación de la curva elíptica.

En campos de característica 2, para cualesquiera $x, y \in F_{2^n}$, se cumple la propiedad $(x + y)^2 = x^2 + y^2$. De esta propiedad se desprende que si $x \in F_{2^n}$, entonces $Tr\,(x) = (Tr\,(x))^2 = Tr\,(x^2)$.

La traza es un operador lineal y su imagen está en el campo base, en este caso $F_2 = \{0, 1\}$, por lo cual para cualquier $z \in F_{2^n}$, la traza de z es 0 o es 1. En el algoritmo se pide que $Tr(z) = 0$, ya que si fuera 1 no habría solución a la ecuación $u^2 + u = z$. Si $Tr(z) = 1$ implica

$$1 = Tr(z) = Tr(u^2 + u) = Tr(u^2) + Tr(u) = 2Tr(u) = 0,$$

lo cual evidentemente es una contradicción.

Hasta el momento se ha dicho cómo encontrar un punto no trivial sobre la curva; sin embargo, es necesario que el punto pertenezca a un subgrupo cíclico cuyo orden sea un primo p, donde p es uno de los factores primos del número m de puntos sobre la curva. Como se verá en el siguiente capítulo, la petición de que el punto pertenezca al subgrupo cíclico de orden primo es necesaria al momento de implementar el algoritmo de firma digital para curvas elípticas.

Supóngase que el orden del punto P no es el número primo p, con el propósito de encontrar, a partir de P, un punto P' cuyo orden sea p, se ejecutará el siguiente algoritmo.

1. Se escoge un número aleatorio $k \in [1, p-1]$.
2. Se calcula $kP = Q$.
3. Se calcula el orden de Q. Si el orden es un divisor de $\frac{m}{p}$ se regresa al punto (1).
4. Se calcula el punto $\frac{m}{p}Q = P'$, este último punto tiene el orden deseado. Puesto que $O(Q)$ no divide a $\frac{m}{p}$, entonces el orden de Q es m o p, si fuera el primero se tendría que $m\,P' = m(\frac{m}{p})\,Q = (\frac{m}{p})\,Q = 0$, lo cual evidentemente es una contradicción, de lo cual se desprende que P' tiene orden p.

El algoritmo presentado en esta sección nos ocupó una parte importante del tiempo de trabajo de esta tesis. En un principio, el problema de resolver una ecuación sobre un campo finito parece algo no muy complicado, lo cual es cierto para quien está iniciado en geometría algebraica. Pasado un tiempo penoso tratando de encontrar una respuesta a dicho problema, nos percatamos de la necesidad de contar con ayuda experta. Cuando estamos hablando de encontrar soluciones a ecuaciones definidas en campos binarios finitos, se consideran a estos últimos de tamaño respetable ($\mathbb{F}_{2^{113}}, \mathbb{F}_{2^{155}}, \mathbb{F}_{2^{209}}$) en la actualidad. Lo anterior nos llevó a consultar matemáticos inmersos en geometría algebraica,

tanto en el Instituto de Matemáticas como en el IIMAS, ambos en la UNAM. Desafortunadamente, a pesar de su generosa disposición no fue posible obtener resultados satisfactorios. Con esto se evidencia la dificultad de encontrar personas que trabajen en geometría algebraica y que se interesen en cuestiones prácticas como es la criptografía. La solución a esta cuestión provino de Neal Koblitz, quien es profesor de la Universidad de Washington y que amablemente nos asistió y contestó a cada una de nuestras dudas.

Por último, es importante señalar que el algoritmo para encontrar puntos sobre una curva elíptica cuyo orden sea un número primo p grande fue escencial para las implementaciones de las dos primeras aproximaciones: las de los campos $F_{2^{113}}$ y $F_{2^{209}}$ con la curva anómala $y^2 + xy = x^3 + x^2 + 1$.

Capítulo 3

Implementación de un sistema de cifrado basado en el PLDCE

El objetivo principal de este trabajo ha sido el implementar un sistema de cifrado y firma digital basado en el PLDCE, dando como resultado una biblioteca de rutinas criptográficas, la cual cuenta con las siguientes opciones:

1. Generación de claves (pública y privada)
2. Cifrado
3. Descifrado
4. Firma digital
5. Verificación de firma digital
6. Ensobretado
7. Apertura y verificación de sobre

Para la construcción de la biblioteca es necesario implementar tres diferentes aritméticas: (1) la aritmética para el campo $\mathbb{F}_{2^{233}}$, (2) la aritmética de puntos sobre la curva elíptica y (3) la aritmética sobre un campo del tipo $\mathbb{Z}_p$ con p primo. Las dos primeras son necesarias para construir toda la biblioteca y la última se requiere específicamente para implementar el algoritmo de firma digital. En lo que resta del trabajo, cuando se hable del campo $\mathbb{Z}_p$, p denotará al factor primo del número de puntos de la curva prescrita por el FIPS 186-2, es decir, el número primo de 70 dígitos decimales

mencionado en la primera sección del capítulo anterior.

Los aspectos relevantes de la implementación se enmarcan en cuatro bloques: (1) representación y aritmética de los elementos del campo $\mathbb{F}_{2^{233}}$; (2) la adición de puntos sobre la curva elíptica; (3) la aritmética en $\mathbb{Z}_p$ que permite implementar la firma digital y su verificación; y (4) la integración de la biblioteca. Es importante mencionar que la implementación de la biblioteca se realizó utilizando el lenguaje de programación C, por lo cual se empleará código escrito en este lenguaje para describir las rutinas.

El orden que se seguirá en la exposición es el mismo en que se enumeraron los bloques, que marca también la secuencia lógica en la construcción de la bliblioteca.

Para ilustrar el funcionamiento de la biblioteca se mostrarán un par de ejemplos al final del capítulo.

3.1. Aritmética en $\mathbb{F}_{2^{233}}$

Los elementos del campo son polinomios de grado menor a 233. Los coeficientes de los polinomios están en $\mathbb{F}_2$, por lo que **una cadena binaria de longitud 233 representa un elemento en** $\mathbb{F}_{2^{233}}$. La implementación de la aritmética de puntos sobre la curva elíptica mencionada en la primera sección del capítulo anterior se basa en la aritmética sobre $\mathbb{F}_{2^{233}}$, con lo cual, uno de los principales objetivos es conseguir una implementación eficiente de la aritmética sobre $\mathbb{F}_{2^{233}}$.

Representación
En este trabajo, cuando se habla de elementos en $\mathbb{F}_{2^{233}}$, los términos polinomio y número son sinónimos. Dado que un número es una cadena binaria de longitud 233, éste se puede representar con 8 palabras de 32 bits cada una. Sin embargo, en ocasiones, al realizar operaciones con polinomios es necesario elevarlos al cuadrado, por lo cual el grado del mismo se puede duplicar, requiriéndose el doble de espacio para almacenarlo. Internamente un polinomo en $\mathbb{F}_{2^{233}}$ está representado como un arreglo de 16 palabras de 32 bits cada una. Una declaración como la siguiente representa un número `A` en el campo.

```
typedef unsigned long int PL32;
#define NUM_BITSxMBYTE 4
#define NUM_BITSxBYTE 8
#define NUM_PALxPOL 16
#define NUM_BITSxPAL 32
PL32 A[NUM_PALxPOL];
```

Ahora se inicia la descripción de algunas de estas operaciones aritméticas.

Con el propósito de facilitar la descripción de cada una de las rutinas, en el lado izquierdo aparecerá la numeración de las líneas de código para cada rutina, a las cuales se hará referencia constantemente.

Suma

La suma entre polinomios se implementa directamente mediante un OR exclusivo (el cual se representa por el operador ^) entre los bits de cada sumando o polinomio, esto debido a que se está trabajando sobre un campo de característica 2. El resultado de la suma se guarda en el primer sumando.

```
0   POL_Suma(A,B)
1   PL32 A[NUM_PALxPOL], B[NUM_PALxPOL];
2   {
3    int im;
4    for(im = 0; im < NUM_PALxPOL; im++)
5      A[im] = (A[im] ^ B[im]);
6   }
```

Debido a que la suma es un `XOR` directo entre 16 enteros se puede decir que la operación `POL_Suma` no tiene costo.

Antes de abordar las siguientes rutinas de la aritmética, se expondrán un par de rutinas, las cuales nos permiten realizar corrimientos de bits y calcular el grado de un polinomio.

En el lenguaje de programación C los operadores >> y << funcionan sobre enteros, en este caso enteros de 32 bits y realizan corrimientos de bits, tratándose de un corrimiento de mas de 31 posiciones provocará que

el operando sea cero. Es por lo anterior que se requiere implementar la rutina POL_IShift para realizar corrimientos de 32 o más posiciones hacia la izquierda. La rutina POL_IShift recibe dos parámetros, un polinomio A y un número entero c y realiza la multiplicación de A por x^c, lo cual equivale a decir que se llevará a cabo un corrimiento hacia la izquierda de A por c bits. Un polinomio está representado por 16 palabras de 32 bits cada una, POL_IShift comprueba primero si el corrimiento será de 32 o más posiciones (línea 2), si así fuera ejecuta una división de c entre NUM_BITSxPAL, el cociente es el número de palabras que se recorrerán hacia la izquierda las palabras significativas de A (líneas 5 y 6), mientras que el residuo (línea 9) será un número entre 0 y 31, si éste es diferente de cero se realiza un corrimiento directo de las palabras significativas de A (líneas 15 y 18).

```
0   POL_IShift(A,c) {
1   int im, jm, tm = 0, ic = (c);
2   if( ic >= NUM_BITSxPAL ) {
3    jm = NUM_PALxPOL - 1;
4    tm = ( ic/NUM_BITSxPAL );
5    for( im = 0; im < (NUM_PALxPOL - tm): im++)
6     (A)[jm-im] = (A)[jm-im-tm];
7    for(im=0;im < tm;im++)
8     (A)[im] = 0;
9    ic -= tm*NUM_BITSxPAL;
10  }
11  jm = tm;
12  tm = NUM_BITSxPAL - ic;
13  if(ic) {
14   for(im=NUM_PALxPOL-1;im > jm;im--) {
15    (A)[im] <<= ic;
16    (A)[im] = (A)[im] XOR ((A)[im-1] >> tm);
17   }
18   (A)[jm] <<= ic;
19  }
20  }
```

Existe la contraparte de esta rutina, a saber, POL_DShift, la cual es análoga a POL_IShift, siendo el significado la división de A por x^c y se obtiene realizando un corrimiento hacia la derecha de A por c bits.

La rutima POL_Grado calcula el grado de un polinomio, recibe como parámetros un polinomio A, al cual se le quiere calcular el grado y un entero c, éste último tendrá al resultado del cálculo. Si el polinomio resultara ser cero, el valor de c será -1. Para esta rutina se utiliza una tabla llamada grado, la cual permite calcular el grado de polinomios de grado menor a 8 de manera directa, para esto se le pasa a la tabla una cadena binaria de 8 bits la cual representa un polinomio de grado a lo más 7 y al mismo tiempo un número entero entre 0 y 255. POL_Grado obtiene la posición im de la palabra más significativa de A (líneas 3 y 4), si A resulta ser cero se regresa -1 (línea 5) como se había comentado. Si A no fuera cero se determina cual es el valor de la palabra más significativa, para esto se utilizan las máscaras de 8, 16 y 24 bits, es decir, 0xff, 0xffff y 0xffffff respectivamente. Posteriormente se suman los múltiplos im de NUM_BITSxPAL y de 8, éste último en términos del valor mencionado respecto a las máscaras, finalmente, se suma a lo anterior el resultado de invocar a la tabla grado con respecto a los 8 bits que restan por considerar.

```
0   typedef unsigned char B8;
1   POL_Grado(A,c) {
2   int im;
3   for(im = NUM_PALxPOL - 1; im >= 0; im--)
4    if((A)[im]) break;
5   if( im < 0) (c) = -1;
6   else {
7    if((A)[im] > 0xffff) {
8      if((A)[im] > 0xffffff)
9       (c) = im*NUM_BITSxPAL + 24 + grado[(B8) ((A)[im] >> 24)];
10     else (c)=im*NUM_BITSxPAL+16 + grado[(B8) ((A)[im] >> 16)];
11   } else {
12    if((A)[im] > 0xff)
13     (c) = im*NUM_BITSxPAL + 8 + grado[(B8) ((A)[im] >> 8)];
14     else (c) = im*NUM_BITSxPAL + grado[(B8) ((A)[im])]; }
15  }
16  }
```

Reducción módulo el polinomio irreducible $x^{233} + x^{74} + 1$
La mayoría de las rutinas utilizan a POL_Cpy, esta rutina lo que hace es

copiar el segundo parámetro en el primero, es decir, se copia un arreglo en otro. La siguiente rutina llamada `POL_ModuloPI` recibe como parámetro un polinomio `A`, el cual es reducido módulo el polinomio irreducible $x^{233}+x^{74}+1$. El resultado de la operación se guarda en `A`. Si el grado de `A` es menor a 233, termina la ejecución. En otro caso, se ejecuta el ciclo, en cada iteración se decrementa al menos en uno el grado de `A`, la ejecución finalizará hasta que el grado de `A` sea menor a 233. La manera de realizar esta rutina es particular, debido a que se evitan los cálculos del grado de `A` y de la diferencia de grados entre `A` y $x^{233}+x^{74}+1$. Las constantes PIXb se refieren a los grados de los monomios que conforman al trinomio $x^{233}+x^{74}+1$.

```
0   #define PI1b 0
1   #define PI2b 74
2   #define PI3b 233
3   POL_ModuloPI(A)
4   PL32 A[NUM_PALxPOL];
5   {
6    PL32 Mtmp[NUM_PALxPOL];
7    POL_Cpy(Mtmp, A);
8    POL_DShift(Mtmp, PI3b);
9    while(1) {
10     if(grado del polinomio Mtmp es menor a 233)
11      break;
12     POL_Suma(A,Mtmp);
13     POL_IShift(Mtmp, PI2b);
14     POL_Suma(A,Mtmp);
15     POL_IShift(Mtmp, PI3b - PI2b);
16     POL_Suma(A,Mtmp);
17     POL_Cpy(Mtmp, A);
18     POL_DShift(Mtmp, PI3b);;
19    }
20  }
```

Multiplicación

La multiplicación es probablemente junto con la suma las dos operaciones más invocadas. Los parámetros que recibe para su ejecución son dos polinomios `A` y `B`, se asume que ambos tienen grado menor a 233. En la multiplicación habitual que conocemos entre dos números enteros, uno de los factores se fija

y el otro se recorre un dígito a la vez a partir del dígito que representa las unidades. Lo anterior es precisamente lo que hace el algoritmo para multiplicar dos polinomios con coeficientes en $\mathbb{F}_{2^{233}}$. Sin embargo, en lugar de recorrer bit por bit y con el propósito de obtener una ejecución más eficiente se recorre en bloques de 4 bits, es por ello que se precalculan los primeros 15 múltiplos de B (líneas 6 y 7). Otra de las rutinas que es frecuentemente usada como subrutina es POL_SetCero (línea 8), recibe un polinomio como parámetro y le asigna el polinomio cero, es decir, el arreglo que representa al polinomio es inicializado a cero.

La rutina `POL_Mult` utiliza dos parámetros, `A` y `B`, se fija el factor `B` y `A` se recorrerá en bloques de 4 bits a partir del bit menos significativo. En cada iteración, de acuerdo al valor de los 4 bits menos significativos de `A` (línea 11) se asigna el correspondiente múltiplo de `B` (entre 0 y 15) a `BC` (línea 12). Posteriormente, `BC` tiene un corrimiento hacia la izquierda, el número de posiciones que debe recorrerse `BC` es un múltiplo de 4 (líneas 5, 13 y 17), se realiza una suma parcial y se actualiza el producto. Al término de cada iteración, los 4 bits menos significativos se descartan (línea 16). El ciclo finalizará al momento en que se terminan de procesar todos los bits significativos de `A`, siendo reducido el resultado de la multiplicación módulo $x^{233} + x^{74} + 1$ y copiándose en `A`.

```
0    void POL_Mult(A,B)
1    PL32 *A, *B;
2    {
3    PL32 BC[NUM_PALxPOL], K[m04bits+1][NUM_PALxPOL],
4    sum[NUM_PALxPOL];
5    PL32 rm, md4 = 0;
6    { Se calculan los primeros 15 múltiplos de B y
7     se guardan en k }
8    POL_SetCero(sum);
9    while(1) {
10     if( A es cero) break;
11     if(rm = (A[0] & m04bits)) {
12       POL_Cpy(BC, K[(B8) rm]);
13       POL_IShift(BC, md4);
14       POL_Suma(sum, BC);
15     }
16    POL_DShift(A, NUM_BITSxMBYTE);
```

```
17    md4 += NUM_BITSxMBYTE;
18    }
19    POL_ModuloPI(sum);
20    POL_Cpy(A,sum);
21    }
```

Cuadrado

`POL_Cuad` es la rutina que calcula el cuadrado de un elemento en $\mathbb{F}_{2^{233}}$, se reciben dos parámetros, el primero de ellos contendrá el resultado, mientras que el segundo es el polinomio al cual se le desea calcular su cuadrado. Para esta rutina se utiliza la propiedad de que el cuadrado de una suma sobre campos $GF(2^r)$ es igual a la suma de los cuadrados, esto debido a que los términos cruzados son múltiplos de 2 y por lo tanto valen cero. Dada la posición x de cualquier bit significativo en `A`, la posición de este bit en A^2 será 2x, es decir, se obtiene el "cuadrado" de cada bit de `A` (línea 10). Una vez que ya se calculó A^2, se procede a reducirlo módulo $x^{233} + x^{74} + 1$.

```
0     POL_Cuad(A2,A)
1     PL32 A2[NUM_PALxPOL], A[NUM_PALxPOL];
2     {
3      int im, sm, mm = 0, km, pm;
4      POL_SetCero(A2);
5      for(im = NUM_PALxPOL-1; im >= 0; im--)
6       if((A)[im]) break;
7      for(;im >= 0; im--)
8      for(sm = (A)[im], mm = 1, km = 0, pm = im * NUM_BITSxPAL;
          km < NUM_BITSxPAL; mm <<= 1, km++);
9      if(sm & mm)
10       A2 = A2 XOR x^{2*(pm+km)};
11     POL_ModuloPI(A2);
12     }
```

Inversión

En DeWin *et al* [3] se expone un algoritmo alternativo para calcular inversos en un campo finito cuando se utilizan bases polinomiales. El algoritmo fue nombrado como "casi inverso" y lo que encuentra es un polinomio B junto con un entero k que satisfacen la ecuación $BA + XM = x^k$. El inverso

podrá hallarse dividiendo B entre x^k mod M, ya que $XM = 0$ mod M. A continuación se expone la rutina, la cual se nombró `POL_CInv`, se reciben dos parámetros `Inv` y `A`, en el primero se guardará A^{-1}. En este caso $M = x^{233} + x^{74} + 1$.

```
0   typedef unsigned char B8;
1   POL_CInv(Inv,A)
2   PL32 Inv[NUM_PALxPOL], A[NUM_PALxPOL];
3   {
4    int k = 0;
5    unsigned int gf, gg;
6    PL32 B[NUM_PALxPOL], C[NUM_PALxPOL],
7    F[NUM_PALxPOL],G[NUM_PALxPOL], M[NUM_PALxPOL];
8    POL_SetCero(B); B[0] = 1;
9    POL_SetCero(C);
10    POL_Cpy(F, A);
11   M = x^233 + x^74 + 1;
12   POL_Cpy(G, M);
13   do{
14    do{
15     if(nc = nceros[(B8) F[0]]) {
16      POL_DShiftSmp(F,nc); /* F /= 2^nc */
17      POL_IShiftSmp(C,nc); /* C *= 2^nc */
18      k += nc;
19      gf -= nc;
20     }
21    } while(nc == NUM_BITSxBYTE);
22    if( F(x) igual a 1 ) break;
23    POL_Grado(F, gf); POL_Grado(G, gg);
24    if(gf < gg) {
25     T = F; F = G; G = T; /* INTERCAMBIA F y G */
26     gt = gf; gf = gg; gg = gt; /* Y SUS GRADOS TAMBIEN */
27     T = B; B = C; C = T; /* INTERCAMBIA B y C */
28     POL_Suma(F,G);
29    } else if(gf igual a gg) {
30     POL_Suma(F,G);
31     POL_Grado(F, gf);
32    } else POL_Suma(F,G);
```

```
33      /* EN ESTE PTO. F VUELVE A TENER FACTORES EN x */
34      POL_Suma(B,C);
35     }while(1); /* EN ESTE PUNTO SE TIENE x^k = B(x)A(x) */
36     Inv = POL_ModuloPI( B(x) * x^-k );
```

Entre las líneas 13 y 35, el algoritmo mantiene las siguientes relaciones invariantes $x^k F = BA + XM$ y $X^k G = CA + YM$, cabe señalar que en la práctica no hay razón para almacenar los valores de X y Y. En la línea 14 se emplea una tabla llamada nceros, la cual sirve para calcular el número de ceros que existe en los cuatro bits menos significativos de F[0]. Entre las líneas 14-21 se quitan los factores de x que contenga $F(x)$ mientras se mantienen invariantes las relaciones mencionadas. Al pasar estas líneas $F(x)$ ya no contiene factores x, en la línea 24 se asegura que el grado de $F(x)$ no sea menor al grado de $G(x)$. La línea 28 es fundamental ya que se suma un múltiplo de G a F (el mismo múltiplo de C a B y de manera implícita el mismo múltiplo de Y a X con el fin de preservar invariantes las relaciones) de tal manera que $F(x)$ posee nuevamente al menos un factor x, el cual será extraído en la siguiente iteración del cíclo. Al terminar la ejecución del ciclo principal se tendrá en ese momento la relación $BA+XM = x^k$, o lo que es lo mismo $x^k = BA$, para hallar A^{-1} se debe realizar todavía una división (línea 36). En DeWin *et al* [3] concluyen que el algoritmo euclideano de la división y el casi inverso poseen velocidades comparables para polinomios sobre campos $GF(2^r)$, sin embargo, sobre campos de tamaño "considerable" el primero resultará ligeramente más rápido, debido a que no se necesita dividir entre x^k.

3.2. Aritmética sobre la curva elíptica

Dados dos puntos P y Q sobre la curva, se presentan dos casos para la suma, cuando $P = Q$ y cuando $P \neq Q$. El primer caso en ocasiones se menciona en la literatura como doblar el punto. Las fórmulas de la suma se exponen a continuación y son las mencionadas en la sección 1.4 del primer capítulo, para el caso de campos de característica 2 cuando la curva es no supersingular.

$$P + Q = (x_3, y_3) = (\lambda^2 + \lambda + x_1 + x_2 + a_2, \lambda(x_1 + x_3) + x_3 + y_1),$$

si $P \neq Q$, $Q \neq O$, $Q \neq -P$, $\lambda = \frac{y_1+y_2}{x_1+x_2}$, y

$P+Q = (x_3, y_3) = (\lambda^2+\lambda+a_2, x_1^2+(\lambda+1)x_3)$, si $P = Q$ y $\lambda = (x_1 + \frac{y_1}{x_1})$.

El cálculo de 2P

Si se observa el pseudocódigo de la siguiente rutina `calcula2P` (abajo), se notará que practicamente es una transcripción de la fórmula antes expuesta para el caso `P = Q`. Se reciben como parámetros las coordenadas del punto. Una vez que se inicializan las variables a cero, se verifica si el punto mismo no resulta ser él mismo su inverso, en caso de serlo, el resultado de la suma será el neutro del grupo, es decir, el punto al infinito (línea 11). En otro caso, se procede al cálculo de 2P mediante las fórmulas anteriores. El término `COEF_CUAD` se refiere al valor del coeficiente del término cuadrático en x de la ecuación de la curva, para este caso particular resulta ser cero, debido a que se está utilizando la ecuación $y^2 + xy = x^3 + 1$, aquí (línea 18) no tiene repercusión debido a que como ya se dijo, la suma no tiene costo.

```
0       int calcula2P(x, y)
1       PL32 *x, *y;
2       {PL32 x3[NUM_PALxPOL], y3[NUM_PALxPOL],
3             inv[NUM_PALxPOL],aux[NUM_PALxPOL],
4             lambda[NUM_PALxPOL], Uno[NUM_PALxPOL];
5       {
6       se inicializan todos los arreglos a cero
7       }
8       Uno = 1;
9       aux = x;
10      POL_Suma(aux, y); /* aux = x + y */
11      if (aux es igual a y) then return 0;
12      lambda = x;
13      aux = y;
14      POL_CInv(inv, x); /* inv = 1/x */
15      POL_Mult(aux, inv); /* aux = y/x */
16      POL_Suma(lambda, aux); /* lambda = x + y/x */
17      POL_Cuad(x3, lambda); POL_Suma(x3, lambda);
18      POL_Suma(x3, COEF_CUAD); /* x3 = lambda^2 + lambda + 0 */
19      POL_Cuad(y3, x); /* y3 = x^2 */
```

```
20      aux = lambda; POL_Suma(aux, Uno); /* aux = lambda + 1 */
21      POL_Mult(aux, x3); /* aux = x3(lambda + 1) */
22      POL_Suma(y3, aux); /* y3 = x^2 + x3(lambda + 1) */
23      x = x3; y = y3;
24      return 1;
25      }
```

Como puede verse todas las operaciones aritméticas que se utilizan para implementar a su vez la artimetica de puntos sobre la curva, realizan aritmética entre polinomios, es decir, en $\mathbb{F}_{2^{233}}$. De las fórmulas para el cálculo de una suma de puntos sobre una curva elíptica, independientemente del caso de que se trate, se observa que **el costo para el cálculo de la suma de puntos es aproximadamente igual al de una inversión más dos multiplicaciones y dos cuadrados**, si se considera que la suma no tienen costo.

Cuando $P \neq Q$ se transcribe la fórmula de manera análoga que para $P = Q$, la rutina correspondiente se nombró `sumaPmasQ`.

Suma k veces P

Ahora se expone el caso más general, cuando se desea sumar un punto P cualquiera consigo mismo un número k de veces. La rutina `sumakvecesP` recibe 5 parámetros, los dos primeros `kP_x` y `kP_y` (ver siguiente fragmento de pseudocódigo) se refieren a las coordenadas del punto resultante de sumar `k` (el quinto parámetro) veces el punto base, las coordenadas de este último son los parametros 3 y 4. Se precalculan los tres primeros múltiplos de `P` (líneas 5 y 6) con el fín de procesar k en parjeas de bits y por tanto eficientar la rutina. se guardan éstos en la variable `multiplosdeP`.

Piénsese en la cadena binaria que representa al número `k` dividida en parejas de bits. En cada iteración del cíclo se invoca en un par de ocasiones la rutina `calcula2P` (líneas 17 y 18), si se recorriera en grupos de 3 bits, entonces `calcula2P` se invocaría tres veces, etcétera. Las líneas 8 a 15 se encargan de contar el número de parejas de bits significativas en la palabra i-ésima de `k` el cual se guarda en `kiBits`, al mismo tiempo se descartan los pares de bits de ésta por la izquierda (línea 12) que no sean significativos y el resultado queda en `ki`. Posteriormente se recorre `kiBits` a partir del par de bits más significativo. En función del valor de los dos bits más significativos de

`kiBits` se suma el correspondiente múltiplo de `P`, línea 20. Antes de iniciar otra iteración, se descartan los dos bits más significativos. El número de veces que se ejecutará el `for` de la línea 16, corresponde al úmero de parejas significativas de bits que se formen en `k`.

```
0   void sumakvecesP(kPtobase_x,kPtobase_y,Ptobase_x,Ptobase_y,k)
1   B32 *kPtobase_x, *kPtobase_y, *Ptobase_x, *Ptobase_y, *k; {
2    B32 multiplosdeP[3][2][NUM_PALxPOL], ki;
3    int i;
4    unsigned int kiBits, j, s;
5    { Se guardan en multiplosdeP los puntos
6     P, 2P y 3P, es decir, el punto base, el doble y triple de este }
7    for(i = NUM_PALxclavePRI - 1; i >= 0; i--) {
8      ki = k[i];
9      kiBits = NUM_BITSxPAL;
10     if(i == (int)(NUM_PALxclavePRI - 1)) {
11       while(!LOS_2BMS(ki)) {
12         ki <<= 2;
13         kiBits -= 2;
14       }
15     }
16     for(j = 0; j < kiBits; j += 2, ki <<= 2) {
17       calcula2P(kPtobase_x, kPtobase_y);
18       calcula2P(kPtobase_x, kPtobase_y);
19       if((s = LOS_2BMS(ki)) != 0)
20         sumaPmasQ(kPtobase_x, kPtobase_y,
21           multiplosdeP[s - 1][0], multiplosdeP[s - 1][1]);
22     }
23   }
24  }
```

Con el propósito de ilustrar el algoritmo, se presenta el siguiente ejemplo. Supondremos que ya se han calculado los tres primeros múltiplos de `P`, por lo cual la atención se centrará en el ciclo que forman las líneas 16 a 22.

Ejemplo 3.2.1. Sean $P = (x, y)$ y el número $k = (45)_{10} = (101101)_2$ en base 10 y 2 respectivamente. Se tendrán tres iteraciones, debido al número

de parejas de bits significativos que posee k. En S se guardará la suma o mejor dicho el producto escalar de k por P, en b se guardará el par de bits más significativo de k en cada iteración.

Iteración 1 ($i = 0$, $k = 101101_2$ y $S = O$)

S = 4S = 0
b = val_2BMS(k) = 2 (Los dos bits más significativos valen 2_{10})
S = S + bP = 0 + 2P = 2P
k <<= 2;

Iteración 2 ($i = 2$, $k = 1101_2$ y $S = 2P$)

S = 4S = 4(2P) = 8P
b = val_2BMS(k) = 3 (Los dos bits más significativos valen 3_{10})
S = S + 3P = 8P + 3P = 11P
k <<= 2;

Iteración 3 ($i = 4$, $k = 01_2$ y $S = 11P$)

S = 4S = 4(11P) = 44P
b = val_2BMS(k) = 1 (Los dos bits más significativos valen 1_{10})
S = S + P = 44P + P = 45P
k <<= 2;

Se observa que al término el valor de S es 45P, lo cual es correcto.

3.3. Aritmética sobre $\mathbb{Z}_p$

La descripción de la aritmética para este tipo de campo será semejante a la que se dió para $\mathbb{F}_{2^{233}}$, presentándose sólamente algunas rutinas. La mayoría de las rutinas para este apartado tiene su inspiración en la biblioteca llamada RSAREF, creada por los laboratorios RSA en 1992.

Antes de iniciar la descripción del conjunto de rutinas, recordemos cuales son los parámetros con los que se realizó la implementación: (1) *la curva*

anómala $y^2 + xy = x^3 + 1$, (2) el polinomio irreducible $x^{233} + x^{74} + 1$, (3) *el campo* $\mathbb{F}_{2^{233}}$, (4) *un punto* $P_0 = (X, Y)$ sobre la curva del cual se dan las coordenadas en formato hexadecimal, tal como aparecen en el FIPS 186-2 [4]

$$X = 17232ba853a7e731af129f22ff4149563a419c26bf50a4c9d6eefad6126,$$
$$Y = 1db537dece819b7f70f555a67c427a8cd9bf18aeb9b56e0c11056fae6a3,$$

y (5) *el orden* de P_0 ($pP_0 = \mathcal{O}$) es el número primo $p = 3450873173395281893 - 717377931138512760570940988862252126328087024741343$, en formato decimal.

Representación

Para representar el orden del punto P_0, es decir, el número primo p, se requieren 232 bits, por lo cual **una cadena binaria de longitud 232 representa un elemento en** $\mathbb{Z}_p$. Para los elementos en $\mathbb{Z}_p$ también se necesitan 8 palabras de 32 bits cada una, debido a que el resultado de dividir 232 entre 32 es 7.25, sin embargo, al igual que los elementos de $\mathbb{F}_{2^{233}}$ en ocasiones es necesario calcular cuadrados, por lo que se duplica el espacio necesario para almacenar dicho cálculo, es decir, se requieren 16 palabras de 32 bits. En $\mathbb{F}_{2^{233}}$ los bits representan los coeficientes de potencias de una indeterminada x, mientras que en $\mathbb{Z}_p$ representan potencias de 2, en la siguiente declaración n está en $\mathbb{Z}_p$.

```
typedef unsigned long int AN_DIGITO;
#define MAX_AN_DIGITOS 16
#define AN_DIGITO n[MAX_AN_DIGITOS]
```

Un elemento en $\mathbb{Z}_p$ es un arreglo de 16 palabras del tipo `AN_DIGITO`, este tipo es una palabra de 32 bits. El prefijo AN es por Aritmética de Naturales.

De manera análoga a las rutinas POL_Cpy, POL_SetCero y POL_IShift en $\mathbb{F}_{2^{233}}$, existen para $\mathbb{Z}_p$, AN_Cpy, AN_SetCero y AN_IShift, las cuales son idénticas en funcionamiento e implantación, con la idea de que para $n \in \mathbb{Z}_p$ los bits representan potencias de 2 y no potencias de la indeterminada x para elementos en $\mathbb{F}_{2^{233}}$.

Suma

Para explicar esta rutina se hace una analogía con la suma habitual entre

números enteros en el sistema decimal. Como se dijo, un elemento en $\mathbb{Z}_p$ es un arreglo de 16 palabras de 32 bits cada una, ahora si se piensa que cada palabra "equivale" a un dígito decimal, entonces un elemento en $\mathbb{Z}_p$ posee "16 *dígitos*". *Con esta analogía en mente, el equivalente del número* 9 del sistema decimal es el número $\sum_{i=0}^{31} 2^i$. En el sistema decimal, cuando se suma cualquier dígito diferente de cero al número 9, se produce un acarreo, lo mismo ocurre cuando se suma 1 o hasta $\sum_{i=0}^{31} 2^i - 1$ al número $\sum_{i=0}^{31} 2^i$. Esta es la esencia de la rutina. Para que se cumpla la primera condición (línea 5, abajo) tiene que pasar que `A[i]`$= \sum_{i=0}^{31} 2^i$ y `acarreo = 1`, cuando esto sucede el acarreo se preserva, se actualiza la suma asignando directamente en la suma parcial el "dígito" `B[i]` y se continua con la siguiente iteración del ciclo (líneas 4 a 8) de la rutina. Esto equivaldría en el sistema decimal a sumar, por ejemplo, los dígitos 9 y 5 donde `acarreo = 1`, entonces 9 + acarreo = 0, con esto se cumple la condición mencionada, ya que la suma es menor al valor del acarreo y por tanto se asigna el valor de 5 a la suma parcial. Ahora bien, para que se cumpla la siguiente condición (línea 6), en este punto tendrá que ocurrir que tanto `A[i]` como `B[i]` no sean cero y que la suma de ambos sea menor a `B[i]` y donde `acarreo = 0`. Nuevamente el ejemplo en formato decimal es con 9 y 5, siendo `acarreo = 0`, entonces `9 + 5 = 4` y `4 < 5` (la condición de la línea 6), actualizándose `acarreo = 1` y siendo la suma parcial igual a 4. Finalmente, sumándose `A[i] + B[i] + acarreo` y no cumpliéndose ninguna de las dos condiciones, entonces `acarreo = 0` y se actualiza la suma parcial.

```
0   AN_Suma (A, B)
1   AN_DIGITO *A, *B;
2   { AN_DIGITO ai, acarreo = 0, C[MAX_AN_DIGITOS];
3    unsigned int i;
4    for (i = 0; i < MAX_AN_DIGITOS; i++) {
5     if ((ai = A[i] + acarreo) < acarreo) ai = B[i];
6       else if ((ai += B[i]) < B[i]) acarreo = 1;
7       else acarreo = 0; C[i] = ai;
8     }
9     AN_Cpy(A, C);
10  }
```

Resta

Esta operación involucra dos términos, el minuendo y el sustraendo, representados por `A` y `B` respectivamente. La rutina se plantea de manera análoga a la de la suma, sin embargo se utiliza el término préstamo en lugar del de acarreo. Para que la variable `prestamo` tome el valor de uno debe suceder que `A[i] < B[i]`, lo cual se presenta en el sistema decimal cuando el dígito correspondiente del sustraendo es mayor que el del minuendo. Asimismo, cuando `A[i] = 0` y `prestamo = 1` se satisface la primera condición y la variable `prestamo` no se altera, actualizándose la resta y continuando con la siguiente iteración del ciclo (líneas 5 a 12) de la rutina.

```
0   #define AN_MAX_DIGITO 0xffffffff
1   AN_Resta (A, B)
2   AN_DIGITO *A, *B;
3   { AN_DIGITO ai, prestamo = 0, C[MAX_AN_DIGITOS];
4    unsigned int i;
5    for (i = 0; i < MAX_AN_DIGITOS; i++) {
6     if ((ai = A[i] - prestamo) > (AN_MAX_DIGITO - prestamo))
7      ai = AN_MAX_DIGITO - B[i];
8     else if ((ai -= B[i]) > (AN_MAX_DIGITO - B[i]))
9       prestamo = 1;
10    else prestamo = 0;
11    C[i] = ai;
12    }
13    AN_Cpy(A, C);
14  }
```

Multiplicación

La rutina `AN_Mult` recibe dos parámetros `A` y `B`, los cuales se asume son números en $\mathbb{Z}_p$ y regresa en `A` el cálculo de la multiplicación de `A` y `B`. La manera de realizar el producto es similar a la multiplicación habitual entre enteros, se fija uno de los factores y el otro se recorre dígito a dígito, a partir del dígito que representa a las unidades hasta el dígito más significativo. Para este caso, un "dígito" será como ya se dijo, una palabra de 32 bits. Nótese que esto es lo que se hace en las líneas 7 y 8 abajo, donde el `A[i]` es el "dígito". La rutina que se encarga de realizar las multiplicaciones parciales es una subrutina llamada `AN_SumaMultDigit`, el nombre de esta última rutina sugiere el hecho de que se calcula un múltiplo de `B` (el múltiplo

está determinado por `A[i]`) y se suma en cada iteración gurdándose en la variable `t`, hasta que ya se recorrió `A` en su totalidad se obtiene la suma total, es decir, el producto de `A` y `B`. Al final se copia el contenido de `t` en `A` (línea 9). La rutina `AN_SumaMultDigit` recibe cinco parámetros A, B, C, D, y E, lo que realiza es `A = B + C*D`.

```
0   AN_Mult (A, B)
1   AN_DIGITO *A, *B;
2   { AN_DIGITO t[MAX_AN_DIGITOS];
3     unsigned int digitosA, digitosB, i;
4     for (i = 0; i < MAX_AN_DIGITOS; i++) t[i] = 0;
5     digitosA = num. de palabras signif. de 32 bits en A;
6     digitosB = num. de palabras signif. de 32 bits en B;
7     for (i = 0; i < digitosA; i++)
8       t[i+digitosB] += AN_SumaMultDigit(&t[i], &t[i], A[i],
                                          B, digitosB);
9     for (i = 0; i < MAX_AN_DIGITOS; i++) A[i] = t[i];
10  }
```

División

La rutina `AN_Div` calcula el cociente y residuo de dividir `A` entre `B`, son cuatro parámetros los que recibe `A`, `B`, `C` y `R`, en los dos últimos se guarda el cociente y el residuo de la división respectivamente.

Por otra parte se utiliza un rutina llamada AN_CmbBitn, que recibe dos parámetros, un elemento $n \in \mathbb{Z}_p$ y un entero c entre 0 y 231, se encarga de cambiar el c-ésimo bit de n. En la primera parte de la rutina se verifica que no se esté dividiendo entre cero y que el denominador no sea más grande que el numerador. Si ambas verificaciones fallasen entonces se realizan copias de `A` y `B` en `R` y `K` respectivamente, es decir, `R` será siempre el numerador y al final contendrá el residuo de la división, mientras que `K` siempre será el denominador. Por otra parte en `C` se tendrá el cociente, sin afectar los valores de `A` y `B`. Existe una subrutina que utiliza `AN_Div` llamada `AN_BMS`, la cual recibe dos parámetros, un elemento en $\mathbb{Z}_p$ y una variable de tipo entero, lo que realiza esta subrutina es devolver en la variable de tipo entero la posición del bit más significativo del primer parámetro. En cada iteración del ciclo en `AN_Div` se calcula con ayuda de la rutina `AN_BMS` la diferencia (`dif`) que existe entre la posición del bit más significativo de `A` y `B`, posteriormente se realiza

un corrimiento de K hacia la izquierda, línea 12 (es decir, se múltiplica K por 2^{dif}), se cambia en C el bit con posición dif, línea 17, esto último actualiza el cociente y antes de finalizar la iteración se realiza la resta entre R y K, lo cual garantiza que el ciclo termine. La ejecución terminará cuando el denominador A sea menor que B, al término R contiene al residuo y C al cociente.

```
0   AN_Div(A, B, C, R)
1   AN_DIGITO A[MAX_AN_DIGITOS], B[MAX_AN_DIGITOS],
2   C[MAX_AN_DIGITOS], R[MAX_AN_DIGITOS]; {
3    int dif, i;
4    unsigned int gr, gb;
5    AN_DIGITO k[MAX_AN_DIGITOS];
6    AN_SetCero(C); AN_Cpy(R, A);
7    { si el denominador es cero regresa error
8      si A < B regresa A, B, C = 0, R = 0 }
9    AN_Bms(R, gr); AN_Bms(B, gb);
10   while(R >= B) {
11    dif = gr - gb; AN_Cpy(K,B);
12    AN_IShift(K,dif);
13    if( K > R ){
14     dif--; AN_Cpy(K,B);
15     AN_IShift(K,dif);
16    }
17    AN_CmbBitn(C, dif);
18    AN_Resta(R,K);
19    if(R es igual a cero) gr = -1;
20    else AN_Bms(R,gr);
21   }
22  }
```

Inverso

Para el caso del inverso, se utiliza de manera directa el algoritmo euclideano extendido de la división, Menezes *et al* [29]. El problema de hallar el inverso de un número $A \in \mathbb{Z}_p$ consiste en encontrar otro Y en $\mathbb{Z}_p$ tal que $AY = 1 \bmod p$. A y p son primos relativos y el algoritmo euclideano extendido de la división asegura que existen X y Y en $\mathbb{Z}_p$ tales que $AY + pX = 1$, donde $pX = 0$ en $\mathbb{Z}_p$, por lo cual $AY = 1$. En la práctica no se guardan los valores que el algoritmo euclideano de la división va generando para X, ya que sólamente

interesa saber quien es Y. En cada ciclo se realiza una división y la parte entera del cociente se guarda en `C`, el residuo en `R` y más adelante a `B` se le asigna el valor de `R`, a su vez `A` toma el valor de `B`, con esto se asegura que el valor de `B` decrezca en cada iteración. La variable `Y2` (línea 3) es la que en todo momento contiene los valores calculados para `Y`, por lo cual, al final de la rutina esta contiene al inverso de A.

```
0   void AN_Inv(Inv, X)
1   AN_DIGITO *Inv, *X;
2   { AN_DIGITO Y[MAX_AN_DIGITOS], Y1[MAX_AN_DIGITOS],
3     Y2[MAX_AN_DIGITOS], Q[MAX_AN_DIGITOS],
4     R[MAX_AN_DIGITOS], A[MAX_AN_DIGITOS], B[MAX_AN_DIGITOS];
5     int i;
6     AN_Cpy(A, p); AN_Cpy(B, X);
7     AN_SetCero(Y1); Y1[0] = 1;
8     AN_SetCero(Y2);
9     while(B != 0)) {
10      AN_Div(A,B,Q,R);
11      AN_Cpy(Y, Y2);
12      AN_Mult(Q,Y1);
13      AN_Resta(Y,Q);              /* Y = Y2 - Q*Y1 */
14      AN_Cpy(A, B); AN_Cpy(B, R);
15      AN_Cpy(Y2, Y1); AN_Cpy(Y1, Y);
16    }
17    AN_Cpy(Inv, Y2);
18  }
```

3.4. La biblioteca

En esta sección se describe la biblioteca, de la cual se hará una descripción general, sin los detalles de la implementación de cada una de las opciones con las que cuenta ésta. Posteriormente, en la siguiente y última sección de este capítulo se muestra la operatividad de dichas opciones mediante la exposición de ejemplos.

Los elementos que conforman la máquina de cifrado y firma digital son: (1) el polinomio irreducible $x^{233} + x^{74} + 1$; (2) el campo $\mathbb{F}_{2^{233}}$; (3) la curva

$y^2 + xy = x^3 + 1$; (4) el punto $P_0 = (X, Y)$; y (5) p el orden del punto.

Cuando se desea establecer una comunicación basada en un sistema de cifrado de clave pública, el primer paso es verificar si los participantes ya cuentan con un par de claves. Ahora bien, si alguna o todas las entidades no poseen un par de claves, entonces es necesario que cada entidad que desee participar genere localmente su correspondiente par de claves, una vez hecho lo anterior, cada entidad debe publicar en algun lugar la clave pública y al mismo tiempo deberá mantener en secreto su clave privada. Una vez concluidos estos procedimientos por parte de cada uno de los participantes, ya puede iniciarse el intercambio de mensajes cifrados y/o firmados.

Generación de claves

Cuando una entidad **A** desea obtener su par de claves, el criptosistema de manera local en la máquina de **A** se encarga de ejecutar los siguientes pasos:

1. Se escoge de manera aleatoria un número $d_A \in [1, p-1]$.
2. Se calcula el punto $Q_A = d_A P_0$.
3. La clave pública será el punto Q_A. **La clave pública será de 466 bits**, ya que las entradas del punto Q_A están en $\mathbb{F}_{2^{233}}$.
4. La clave privada es el número d_A. d_A es un número en $\mathbb{Z}_p$, por lo cual **la longitud de la clave privada es de 232 bits.**

Observando el algoritmo que genera el par de claves se puede notar que para obtener la clave pública Q_A, es necesario sumar d_A veces el punto P_0 consigo mismo. Recuérdese que los puntos P_0 y Q_A son públicos y el primero de ellos es el generador del grupo de puntos en la curva cuyo orden es el número primo p de 70 dígitos decimales, ver sección 2.1 del capítulo previo. De lo anterior, si alguien quisiera deducir la clave privada d_A a partir de la clave pública Q_A, deberá resolver el PLDCE, esta es la relación que guardan ambas claves, ver definición 1.4.3.

El punto Q_A se publica en algún sitio, mientras que el número d_A será resguardado de manera que sólamente pueda ser usado por su dueño.

Con el propósito de evitar el mal uso de la clave privada por parte de otras personas, se utiliza DES (*Data Encryption Standard*) para cifrar d_A y se recomienda depositarla en una unidad de almacenamiento removible, como puede ser un disco flexible o una tarjeta inteligente.

El DES es un sistema de cifrado simétrico, el cual ya es obsoleto, ya que el tamaño de la clave es de 56 bits, lo cual ya no es seguro en la actualidad. Se utilizó DES debido a que se contaba con los fuentes, sin embargo, al momento de usarla se pensó que dicho algoritmo bien puede sustituirse con una versión mejorada del DES y puede ser considerada como una segunda versión de la biblioteca. Dentro de los criptosistemas simétricos a usarse en lugar de DES podrían ser 3DES o AES (*Advanced Encryption Standard*). Este último surgió como el resultado de una convocatoria lanzada a nivel mundial en septiembre de 1997 por el Instituto Nacional de Estándares y Tecnología de E.U. (NIST, de sus siglas en Inglés), con el propósito de desarrollar un Estándar de Cifrado Avanzado (AES). La meta primordial de este proyecto para la NIST fue desarrollar un Estándar Federal en el Proceso de la Información (FIPS), el cual especifique un algoritmo de cifrado que sea capaz de proteger la información de manera satisfactoria y que se convierta en uno de los mecanismos de protección de datos electrónicos en el siglo XXI. El 9 de agosto de 1999 el Secretario de Comercio de E.U. anunció que el proyecto de origen belga llamado *Rijndael* [2] era el ganador de la convocatoria. Si se desea conocer con más detalle este proyecto se puede visitar el sitio *http://csrc.nist.gov/encryption/aes/rijndael.*

El hecho de proteger d_A es debido a que ésta puede alojarse en una PC o en algún servidor dentro de una red y únicamente mediante el conocimiento de una contraseña específica podrá usarse. Como se dijo, DES utiliza una clave con tamaño de 56 bits, pues bien, la contraseña que se pide debe tener una longitud de 7 caracteres ASCII estándar, cada caracter se representa con 8 bits, esto justifica la longitud de 56 bits. Vale la pena mencionar, que una vez que ya se descifró la clave privada, por motivos de seguridad, esta nunca es manejada directamente en disco o en algún otro dispositivo que no sea la memoria.

Cifrado

Una entidad **A** desea enviar el mensaje m a **B**, para lo cual debe realizar

lo siguiente:.

1. Se recupera la clave pública de **B**, es decir Q_B.

2. El mensaje m se representa como una pareja ordenada $(m_1, m_2) \in \mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$. Recordemos que las entradas de la pareja ordenada son cadenas binarias de longitud 233 y los caracteres del código ASCII son cadenas binarias de longitud 8, en este sentido es posible "acomodar" texto en $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$, claro, estas parejas no necesariamente son puntos sobre la curva.

3. Se escoge de manera aleatoria un número $k \in [1, p-1]$.

4. Se calcula el punto $kP_0 = (x_1, y_1)$.

5. Se calcula el punto $kQ_B = (x_2, y_2)$.

6. Se genera el par ordenado (x_4, y_4) a partir de x_2 y x_2^3 de la siguiente manera. Se toman los primeros 117 bits de x_2 y los últimos 116 bits de x_2^3 para formar x_4. De manera simétrica se forma y_4, tomando los primeros 117 bits de x_2^3 y los últimos 116 bits de x_2. El par ordenado (x_4, y_4) no necesariamente es un punto sobre la curva. Lo anterior no obedece a alguna consideración especial respecto a 117 y 116 bits, lo que se quiere es un proceso que sea invertible al momento del descifrado.

7. Se forman $c_1 = (m_1 + y_2)\, x_4$ y $c_2 = (m_2 + x_2)\, y_4$, **A** entrega como mensaje cifrado $c = (x_1, y_1, c_1, c_2)$.

El primer paso es recuperar la clave pública de la entidad a la cual se desea enviar un mensaje cifrado, en este caso para **B**.

El mensaje m se procesará de manera que el texto quede representado como parejas ordenadas en $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$, en este sentido en una pareja ordenada se tienen 466 bits, por lo que una cadena de 58 caracteres está representada por un elemento en $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$.

Descifrado

1. La entidad **B** recibe el mensaje $c = (x_1, y_1, c_1, c_2)$ que envió **A**.

2. Utilizando su clave privada **B** calcula $d_B(x_1, y_1) = (x_2, y_2)$.

3. Se calcula la pareja ordenada (x_4, y_4) de manera idéntica a como lo hizo **A**.

4. Se calcula $m_1 = c_1 x_4^{-1} + y_2$ y $m_2 = c_2 y_4^{-1} + x_2$, con lo que se recupera el texto plano.

Es importante hacer notar que en el paso 5 del cifrado el punto resultante es $kQ_B = (kd_B)P_0 = (x_2, y_2)$, donde k y d_B están en $[1, p-1]$, por lo cual kQ_B no es el punto al infinito y por lo tanto sus entradas no son cero, de aquí que existan los inversos para x_4 y y_4.

Lo que se observa es que la seguridad del cifrado se basa precisamente en la necesidad de resolver el PLDCE. Dado el texto cifrado c, para recuperar el texto plano es necesario saber quien es la pareja ordenada (x_2, y_2) y para esto se requiere saber cual es la clave privada de **B**, es decir, se tiene que resolver el PLDCE.

Firma digital

Recordemos nuevamente, una firma digital es una liga entre un mensaje M y la identidad de quien firma, de tal forma que si el mensaje cambia la firma también. Por medio de la firma digital se verifica integridad (el mensaje no ha sido alterado), autenticación (se conoce la identidad de quien firma), si el mensaje viaja cifrado se tiene confidencialidad y no-repudio (la entidad firmante no puede negar su participación como firmante del mensaje).

Antes de describir el proceso de firma digital, se dirá brevemente que es una función *hash* o de resumen. Como veremos este tipo de funciones intervienen en el proceso de firma digital de documentos.

Una función *hash* mapea cadenas finitas de longitud arbitraria en cadenas de longitud fija, digamos de n bits. Para un dominio D y un rango R con $h : D \to R$ y $|D| > |R|$, la función resulta varios-a-uno, implicando la existencia de *colisiones*, es decir, un par de entradas pueden tener una salida o resultado *hash* iguales. Sin embargo, si h se viera restringida a tomar cadenas de longitud t $(t > n)$ y h fuera "aleatoria" en el sentido de que todas las salidas fueran equiprobables, entonces dos cadenas elegidas

aleatoriamente tendrían una probabilidad de 2^{-n} de tener el mismo valor de salida (idependientemente de t), Menezes *et al* [29]. A continuación se define (Menezes *et al* [29]) lo que es una función *hash*.

Definición 3.4.1. Una *función hash o de resumen* es una función h que tiene como mínimo las siguientes dos propiedades:

1. *compresión* - h mapea una entrada x de longitud en bits arbitraria, en una salida $h(x)$ de longitud n en bits fija.

2. *fácil de computar* - dada h y una entrada x, $h(x)$ es fácil de computar.

La esencia de utilizar a las funciones hash en el proceso de firma digital es que estas sirven para representar de manera compacta la imagen de una cadena (mensaje) de entrada (a esta representación se le llama en ocasiones *la huella digital* o también *el resumen del mensaje*), pudiendo ser utilizable como si fuera un identificador unívoco de dicha cadena.

La relación que guarda el uso de la función de resumen con el proceso de firma digital puede ser más clara en la figura 3.1. Al término del resumen se tiene una cadena de 16 bytes (128 bits), la cual es posteriormente firmada con la clave privada del emisor.

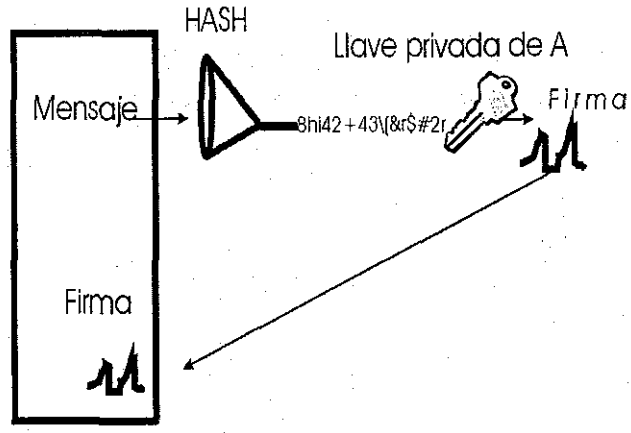

Figura 3.1: Firma digital de un documento por parte de A.

Como se observa, el proceso de firma de un mensaje se compone de dos pasos: el primero es la ejecución del proceso de resumen sobre el mensaje, el cual puede ser de cualquier tamaño; el segundo, la cadena resultante del resumen se cifra con la clave privada de quien firma el documento y se adjunta al mensaje original que después se envía al destinatario.

El algoritmo de resumen o *hash* que se usa en este trabajo es MD5. Pueden encontrarse algunas implementaciones hechas en C (versión original), en C++ o Pearl entre otras, en

http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html.

El algoritmo que se utiliza para la generación de la firma digital es el que se prescribe en el FIPS 186-2 [4] para el caso de utilizar curvas elípticas y se denomina **Algoritmo Estándar de Firma Digital para Curvas Elípticas (ECDSA**, por sus siglas en Inglés).

Generación de firma

Para firmar un mensaje m, una entidad **A** deberá realizar lo siguiente:

1. Elíjase de manera aleatoria un número entero $k \in [1, p-1]$.
2. Calcular $kP_0 = (x_1, y_1)$ y $r = x_1 \bmod p$. Aquí x_1 debe considerarse como un número entero, dado que una cadena de 233 bits representa a $x_1 \in \mathbb{F}_{2^{233}}$, dicha cadena también puede verse como un número en base dos y no como un polinomio. Si $r = 0$, entonces regresar al paso 1 (si $r = 0$ entonces la ecuación que define a la firma $s = k^{-1}(h(m) + d_A r)$ del paso 4 no podría involucrar a la clave privada de quien firma).
3. Calcular $k^{-1} \bmod p$.
4. Calcular $s = k^{-1}(h(m) + d_A r) \bmod p$, donde h es la función de resumen (*hash*) MD5. El valor de $h(m)$ se obtiene simplemente pasando como parámetro a MD5 el mensaje y se obtiene una cadena de 16 caracteres o 128 bits, la cual puede verse como un numero en base dos de 128 bits.
5. Si $s = 0$, entonces regresar al paso 1 (si $s = 0$, no existiría $s^{-1} \bmod p$, s^{-1} es requerido para la verificación de la firma digital).

6. La firma digital que corresponde al mensaje m es la pareja (r, s).

Dado que $r, s \in \mathbb{Z}_p$, entonces **la firma tiene un tamaño de 464 bits.**

Al momento de establecer comunicación basada en criptografía de clave pública con un conjunto reducido de personas, es posible entregar una copia de su clave pública de manera directa, mediante un disco flexible, por ejemplo. Este esquema de distribución de clave pública puede funcionar cuando el número de personas es reducido, sin embargo, cuando se trata de miles de personas esto se vuelve un problema y ya no resulta seguro, debido a que álguien puede extraviar o prestar el disco, además de que resulta impráctico dar un disco por clave. En estos casos es cuando se emplea lo que mencionamos al inicio del capítulo 1 y se conoce como una *Infraestructura de clave Pública* (PKI) la cual resulta una solución al problema planteado.

Los servicios que ofrece una PKI están basados en la criptografía de clave pública, dentro de los cuales se tienen la generación-distribución de claves, cifrado/descifrado, ensobretado y firma digital. El componente principal de una PKI es la llamada Autoridad Certificadora AC (*Certify Authority*), las dos funciones principales de una AC son la expedición y revocación de certificados digitales. Un certificado digital es un conjunto de datos (que incluyen a la clave pública) que identifican a una entidad y a los que se les anexa la firma de una autoridad en la cual se confía y a la que se otorga reconocimiento por una comunidad de usuarios o entidades. Entonces, un certificado digital es una liga entre una clave pública y la identidad de su dueño, dicha liga contiene la firmada digital de la AC. En el contexto de la PKI cuando se habla de certificado digital equivale a clave pública firmada por la AC, es decir, son sinónimos.

Cuando un grupo de entidades decide de común acuerdo utilizar los servicios de una PKI, en el fondo lo que están aceptando es *confiar* en la AC asociada a esa PKI. Para simplificar la exposición se asumirá cuando se hable de una PKI, que ésta tiene sólamente una AC.

Como se dijo, cada entidad genera de manera local su par de claves, entonces el procedimiento que sigue como usuario de la PKI es obtener la *certificación* por parte de la AC. Esto último no es otra cosa que la expedición del certificado digital correspondiente por parte de la AC para

una determinada entidad. La certificación es la manera como las entidades usuarias de una PKI *confían* que sus comunicaciones sean seguras entre ellas.

Al implantar una PKI por vez primera la AC también genera su propio par de claves, a diferencia de los usuarios, en el caso de la AC se emplean tamaños de clave dos o tres veces más grandes, uno de los propósitos de lo anterior es evitar que se intente suplantar a la AC. Para el caso excepcional de la AC, esta se autogenera su propio certificado.

Ahora se expone una situación que justifica la necesidad de utilizar una PKI. Imagínese que una persona, la cual llamaremos **A**, desea comunicarse utilizando criptografía de clave pública con otra persona **B**, a la cual no conoce. **A** constata primeramente que posea una copia auténtica de la clave pública de **B**, si no fuera el caso entonces tiene dos opciones (en caso de que **A** y **B** sean usuarios de una PKI), solicitar a **B** su certificado o pedirlo a la AC. A partir de que **A** ya cuenta con el certificado y por tanto con la clave pública de la otra persona, es posible iniciar la comunicación usando criptografía de clave pública. Cuando se dice que ya se cuenta con el certificado digital de una entidad lo que quiere decirse es que se realizó un proceso de verificación del mismo. De esto último se desprende la confianza de la que hablamos, la cual radica en el hecho de verificar por parte de **A** la validez del certificado. Con el fin de verificar la validez de un certificado se debe comprobar la autenticidad de la firma digital de la AC que acompaña al certificado.

Por otra parte, si el dueño de una clave privada sospecha que alguien más está utilizándola, éste deberá notificarlo a la AC, la cual invalidará o revocará el certificado digital correspondiente y lo hará público, es decir, el par de claves que corresponde a dicha persona ya no es válido.

Verificación de la firma

Con el propósito de que una entidad **B** verifique la firma (r, s) que otra entidad **A** produjo, se deberá:

1. Conseguir una copia auténtica de la clave pública de **A**, Q_A.
2. Corroborar que los enteros $r, s \in [1, p-1]$.
3. Calcular $w = s^{-1} \bmod p$ y $h(m)$.

4. Calcular $u_1 = h(m)\, w \bmod p$ y $u_2 = rw \bmod p$.

5. Calcular $u_1 P_0 + u_2 Q_A = (x_0, y_0)$ y $v = x_0 \bmod p$.

6. Aceptar la firma si y sólo si $v = r$.

Al firmar se tiene $s = k^{-1}(h(m) + d_A r) \bmod p$, por lo que $w = s^{-1} = \frac{1}{k^{-1}(h(m)+dr)} \bmod p$. La suma de puntos en (5), se detalla como sigue

$$u_1 P_0 + u_2 Q_A = (h(m)\, w)\, P_0 + (rw)\, Q_A = P_0 (h(m)\, w + rwd) = P_0 w (h(m) + rd) = \frac{P_0}{k^{-1}(h(m)+dr)}(h(m) + rd) = P_0(\tfrac{1}{k^{-1}}) = kP_0.$$

Si **A** realmente utilizó su clave privada para producir una firma digital, lo único que **B** necesita para verificarla es conseguir una copia auténtica de la clave publica Q_A de **A**. Si en el trayecto alguien modificara el mensaje y la firma permaneciera intacta, la verificación fallaría debido a que la función hash arrojará otro valor y por ende la firma será distinta a la que acompaña al archivo.

Ensobretado

La idea que se tiene de *ensobretado* en el contexto que aquí se usará proviene del hecho de guardar documentos de manera confidencial dentro de un sobre de papel, es decir, sólamente quien tenga autorización podrá abrirlo y ver los documentos. En este caso, el sobre podría contener algún tipo de sello que indique si alguién intentó abrirlo. Esta misma situación la trasladamos a un *sobre digital*.

Una entidad **A** desea enviar a otra entidad **B** un documento o archivo **m** de texto (puede ser una imagen) plano dentro de un "sobre" s (en realidad s es un archivo), para lo cual deberá realizar los siguientes pasos:

1. Se firma **m** con la clave privada de **A**.

2. Se genera de manera aleatoria una clave L para el sistema de cifrado simétrico, en este caso se trata del DES.

3. Se cifra con L el mensaje **m**.

4. Se cifra L con la clave pública de **B**.

5. El sobre s contendrá el cifrado de L y $\mathbf{m}$, junto con la firma, en este orden, se envía s.

Apertura y verificación de sobre

Una entidad **B** recibe de parte de otra entidad **A** un sobre o archivo s, con el propósito de verificar el sobre, **B** deberá realizar lo siguiente.

1. Del archivo o sobre s se lee la parte que corresponde al cifrado de L, procediendo a su descifrado usando la clave privada de **B**, con lo que se recupera la clave simétrica L.

2. Se descifra del sobre s, con DES y L, la parte correspondiente a $\mathbf{m}$, recuperándose éste.

3. Se verifica la firma que contiene s usando la clave pública de **A**.

Para finalizar esta sección y con el propósito de dar una idea del poder computacional que se requiere para resolver el PLDCE, utilizando el mejor algoritmo conocido hasta ahora, el método-ρ de Pollard [30], se presenta la siguiente tabla. Donde n es el orden del punto generador de $E(\mathbb{F}_{2^m})$ ($E(\mathbb{F}_{2^m})$ es cíclico) y el termino MIPS se refiere a que una computadora procese un-millón-de-instrucciones-por-segundo, año MIPS denota la utilización de una computadora ininterrumpidamente durante un año procesando un MIPS.

Tamaño del campo (en bits)	**Tamaño de n (en bits)**	**Años MIPS**
163	160	8.5×10^{11}
191	186	7.0×10^{15}
239	234	1.2×10^{23}
359	354	1.3×10^{41}
431	426	9.2×10^{51}

Tiempo necesario para resolver el PLDCE usando el método-ρ de Pollard y una computadora trabajando a un MIPS, Koblitz *et al* [22].

Por ejemplo, si se tuvieran 10,000 computadoras cada una con una capacidad de procesamiento de 1,000 MIPS y $n \approx 2^{160}$, entonces el tiempo que tomará calcular la solución de una instancia del PLDCE será de 85,000 años, Koblitz *et al* [22]

3.5. Tiempos de ejecución

Los tiempos que a continuación se reportan corresponden a los que se generaron usando un equipo con procesador tipo Céleron a 433 MHz y 64 MB en RAM, utilizando los elementos mencionados en el capítulo anterior y que recomienda el FIPS 186-2. En López y Dahab [25], se reportan velocidades de ejecución para criptosistemas basados en el PLDCE, los cuales fueron realizados por diversas personas y grupos de trabajo, asimismo, se emplean diferentes equipos, plataformas, lenguajes (ensamblador, C, C++). Realizando una comparativa equivalente de plataforma y equipo, la eficiencia en las rutinas de esta tesis es la siguiente. Para el caso de la aritmética sobre $\mathbb{F}_{2^{233}}$, se tiene aquí un tiempo 3 veces más lento, para la aritmética en $E(\mathbb{F}_{2^{233}})$, aquellas son 5 veces más rápidas y finalmente, para las rutinas de la biblioteca presentadas en este trabajo, estas son 10 veces más lentas. Con $\mathbb{Z}_p$ los registros son los que coorresponden a la implementación hecha en RSAREF.

Operación	Microsegundos
Inverso	505
Multiplicación	151
Suma	19
Cuadrado	41

Operaciones en $\mathbb{F}_{2^{233}}$.

Operación	Microsegundos
Inverso	678
Multiplicación	5
División	108
Resta	0.3
Suma	0.4

Operaciones en $\mathbb{Z}_p$.

Operación	Microsegundos
Doblar un punto	1046
Suma de puntos	857

Operaciones en $E(\mathbb{F}_{2^{233}})$.

Operación	Parámetros	Segundos
Generación de claves	s/p	0.25
Firma digital	Mensaje de 175,000 bytes	0.25
Verificación de firma	Mensaje de 58 bytes	0.52

Operaciones en la biblioteca.

3.6. Ejemplos

En esta sección se presentan un par de ejemplos de como funciona el criptosistema, el primero de ellos es con un campo muy modesto, este tiene como propósito servir como preliminar y para fijar ideas, posteriormente se muestra un ejemplo empleando los parámetros del FIPS 186-2 que finalmente se utilizaron.

3.6.1. Ejemplo de pizarrón

Para ésta subsección, el campo es $\mathbb{F}_{2^3}$, la curva elíptica anómala $y^2+xy = x^3+x^2+1$. el polinomio irreducible x^3+x+1, el punto $P_1 = (x+1, x+1)$, el número de puntos es $E(\mathbb{F}_{2^3}) = 14$ y el orden de P_1 es 7, es decir, $p = 7$. La siguiente tabla muestra el grupo completo de puntos.

$+$	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}
P_1	P_7	O	P_8	P_3	P_{12}	P_{10}	P_4	P_2	P_5	P_9	P_6	P_{13}	P_{11}
P_2	O	P_8	P_4	P_7	P_9	P_{11}	P_1	P_3	P_{10}	P_6	P_{13}	P_5	P_{12}
P_3	P_8	P_4	P_1	O	P_6	P_{12}	P_2	P_7	P_{11}	P_{13}	P_5	P_{10}	P_9
P_4	P_3	P_7	O	P_2	P_{11}	P_5	P_8	P_1	P_{13}	P_{12}	P_9	P_6	P_{10}
P_5	P_{12}	P_9	P_6	P_{11}	P_4	O	P_{13}	P_{10}	P_7	P_1	P_2	P_3	P_8
P_6	P_{10}	P_{11}	P_{12}	P_5	O	P_3	P_9	P_{13}	P_2	P_8	P_4	P_1	P_7
P_7	P_4	P_1	P_2	P_8	P_{13}	P_9	P_3	O	P_{12}	P_5	P_{10}	P_{11}	P_6
P_8	P_2	P_3	P_7	P_1	P_{10}	P_{13}	O	P_4	P_6	P_{11}	P_{12}	P_9	P_5
P_9	P_5	P_{10}	P_{11}	P_{13}	P_7	P_2	P_{12}	P_6	P_1	O	P_8	P_4	P_3
P_{10}	P_9	P_6	P_{13}	P_{12}	P_1	P_8	P_5	P_{11}	O	P_2	P_3	P_7	P_4
P_{11}	P_6	P_{13}	P_5	P_9	P_2	P_4	P_{10}	P_{12}	P_8	P_3	P_7	O	P_1
P_{12}	P_{13}	P_5	P_{10}	P_6	P_3	P_1	P_{11}	P_9	P_4	P_7	O	P_8	P_2
P_{13}	P_{11}	P_{12}	P_9	P_{10}	P_8	P_7	P_6	P_5	P_3	P_4	P_1	P_2	O

El grupo $E(\mathbb{F}_{2^3})$.

Con $P_1 = (x+1, x+1)$, $P_2 = (x+1, 0)$, $P_3 = (x^2+1, x^2+1)$, $P_4 = (x^2+1, 0)$, $P_5 = (x^2, x+1)$, $P_6 = (x^2, x^2+x+1)$, $P_7 = (x^2+x+1, x^2+x+1)$, $P_8 = (x^2+x+1, 0)$, $P_9 = (x, x^2+1)$, $P_{10} = (x, x^2+x+1)$, $P_{11} = (x^2+x, x+1)$, $P_{12} = (x^2+x, x^2+1)$ y $P_{13} = (0, 1)$

Generación de claves

Ejemplo 3.6.1. Supóngase que las entidades **A** y **B** desean establecer comunicación mediante criptografía de clave pública, para lo cual cada entidad genera su correspondiente par de claves.

1. Supóngase que **A** elige aleatoriamente $5 \in [1, 6]$.
2. Se calcula $Q_A = 5\, P_1 = P_8$.
3. La clave pública de **A** es P_8.
4. La clave privada de **A** es 5.

En el caso de **B**.

1. Supóngase que **B** elige aleatoriamente $3 \in [1, 6]$.
2. Se calcula $Q_B = 3\, P_1 = P_4$.
3. La clave pública de **B** es P_4.
4. La clave privada de **B** es 3.

Cifrado de un mensaje

Ejemplo 3.6.2. Una vez que cada entidad generó su correspondiente par de claves, supongamos que la entidad **A** desea enviar a **B** el mensaje m de manera cifrada.

1. Se recupera la clave pública de **B**, es decir P_4, imaginemos que **B** la proporcionó directamente.
2. Supóngase que el mensaje $m = (m_1, m_2) = (x+1, 1) = (011, 001)$ será el que envíe **A**.

3. Supóngase que se elige de manera aleatoria $4 \in [1,6]$.
4. Se calcula el punto $4\,P_1 = P_3 = (x^2+1, x^2+1) = (x_1, y_1) = (101, 101)$.
5. Se calcula el punto $4\,P_4 = P_8 = (x^2+x+1, 0) = (x_2, y_2) = (111, 000)$.
6. Se calcula $x_2^3 = (x^2+x+1)^3 \bmod x^3+x+1 = x = (010)$. Se forma x_4 tomando los 2 primeros bits de x_2 y el último de x_2^3, resultando en $x_4 = (011) = x+1$. De manera simétrica se forma a y_4 tomando los 2 primeros bits de x_2^3 y el último de x_2, obteniéndose $y_4 = (110) = x^2+x$. Ahora se generan $c_1 = (m_1 + y_2)\,x_4 = (x+1+0)\,(x+1) = (x+1)^2 = x^2+1 = (101)$ y $c_2 = (m_2 + x_2)\,y_4 = (1+x^2+x+1)\,(x^2+x) = (x^2+x)^2 = x = (010)$.
7. **A** envía
   $c = (x_1, y_1, c_1, c_2) = (x^2+1, x^2+1, x^2+1, x) = (101, 101, 101, 010)$.

Descifrado de un mensaje

Ejemplo 3.6.3. Una vez que la entidad **B** recibe de **A** un mensaje cifrado $c = (x_1, y_1, c_1, c_2)$, **B** procede como sigue.

1. **B** clacula $3\,(x_1, y_1) = (x_2, y_2) = (111, 000)$ usando su clave secreta, en este caso 3.
2. **B** calcula el par ordenado (x_4, y_4) de manera idéntica a como lo hizo **A**.
3. Finalmente **B** calcula
   $m_1 = c_1 x_4^{-1} + y_2 = (x^2+1)\,(x+1)^{-1} + 0 = (x^2+1)\,(x^2+x) = x+1$,
   $m_2 = c_2 y_4^{-1} + x_2 = x\,(x^2+x)^{-1} + x^2+x+1 = x\,(x+1) + x^2+x+1 = 1$.

Una firma digital

Ejemplo 3.6.4. La entidad **A** desea firmar un mensaje m y enviárselo a **B**.

1. Supóngase que **A** escoge de manera aleatoria el número $6 \in [1,6]$.
2. **A** calcula $6P_1 = P_2 = (x+1, 0) = (011, 000)$ y $r = 011 = 3 \bmod 7 = 3$.

3. **A** calcula $6^{-1} \bmod 7 = 6$.

4. Supóngase que $h(m) \bmod 7 = 2$. $s = k^{-1}(h(m) + d_A r) \bmod 7 = 6(2 + 5 * 3) \bmod 7 = 4$.

5. La firma digital que **A** produce para el mensaje m es $(3, 4)$.

Verificación de una firma digital

Ejemplo 3.6.5. La entidad **B** recibe por parte de **A** la firma $(r, s) = (3, 4)$ junto con el correspondiente mensaje m, para su verificación.

1. La entidad **B** consigue una copia de la clave pública de **A**, P_8. Nuevamente imagínese que **A** la proporcionó directamente.

2. Los números 3 y 4 están en $[1, 6]$.

3. B calcula $w = s^{-1} \bmod n = 4^{-1} \bmod 7 = 2$ y $h(m) = 2$.

4. B calcula $u_1 = h(m)w \bmod p = 2 * 2 \bmod 7 = 4$, y $u_2 = rw \bmod p = 3 * 2 \bmod 7 = 6$, $u_1 P + u_2 Q_A = 4P_1 + 6P_8 = 4P_1 + 6 * (5P_1) = 4P_1 + 2P_1 = 6P_1 = P_2 = (x_0, y_0)$, y $v = x_0 \bmod n = 3 \bmod 7 = 3$.

5. Se acepta la firma, ya que $v = r$.

3.6.2. Ejemplo con los parámetros del FIPS 186-2

En el capítulo 2 se listaron los elementos con los que se construyó el criptosistema de esta tesis, nuevamente se enlistan.

Tenemos el polinomio irreducible $x^{233} + x^{74} + 1$, al campo $\mathbb{F}_{2^{233}}$, cuyos elementos son polinomios de grado menor a 233 y con coeficientes en $\mathbb{F}_2$. También está la curva, cuya ecuación es $y^2 + xy = x^3 + 1$. Otro de los elementos es el punto base P, en aquel capítulo se dieron sus coordenadas en formato hexadecimal, como aparecen en el FIPS 186-2, ahora las mostramos como realmente están representadas en $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$. El ejemplo anterior tiene como intención que el lector pueda seguir los cálculos, el que se haya utilizado una curva distinta no es relevante para la exposición, lo importante es que se visualice y se verifique el funcionamiento del criptosistema, el siguiente

ejemplo corresponde a uno que podemos considerarlo en la práctica como real.

La abscisa del punto P es,

$$x+x^2+x^5+x^8+x^{13}+x^{14}+x^{16}+x^{18}+x^{19}+x^{21}+x^{23}+x^{24}+x^{25}+x^{26}+x^{27}+x^{29}+x^{30}+x^{31}+x^{33}+x^{34}+x^{35}+x^{37}+x^{38}+x^{40}+x^{42}+x^{43}+x^{44}+x^{47}+x^{50}+x^{51}+x^{54}+x^{57}+x^{59}+x^{64}+x^{66}+x^{68}+x^{69}+x^{70}+x^{71}+x^{72}+x^{73}+x^{75}+x^{77}+x^{78}+x^{81}+x^{86}+x^{87}+x^{88}+x^{91}+x^{92}+x^{98}+x^{101}+x^{103}+x^{104}+x^{105}+x^{109}+x^{110}+x^{112}+x^{114}+x^{116}+x^{119}+x^{122}+x^{124}+x^{130}+x^{132}+x^{133}+x^{134}+x^{135}+x^{136}+x^{137}+x^{138}+x^{139}+x^{141}+x^{145}+x^{148}+x^{149}+x^{150}+x^{151}+x^{152}+x^{155}+x^{157}+x^{160}+x^{164}+x^{165}+x^{166}+x^{167}+x^{169}+x^{171}+x^{172}+x^{176}+x^{177}+x^{180}+x^{181}+x^{182}+x^{185}+x^{186}+x^{187}+x^{188}+x^{189}+x^{190}+x^{193}+x^{195}+x^{196}+x^{197}+x^{200}+x^{202}+x^{207}+x^{209}+x^{211}+x^{212}+x^{213}+x^{215}+x^{217}+x^{220}+x^{221}+x^{225}+x^{228}+x^{229}+x^{230}+x^{232},$$

mientras que su ordenada es,

$$1+x+x^5+x^7+x^9+x^{10}+x^{13}+x^{14}+x^{15}+x^{17}+x^{19}+x^{20}+x^{21}+x^{22}+x^{23}+x^{25}+x^{26}+x^{28}+x^{30}+x^{36}+x^{40}+x^{46}+x^{47}+x^{53}+x^{54}+x^{55}+x^{57}+x^{58}+x^{60}+x^{62}+x^{64}+x^{65}+x^{67}+x^{68}+x^{71}+x^{72}+x^{73}+x^{75}+x^{77}+x^{78}+x^{79}+x^{81}+x^{83}+x^{87}+x^{88}+x^{92}+x^{93}+x^{94}+x^{95}+x^{96}+x^{97}+x^{99}+x^{100}+x^{103}+x^{104}+x^{106}+x^{107}+x^{110}+x^{111}+x^{115}+x^{117}+x^{119}+x^{120}+x^{121}+x^{122}+x^{125}+x^{130}+x^{134}+x^{135}+x^{136}+x^{137}+x^{138}+x^{141}+x^{142}+x^{145}+x^{147}+x^{148}+x^{150}+x^{152}+x^{154}+x^{156}+x^{158}+x^{160}+x^{161}+x^{162}+x^{163}+x^{168}+x^{169}+x^{170}+x^{172}+x^{173}+x^{174}+x^{175}+x^{176}+x^{177}+x^{178}+x^{180}+x^{181}+x^{183}+x^{184}+x^{187}+x^{188}+x^{195}+x^{197}+x^{198}+x^{199}+x^{202}+x^{203}+x^{205}+x^{206}+x^{207}+x^{208}+x^{210}+x^{211}+x^{212}+x^{213}+x^{214}+x^{216}+x^{217}+x^{220}+x^{222}+x^{224}+x^{225}+x^{227}+x^{228}+x^{230}+x^{231}+x^{232}.$$

Finalmente el orden del punto P es el número primo n ($nP = \mathcal{O}$) de 70 dígitos decimales,

$n = 345087317339528189371737793113851276057094098886225212 6-328087024741343.$

Generación de claves

Ejemplo 3.6.6. Dos entidades **A** y **B** generan independiente y localmente su par de claves de la siguiente manera.

1. Supóngase que **A** elige aleatoriamente el número

   3329213996796242754435361955520098412039689976925789837360-026266230594, al que llamaremos d_A y que está en $[1, n-1]$.

2. Se calcula $Q_A = d_A P$, la abscisa de Q_A es

   $$1+x^2+x^3+x^6+x^{10}+x^{11}+x^{12}+x^{16}+x^{18}+x^{19}+x^{20}+x^{21}+x^{22}+x^{24}+x^{25}+x^{26}+x^{30}+x^{32}+x^{34}+x^{37}+x^{38}+x^{39}+x^{40}+x^{42}+x^{44}+x^{46}+x^{47}+x^{48}+x^{51}+x^{52}+x^{53}+x^{55}+x^{57}+x^{58}+x^{60}+x^{61}+x^{62}+x^{64}+x^{65}+x^{66}+x^{68}+x^{69}+x^{71}+x^{72}+x^{73}+x^{76}+x^{77}+x^{79}+x^{81}+x^{82}+x^{83}+x^{85}+x^{88}+x^{89}+x^{90}+x^{91}+x^{94}+x^{98}+x^{101}+x^{102}+x^{104}+x^{105}+x^{106}+x^{107}+x^{108}+x^{109}+x^{111}+x^{114}+x^{118}+x^{119}+x^{120}+x^{121}+x^{122}+x^{128}+x^{129}+x^{130}+x^{133}+x^{137}+x^{138}+x^{139}+x^{140}+x^{141}+x^{142}+x^{143}+x^{146}+x^{147}+x^{148}+x^{149}+x^{150}+x^{151}+x^{153}+x^{154}+x^{155}+x^{159}+x^{161}+x^{163}+x^{165}+x^{166}+x^{176}+x^{182}+x^{183}+x^{189}+x^{191}+x^{193}+x^{199}+x^{208}+x^{211}+x^{214}+x^{216}+x^{217}+x^{220}+x^{222}+x^{223}+x^{224}+x^{225}+x^{227}+x^{229},$$

   y su ordenada

   $$x^4+x^5+x^6+x^{12}+x^{14}+x^{15}+x^{19}+x^{20}+x^{23}+x^{25}+x^{26}+x^{27}+x^{28}+x^{32}+x^{34}+x^{36}+x^{37}+x^{38}+x^{39}+x^{40}+x^{42}+x^{44}+x^{45}+x^{48}+x^{49}+x^{52}+x^{53}+x^{54}+x^{55}+x^{56}+x^{57}+x^{58}+x^{59}+x^{65}+x^{68}+x^{69}+x^{72}+x^{73}+x^{76}+x^{77}+x^{78}+x^{79}+x^{80}+x^{81}+x^{82}+x^{84}+x^{85}+x^{86}+x^{89}+x^{90}+x^{93}+x^{95}+x^{97}+x^{101}+x^{105}+x^{107}+x^{108}+x^{109}+x^{110}+x^{111}+x^{112}+x^{114}+x^{116}+x^{120}+x^{121}+x^{124}+x^{126}+x^{129}+x^{132}+x^{133}+x^{134}+x^{135}+x^{141}+x^{146}+x^{147}+x^{148}+x^{150}+x^{153}+x^{154}+x^{155}+x^{156}+x^{158}+x^{160}+x^{164}+x^{166}+x^{170}+x^{172}+x^{173}+x^{174}+x^{175}+x^{179}+x^{180}+x^{182}+x^{184}+x^{186}+x^{189}+x^{197}+x^{198}+x^{200}+x^{205}+x^{208}+x^{211}+x^{212}+x^{215}+x^{217}+x^{218}+x^{219}+x^{220}+x^{222}+x^{224}+x^{225}+x^{227}.$$

3. La clave pública de **A** es Q_A.

4. La clave privada de **A** es d_A.

En el caso de **B**.

1. Supóngase que **B** elige aleatoriamente el número

   150410132305498090422574350918755078433610950381391124232-2619772964461, al que llamaremos d_B y que está en $[1, n-1]$.

2. Se calcula $Q_B = d_B\, P$, la abscisa de Q_B es

$$x+x^2+x^3+x^4+x^6+x^7+x^8+x^9+x^{11}+x^{13}+x^{15}+x^{16}+x^{17}+x^{19}+x^{20}+ x^{21}+x^{28}+x^{30}+x^{32}+x^{33}+x^{34}+x^{36}+x^{38}+x^{39}+x^{40}+x^{41}+x^{42}+x^{43}+ x^{45}+x^{48}+x^{52}+x^{53}+x^{55}+x^{56}+x^{57}+x^{62}+x^{63}+x^{64}+x^{65}+x^{68}+x^{70}+ x^{72}+x^{74}+x^{75}+x^{77}+x^{79}+x^{80}+x^{81}+x^{83}+x^{84}+x^{85}+x^{86}+x^{89}+x^{92}+ x^{96}+x^{99}+x^{102}+x^{104}+x^{105}+x^{107}+x^{108}+x^{109}+x^{110}+x^{111}+x^{113}+x^{116}+ x^{117}+x^{118}+x^{120}+x^{122}+x^{126}+x^{128}+x^{129}+x^{131}+x^{134}+x^{135}+x^{140}+x^{141}+ x^{142}+x^{144}+x^{145}+x^{147}+x^{148}+x^{149}+x^{150}+x^{155}+x^{158}+x^{161}+x^{162}+x^{166}+ x^{170}+x^{171}+x^{173}+x^{175}+x^{176}+x^{177}+x^{178}+x^{180}+x^{182}+x^{183}+x^{186}+x^{188}+ x^{189}+x^{190}+x^{191}+x^{196}+x^{197}+x^{198}+x^{199}+x^{200}+x^{203}+x^{205}+x^{206}+x^{207}+ x^{210}+x^{211}+x^{213}+x^{217}+x^{218}+x^{219}+x^{222}+x^{223}+x^{224}+x^{228}+x^{230}+x^{231},$$

y su ordenada

$$x^2+x^4+x^6+x^9+x^{14}+x^{15}+x^{16}+x^{17}+x^{18}+x^{22}+x^{24}+x^{25}+x^{27}+x^{28}+ x^{31}+x^{32}+x^{34}+x^{35}+x^{43}+x^{46}+x^{48}+x^{51}+x^{52}+x^{54}+x^{57}+x^{58}+x^{59}+ x^{62}+x^{64}+x^{68}+x^{69}+x^{70}+x^{71}+x^{77}+x^{78}+x^{81}+x^{82}+x^{88}+x^{90}+x^{97}+ x^{98}+x^{99}+x^{100}+x^{103}+x^{104}+x^{105}+x^{108}+x^{109}+x^{110}+x^{112}+x^{114}+x^{117}+ x^{118}+x^{119}+x^{121}+x^{124}+x^{126}+x^{128}+x^{130}+x^{135}+x^{136}+x^{137}+x^{138}+ x^{142}+x^{143}+x^{144}+x^{145}+x^{146}+x^{151}+x^{152}+x^{153}+x^{157}+x^{159}+x^{160}+ x^{163}+x^{165}+x^{166}+x^{167}+x^{168}+x^{170}+x^{171}+x^{172}+x^{175}+x^{178}+x^{180}+ x^{181}+x^{183}+x^{189}+x^{190}+x^{193}+x^{195}+x^{199}+x^{201}+x^{204}+x^{208}+x^{210}+x^{211}+ x^{215}+x^{216}+x^{217}+x^{220}+x^{221}+x^{222}+x^{223}+x^{224}+x^{226}+x^{228}+x^{229}+x^{231}.$$

3. La clave pública de **B** es Q_B.

4. La clave privada de **B** es d_B.

Cifrado de un mensaje

Ejemplo 3.6.7. Supongamos que la entidad **A** desea enviar a **B** un mensaje m de manera cifrada. Donde m = "Esta es una prueba de como funciona el criptosistema con el caso del cifrado.".

1. Se recupera la clave pública de **B**, es decir Q_B.

2. El mensaje tiene 77 caracteres, ahora procedemos a representarlo como parejas ordenadas de $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$. Cada entrada de la pareja posee 233

bits, si se divide 233 entre 8 (el tamaño de un caracter en el código ASCII), tenemos que caben 29 caracteres, por lo cual en la pareja se pueden tener hasta 58 caracteres. Entonces, se toman los primeros 58 caracteres de la cadena, lo cual equivale a "Esta es una prueba de como funciona el criptosistema con e", esta cadena puede verse también como una cadena binaria, si se piensa en la representación binaria de la posición de cada caracter en la tabla del código ASCII. Ahora bien, la cadena se parte por la mitad, entonces tenemos dos cadenas de 29 caracteres, cada una de ellas tiene 232 bits, sin embargo, nos interesa ver este par de cadenas como elementos en $\mathbb{F}_{2^{233}}$, por lo que este fragmento de texto de 58 caracteres sería el siguiente par ordenado

$$(1+x^2+x^6+x^8+x^9+x^{12}+x^{13}+x^{14}+x^{18}+x^{20}+x^{21}+x^{22}+x^{24}+x^{29}+x^{30}+x^{37}+x^{40}+x^{42}+x^{45}+x^{46}+x^{48}+x^{49}+x^{52}+x^{53}+x^{54}+x^{61}+x^{64}+x^{66}+x^{68}+x^{69}+x^{70}+x^{73}+x^{74}+x^{75}+x^{77}+x^{78}+x^{80}+x^{85}+x^{86}+x^{93}+x^{100}+x^{101}+x^{102}+x^{105}+x^{108}+x^{109}+x^{110}+x^{112}+x^{114}+x^{116}+x^{117}+x^{118}+x^{120}+x^{122}+x^{125}+x^{126}+x^{129}+x^{133}+x^{134}+x^{136}+x^{141}+x^{142}+x^{149}+x^{154}+x^{157}+x^{158}+x^{160}+x^{162}+x^{165}+x^{166}+x^{173}+x^{176}+x^{177}+x^{181}+x^{182}+x^{184}+x^{185}+x^{186}+x^{187}+x^{189}+x^{190}+x^{192}+x^{194}+x^{195}+x^{197}+x^{198}+x^{200}+x^{201}+x^{202}+x^{203}+x^{205}+x^{206}+x^{213}+x^{217}+x^{218}+x^{221}+x^{222}+x^{224}+x^{226}+x^{228}+x^{229}+x^{230},$$
$$x+x^2+x^3+x^5+x^6+x^8+x^9+x^{13}+x^{14}+x^{16}+x^{19}+x^{21}+x^{22}+x^{24}+x^{25}+x^{26}+x^{27}+x^{29}+x^{30}+x^{33}+x^{34}+x^{35}+x^{37}+x^{38}+x^{40}+x^{45}+x^{46}+x^{53}+x^{56}+x^{58}+x^{61}+x^{62}+x^{66}+x^{67}+x^{69}+x^{70}+x^{77}+x^{80}+x^{81}+x^{85}+x^{86}+x^{89}+x^{92}+x^{93}+x^{94}+x^{96}+x^{99}+x^{101}+x^{102}+x^{108}+x^{109}+x^{110}+x^{114}+x^{116}+x^{117}+x^{118}+x^{120}+x^{121}+x^{122}+x^{123}+x^{125}+x^{126}+x^{128}+x^{129}+x^{132}+x^{133}+x^{134}+x^{136}+x^{139}+x^{141}+x^{142}+x^{144}+x^{145}+x^{148}+x^{149}+x^{150}+x^{154}+x^{156}+x^{157}+x^{158}+x^{160}+x^{162}+x^{165}+x^{166}+x^{168}+x^{170}+x^{171}+x^{173}+x^{174}+x^{176}+x^{181}+x^{182}+x^{189}+x^{192}+x^{193}+x^{197}+x^{198}+x^{200}+x^{201}+x^{202}+x^{203}+x^{205}+x^{206}+x^{209}+x^{210}+x^{211}+x^{213}+x^{214}+x^{221}+x^{224}+x^{226}+x^{229}+x^{230}).$$

El resto de la cadena "l caso del cifrado." se representa por el par ordenado

$$(x^2+x^3+x^5+x^6+x^{13}+x^{16}+x^{17}+x^{21}+x^{22}+x^{24}+x^{29}+x^{30}+x^{32}+x^{33}+x^{36}+x^{37}+x^{38}+x^{40}+x^{41}+x^{42}+x^{43}+x^{45}+x^{46}+x^{53}+x^{58}+x^{61}+x^{62}+x^{64}+x^{66}+x^{69}+x^{70}+x^{74}+x^{75}+x^{77}+x^{78}+x^{85}+x^{88}+x^{89}+x^{93}+x^{94}+x^{96}+x^{99}+x^{101}+x^{102}+x^{105}+x^{106}+x^{109}+x^{110}+$$

$x^{113}+x^{116}+x^{117}+x^{118}+x^{120}+x^{125}+x^{126}+x^{130}+x^{133}+x^{134}+x^{136}+x^{137}+x^{138}+x^{139}+x^{141}+x^{142}+x^{145}+x^{146}+x^{147}+x^{149},0)$.

Entonces, estas parejas representan al mensaje m visto como elementos en $\mathbb{F}_{2^{233}} \times \mathbb{F}_{2^{233}}$.

3. Supóngase que se elige de manera aleatoria

   7552142948134102344815845565859858714881598134101390636463−02507579662 $\in [1, n-1]$. A este número lo llamaremos k.

4. Se calcula el punto $kP = (x_1, y_1)$, del cual se dan sus coordenadas

   $(x+x^3+x^7+x^8+x^9+x^{10}+x^{11}+x^{13}+x^{15}+x^{17}+x^{18}+x^{20}+x^{21}+x^{22}+x^{23}+x^{24}+x^{25}+x^{30}+x^{31}+x^{32}+x^{34}+x^{35}+x^{37}+x^{38}+x^{39}+x^{40}+x^{41}+x^{42}+x^{44}+x^{48}+x^{50}+x^{51}+x^{55}+x^{57}+x^{58}+x^{59}+x^{61}+x^{62}+x^{63}+x^{66}+x^{73}+x^{79}+x^{81}+x^{82}+x^{85}+x^{91}+x^{92}+x^{93}+x^{95}+x^{96}+x^{103}+x^{105}+x^{106}+x^{111}+x^{112}+x^{114}+x^{118}+x^{121}+x^{123}+x^{125}+x^{133}+x^{138}+x^{141}+x^{142}+x^{146}+x^{147}+x^{148}+x^{149}+x^{150}+x^{151}+x^{153}+x^{154}+x^{157}+x^{158}+x^{160}+x^{162}+x^{163}+x^{165}+x^{166}+x^{167}+x^{169}+x^{171}+x^{172}+x^{174}+x^{175}+x^{179}+x^{181}+x^{182}+x^{185}+x^{186}+x^{187}+x^{193}+x^{194}+x^{195}+x^{197}+x^{200}+x^{204}+x^{206}+x^{208}+x^{209}+x^{210}+x^{216}+x^{220}+x^{221}+x^{223}+x^{224}+x^{225}+x^{226}+x^{229}+x^{232}, x+x^3+x^4+x^5+x^6+x^7+x^8+x^{10}+x^{12}+x^{14}+x^{15}+x^{19}+x^{20}+x^{23}+x^{25}+x^{27}+x^{28}+x^{33}+x^{34}+x^{35}+x^{37}+x^{38}+x^{39}+x^{40}+x^{41}+x^{46}+x^{47}+x^{48}+x^{49}+x^{52}+x^{56}+x^{57}+x^{59}+x^{64}+x^{66}+x^{71}+x^{72}+x^{73}+x^{75}+x^{79}+x^{80}+x^{81}+x^{82}+x^{86}+x^{89}+x^{90}+x^{94}+x^{96}+x^{97}+x^{99}+x^{100}+x^{101}+x^{104}+x^{107}+x^{108}+x^{110}+x^{113}+x^{115}+x^{120}+x^{122}+x^{127}+x^{128}+x^{129}+x^{130}+x^{135}+x^{137}+x^{138}+x^{142}+x^{147}+x^{148}+x^{151}+x^{152}+x^{153}+x^{156}+x^{159}+x^{160}+x^{161}+x^{162}+x^{163}+x^{166}+x^{167}+x^{169}+x^{171}+x^{172}+x^{173}+x^{174}+x^{176}+x^{177}+x^{179}+x^{181}+x^{182}+x^{183}+x^{184}+x^{185}+x^{188}+x^{191}+x^{194}+x^{196}+x^{197}+x^{199}+x^{201}+x^{202}+x^{203}+x^{204}+x^{205}+x^{206}+x^{207}+x^{208}+x^{209}+x^{211}+x^{212}+x^{213}+x^{215}+x^{220}+x^{223}+x^{226}+x^{230}+x^{231})$.

5. Se calcula el punto $kQ_B = (x_2, y_2)$, se dan sus coordenadas

   $(1+x^3+x^5+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}+x^{15}+x^{18}+x^{19}+x^{20}+x^{23}+x^{26}+x^{28}+x^{30}+x^{34}+x^{35}+x^{36}+x^{37}+x^{38}+x^{39}+x^{43}+x^{44}+x^{46}+x^{47}+x^{48}+x^{50}+x^{51}+x^{56}+x^{57}+x^{58}+x^{59}+x^{60}+x^{63}+x^{64}+x^{65}+x^{66}+x^{69}+$

$$x^{71}+x^{72}+x^{73}+x^{75}+x^{77}+x^{79}+x^{83}+x^{89}+x^{93}+x^{94}+x^{98}+x^{99}+x^{102}+x^{103}+x^{104}+x^{105}+x^{107}+x^{112}+x^{114}+x^{115}+x^{120}+x^{121}+x^{122}+x^{124}+x^{126}+x^{127}+x^{130}+x^{131}+x^{138}+x^{143}+x^{145}+x^{146}+x^{147}+x^{149}+x^{157}+x^{158}+x^{159}+x^{160}+x^{161}+x^{162}+x^{165}+x^{168}+x^{169}+x^{171}+x^{172}+x^{174}+x^{176}+x^{179}+x^{180}+x^{181}+x^{187}+x^{188}+x^{192}+x^{194}+x^{195}+x^{197}+x^{199}+x^{200}+x^{201}+x^{202}+x^{203}+x^{205}+x^{206}+x^{207}+x^{208}+x^{209}+x^{211}+x^{212}+x^{213}+x^{216}+x^{217}+x^{218}+x^{219}+x^{220}+x^{223}+x^{224}+x^{225}+x^{226}+x^{227}+x^{228}+x^{229}, x^4+x^6+x^{10}+x^{19}+x^{20}+x^{21}+x^{22}+x^{25}+x^{26}+x^{27}+x^{29}+x^{30}+x^{31}+x^{32}+x^{33}+x^{36}+x^{38}+x^{40}+x^{42}+x^{44}+x^{45}+x^{47}+x^{48}+x^{50}+x^{52}+x^{59}+x^{61}+x^{65}+x^{67}+x^{68}+x^{69}+x^{70}+x^{74}+x^{78}+x^{82}+x^{83}+x^{91}+x^{97}+x^{100}+x^{104}+x^{105}+x^{106}+x^{109}+x^{110}+x^{115}+x^{116}+x^{121}+x^{122}+x^{125}+x^{128}+x^{129}+x^{130}+x^{131}+x^{133}+x^{134}+x^{136}+x^{139}+x^{140}+x^{142}+x^{143}+x^{144}+x^{149}+x^{151}+x^{152}+x^{154}+x^{155}+x^{156}+x^{157}+x^{160}+x^{161}+x^{162}+x^{163}+x^{164}+x^{165}+x^{168}+x^{169}+x^{170}+x^{171}+x^{173}+x^{174}+x^{175}+x^{177}+x^{178}+x^{179}+x^{180}+x^{181}+x^{183}+x^{185}+x^{186}+x^{189}+x^{190}+x^{191}+x^{192}+x^{194}+x^{196}+x^{197}+x^{198}+x^{199}+x^{201}+x^{202}+x^{204}+x^{205}+x^{207}+x^{208}+x^{211}+x^{212}+x^{213}+x^{216}+x^{221}+x^{224}+x^{227}+x^{228}+x^{232}).$$

6. Se calcula el x_2^3 y se forma x_4 tomando los primeros 117 bits de x_2 y los últimos 116 de x_2^3. Asimismo, y_4 se forma tomando los primeros 117 bits de x_2^3 y los últimos 116 de x_2. Resultando la pareja (x_4, y_4).

$$(1+x^3+x^5+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}+x^{15}+x^{18}+x^{19}+x^{20}+x^{23}+x^{26}+x^{28}+x^{30}+x^{34}+x^{35}+x^{36}+x^{37}+x^{38}+x^{39}+x^{43}+x^{44}+x^{46}+x^{47}+x^{48}+x^{50}+x^{51}+x^{56}+x^{57}+x^{58}+x^{59}+x^{60}+x^{63}+x^{64}+x^{65}+x^{66}+x^{69}+x^{71}+x^{72}+x^{73}+x^{75}+x^{77}+x^{79}+x^{83}+x^{89}+x^{93}+x^{94}+x^{98}+x^{99}+x^{102}+x^{103}+x^{104}+x^{105}+x^{107}+x^{112}+x^{114}+x^{115}+x^{117}+x^{119}+x^{121}+x^{123}+x^{124}+x^{125}+x^{126}+x^{129}+x^{132}+x^{133}+x^{140}+x^{141}+x^{143}+x^{146}+x^{147}+x^{148}+x^{149}+x^{150}+x^{153}+x^{156}+x^{161}+x^{163}+x^{164}+x^{166}+x^{167}+x^{168}+x^{171}+x^{172}+x^{174}+x^{177}+x^{179}+x^{181}+x^{184}+x^{188}+x^{189}+x^{190}+x^{197}+x^{198}+x^{199}+x^{200}+x^{203}+x^{206}+x^{208}+x^{212}+x^{214}+x^{216}+x^{217}+x^{220}+x^{223}+x^{224}+x^{227}+x^{229}+x^{231}+x^{232}, x+x^3+x^4+x^9+x^{10}+x^{11}+x^{15}+x^{16}+x^{17}+x^{18}+x^{19}+x^{20}+x^{21}+x^{23}+x^{25}+x^{26}+x^{27}+x^{29}+x^{31}+x^{35}+x^{36}+x^{37}+x^{40}+x^{47}+x^{49}+x^{50}+x^{51}+x^{52}+x^{54}+x^{57}+x^{58}+x^{66}+x^{67}+x^{71}+x^{73}+x^{74}+x^{75}+x^{78}+x^{79}+x^{80}+x^{84}+x^{87}+x^{89}+x^{90}+x^{91}+x^{92}+x^{94}+x^{95}+x^{96}+x^{97}+x^{98}+x^{100}+x^{101}+x^{103}+x^{104}+x^{105}+x^{106}+x^{107}+x^{108}+x^{110}+x^{111}+x^{112}+x^{113}+x^{115}+x^{120}+x^{121}+x^{122}+x^{124}+x^{126}+x^{127}+x^{130}+x^{131}+x^{138}+x^{143}+x^{145}+x^{146}+x^{147}+x^{149}+x^{157}+x^{158}+x^{159}+x^{160}+x^{161}+x^{162}+x^{165}+x^{168}+x^{169}+x^{171}+x^{172}+x^{174}+x^{176}+x^{179}+x^{180}+x^{181}+x^{187}+x^{188}+x^{192}+x^{194}+x^{195}+x^{197}+x^{199}+$$

$x^{200}+x^{201}+x^{202}+x^{203}+x^{205}+x^{206}+x^{207}+x^{208}+x^{209}+x^{211}+x^{212}+x^{213}+x^{216}+x^{217}+x^{218}+x^{219}+x^{220}+x^{223}+x^{224}+x^{225}+x^{226}+x^{227}+x^{228}+x^{229})$.

Tenemos dos parejas ordenadas donde "acomodamos" el texto, cada una de estas parejas podemos verlas como $M_i = (m_1, m_2)_i$ donde en esta caso $i = 1, 2$ ya que únicamente se tienen dos parejas. El siguiente proceso se aplicará a cada pareja. Se generan $C_i = (c_{i1}, c_{i2})$, donde $i = 1, 2$, $c_{i1} = (m_{i1} + y_2)x_4$ y $c_{i2} = (m_{i2} + x_2)y_4$.

7. **A** envía (x_1, y_1, C_1, C_2).

Descifrado de un mensaje

Ejemplo 3.6.8. Una vez que la entidad **B** recibe de **A** un mensaje cifrado (x_1, y_1, C_1, C_2), **B** procede como sigue.

1. **B** clacula $d_B(x_1, y_1) = d_B(kP) = k(d_BP) = k(Q_B) = (x_2, y_2)$ usando su clave secreta, en este caso d_B.

2. **B** calcula el par ordenado (x_4, y_4) de manera idéntica a como lo hizo **A**.

3. Finalmente **B** calcula

$$m_{i1} = c_{i1}x_4^{-1} + y_2 \text{ y } m_{i2} = c_{i2}y_4^{-1} + x_2,$$

para cada $C_i = (c_{i1}, c_{i2})$, con $i = 1, 2$, con lo que se obtienen las respectivas M_i, es decir, se recupera el mensaje.

Una firma digital

Ejemplo 3.6.9. Supóngase que la entidad **A** ahora desea firmar el mensaje m y enviárselo a **B**. El mensaje es el mismo que se empleó para el caso del cifrado-descifrado.

1. Supóngase que **A** escoge de manera aleatoria el número 2039781769362−461478363077735430872913579407259824170175001122105754300, el cual está en $[1, n-1]$. A este número le llamaremos k.

2. **A** calcula $k\,P = (x_1, y_1)$, del cual se presentan sus coordenadas

$$(1+x^4+x^6+x^7+x^8+x^{10}+x^{12}+x^{13}+x^{14}+x^{15}+x^{16}+x^{20}+x^{22}+x^{23}+x^{24}+x^{25}+x^{29}+x^{30}+x^{31}+x^{33}+x^{34}+x^{38}+x^{42}+x^{44}+x^{49}+x^{50}+x^{53}+x^{55}+x^{57}+x^{58}+x^{61}+x^{63}+x^{64}+x^{69}+x^{70}+x^{72}+x^{77}+x^{78}+x^{81}+x^{82}+x^{85}+x^{88}+x^{89}+x^{92}+x^{94}+x^{97}+x^{101}+x^{102}+x^{103}+x^{104}+x^{106}+x^{108}+x^{113}+x^{114}+x^{115}+x^{119}+x^{120}+x^{123}+x^{124}+x^{127}+x^{130}+x^{133}+x^{134}+x^{136}+x^{138}+x^{139}+x^{141}+x^{142}+x^{147}+x^{151}+x^{153}+x^{154}+x^{157}+x^{158}+x^{159}+x^{161}+x^{162}+x^{163}+x^{165}+x^{166}+x^{168}+x^{169}+x^{171}+x^{172}+x^{175}+x^{176}+x^{182}+x^{183}+x^{184}+x^{185}+x^{186}+x^{193}+x^{194}+x^{195}+x^{196}+x^{201}+x^{203}+x^{205}+x^{207}+x^{209}+x^{213}+x^{221}+x^{222}+x^{223}+x^{225}+x^{230}, 1+x+x^4+x^5+x^6+x^7+x^8+x^{10}+x^{12}+x^{15}+x^{16}+x^{20}+x^{21}+x^{23}+x^{26}+x^{27}+x^{30}+x^{31}+x^{32}+x^{36}+x^{37}+x^{38}+x^{40}+x^{43}+x^{44}+x^{46}+x^{50}+x^{52}+x^{53}+x^{55}+x^{57}+x^{58}+x^{60}+x^{61}+x^{65}+x^{68}+x^{69}+x^{71}+x^{72}+x^{73}+x^{74}+x^{75}+x^{76}+x^{77}+x^{78}+x^{80}+x^{82}+x^{85}+x^{90}+x^{91}+x^{92}+x^{93}+x^{95}+x^{97}+x^{98}+x^{99}+x^{100}+x^{101}+x^{102}+x^{103}+x^{106}+x^{107}+x^{111}+x^{112}+x^{114}+x^{115}+x^{116}+x^{117}+x^{118}+x^{119}+x^{122}+x^{124}+x^{125}+x^{126}+x^{127}+x^{129}+x^{130}+x^{131}+x^{132}+x^{133}+x^{139}+x^{140}+x^{141}+x^{146}+x^{148}+x^{152}+x^{153}+x^{154}+x^{159}+x^{160}+x^{161}+x^{162}+x^{163}+x^{165}+x^{168}+x^{169}+x^{175}+x^{177}+x^{178}+x^{180}+x^{181}+x^{183}+x^{184}+x^{185}+x^{187}+x^{188}+x^{189}+x^{190}+x^{197}+x^{199}+x^{201}+x^{202}+x^{203}+x^{204}+x^{206}+x^{208}+x^{209}+x^{210}+x^{213}+x^{215}+x^{217}+x^{219}+x^{221}+x^{223}+x^{224}+x^{225}+x^{226}+x^{227}+x^{232})$$

Nótese que la pareja ordenada también la podemos ver como una pareja de números enteros, donde a cada x^i le corresponde 2^i. Entonces se calcula $r = x_1 \bmod n$, y como resultado se tiene $r = 180296069352240702599216715027234411336797664369560701199272-5349332433$

3. **A** calcula $k^{-1} \bmod n$, arrojando el número $k^{-1} = 238333946788420247-505125536494846062632732051619495874917284895747599$.

4. El valor que arroja la función hash (MD5) respecto a m visto como número entero es $h(m) = 26881275299219864519723640679990991075$, si se reduce módulo n este valor permanece inalterado, posteriormente se calcula $s = k^{-1}(h(m) + d_A r) \bmod n$, siendo el resultado $s = 3104293681825607129703176128036375412983281147342321190519106-3270778$.

5. La firma digital que **A** produce para el mensaje m es (r, s).

Verificación de una firma digital

Ejemplo 3.6.10. La entidad **B** recibe por parte de **A** la firma (r, s) junto con el correspondiente mensaje m, para su verificación.

1. La entidad **B** consigue una copia de la clave pública de **A**, Q_A. Nuevamente imagínese que **A** la proporcionó directamente.

2. Los números r y s están en $[1, n-1]$.

3. B calcula $h(m)$ y $w = s^{-1} \bmod n$, siendo $w = 2677970453423971532835-6812771596821591055432721765330152435484829968 04$.

4. B calcula $u_1 = h(m)w \bmod n = 4190474365430036463912477236954 7-0155947181508332054917859996279202560$ y $u_2 = rw \bmod n = 13865-4760269484883987369217812129408612142766520330485411127915 45-87123$. Hecho lo anterior se calcula el punto $u_1 P + u_2 Q_A = (x_2, y_2)$, y $v = x_2 \bmod n = 1802960693522407025992167150272344113367976643-6956070119927253493324 33$.

5. Se acepta la firma, ya que $v = r$.

Capítulo 4

Conclusiones y futuras líneas de trabajo

En este último capítulo se tratan de resumir los resultados más importantes de esta tesis, asimismo, se mencionan algunas posibles líneas de trabajo que pueden tomarse para los diferentes temas relacionados.

En este momento el PLDCE parece ser más difícil que los problemas de factorización entera y el PLD, dado que para el caso general del primero, no se ha encontrado un algoritmo de tiempo subexponencial que lo resuelva. Por lo anterior, es posible utilizar una curva cuyo grupo de puntos racionales sea significativamente más pequeño que el utilizado en $\mathbb{F}_p^*$ para el PLD (o el problema de factorizar un número entero compuesto n), Koblitz *et al* [22]. Por ejemplo, una curva elíptica $E(\mathbb{F}_{p^n})$ con un punto generador P cuyo orden es aproximadamente un número primo de 160 bits ofrece aproximadamente el mismo nivel de seguridad que DSA con un módulo p de 1024 bits y RSA con un módulo n de 1024 bits, Koblitz *et al* [22].

Una implicación importante de lo anterior es que los criptosistemas basados en el PLDCE emplean claves significativamente más pequeñas que las usadas por tecnologías competidoras como DSA y RSA (160 bits vs 1024 bits). Algunas de las ventajas que se logran al trabajar con claves más pequeñas son: mayor velocidad y ahorros en consumo de memoria y almacenamiento. Estas ventajas son especialmente importantes en situaciones donde se tienen limitaciones de capacidad computacional, ancho de banda o espacio de almacenamiento. De aquí que la criptografía

de curvas elípticas resulte adecuada para entornos restringidos como tarjetas inteligentes (telefónicas), teléfonos celulares, agendas de bolsillo, *Palm Tops*. Otro aspecto que resulta de la carencia de un algorimo de tiempo subexponencial para el PLDCE es que se puede escoger el campo $\mathbb{F}_{2^m}$ sobre el que se define la curva y la representación de sus elementos (bases estandar o normal), de tal manera que la aritmética sobre el campo (suma, multiplicación, división e inversión) pueda ser optimizada. Esto último no es el caso para los criptosistemas basados en los problemas del logaritmo discreto y de factorización, donde el módulo primo p (así como el módulo compuesto n) no deben escogerse de una forma particular, pues esto haría la tarea del criptoanalista más fácil, Koblitz *et al* [22].

La criptografía de curvas elípticas actualmente ha sido aceptada y está siendo ofrecida a través de productos comerciales por una gran cantidad de compañías importantes como Motorola, Sony Europe, Palm, Lucent Technologies y Certicom, por mencionar algunas. Por otra parte, la criptografía de curvas elípticas también cuenta con una amplia estandarización, acreditada por organizaciones reconocidas internacionalmente. Los estándares para la implementación del algoritmo de firma digital basada en curvas elípticas (ECDSA) con mayor reconocimiento son ISO 14888-3, ISO/IEC 9796-4, ISO/IEC 14946, IEEE 1363-2000, ANSI X9.62, ANSI X9.63, SECG y el FIPS 186-2. Este último, es el estándar que actualmente utilizan todas las entidades de gobierno de los E.U. que demanden el uso de firmas digitales, las cuales pueden ser generadas mediante DSA, el algoritmo de firma digital de RSA o ECDSA.

Dentro de las tecnologías estándar a nivel comercial que han adoptado a la criptografía de curvas elípticas están ATM (*Asynchronous Transport Mode*), WAP (*Wireless Application Protocol*), ReFLEX (estándar de paginación en dos sentidos de Motorola), la organización IETF (*The Internet Engineering Task Force*) la cual enmarca a los protocolos SSL/TLS, IPSEC, PKIX, S/MIME, por mencionar algunas.

En la actualidad, la oferta más popular de criptografía de clave pública sigue siendo RSA, entonces, resulta natural preguntarse ¿qué criptosistema es mejor, RSA o ECC (*Elliptic Curve Cryptosystem*)?

Lo que puede asegurarse en este momento respecto a qué criptosistema

es mejor, DSA, RSA o ECC, es que los criptosistemas de curvas elípticas resultan más apropiados para aplicaciones cuyos recursos de memoria, espacio o transmisión se ven restringidos.

Con lo anterior lo que puede concluirse es que existe una base sólida que respalda a la criptografía de curvas elípticas, tanto teórica como práctica, y por tanto puede considerarse una opción seria para su implementación y uso.

En la construcción de un criptosistema que basa su seguridad en la dificultad de resolver el PLDCE, es de vital importancia la elección de sus elementos; en particular el campo y la curva. Para la elección del campo se tienen básicamente tres opciones: (1) campos $\mathbb{Z}_p$; (2) campos del tipo $\mathbb{F}_{2^{mn}}$, también llamados "compuestos" y (3) campos del tipo $\mathbb{F}_{2^p}$, siendo p un número primo. Para la primera opción deberá tomarse en cuenta la eficiencia de la aritmética sobre $\mathbb{Z}_p$, Koc [23]. Para el segundo y tercer caso debe considerarse si se cuenta o no con una base normal óptima (BNO) o una base estándar (polinomial) para el campo en cuestión. En términos generales, las mejores opciones son los campos $\mathbb{Z}_p$ y $\mathbb{F}_{2^p}$, ya que en el trabajo de Gaudry P. y Hess F. [9], se reporta que los campos del tipo $\mathbb{F}_{2^{mn}}$ no resultan ser adecuados desde el punto de vista de la seguridad, ya que se ha encontrado un algoritmo que resuelve el PLDCE eficientemente para dichos campos, ver también Woodbury A. D. *et al* [39]. En particular, uno de estos campos es $\mathbb{F}_{2^{209}} = \mathbb{F}_{2^{11 * 19}}$.

Para cuestiones de aplicación siempre se usan campos de característica 2, uno de los tipos de curvas que resultan más adecuado son las llamadas curvas anómalas o de Koblitz, ya que dichas curvas poseen buenas propiedades criptográficas: (1) estas curvas son no-supersingulares (se evita el ataque MOV); (2) el orden del grupo de puntos posee un factor primo grande (se evitan los ataques que se basan en la factorización del número de puntos); (3) el sumar un punto consigo mismo puede ejecutarse eficientemente; (4) las curvas son fáciles de encontrar.

El resultado final de este trabajo es una biblioteca de rutinas criptográficas. La biblioteca cuenta con rutinas de cifrado/descifrado, firma digital/verificación de firma digital y ensobretado/apertura de sobre. La construcción de la biblioteca utiliza elementos recomendados por los estándares internacionales, por lo que se garantiza la seguridad en el uso de

las rutinas. Con respecto a la eficiencia, se puede señalar que de acuerdo a los tiempos registrados, la biblioteca resulta competitiva comparada con otras implementaciones, por ejemplo las rutinas en De Win *et al* [3]. Por lo anterior, se tienen elementos para concluir que la implementación de la biblioteca es segura en términos prácticos, por lo que puede usarse en aplicaciones reales.

La implementación aquí lograda lleva de manera natural a pensar en futuras versiones de la misma. Por ejemplo, con el propósito de continuar trabajando de acuerdo a los estándares, es conveniente usar el algoritmo de digestión SHA-1 (*Secure Hash Algorithm*), del cual se tiene una especificación muy precisa en el FIPS 186-2 [4]. Asimismo, deberá usarse un algoritmo de cifrado simétrico actual, dentro de las posibles opciones destaca *Rijndael* [2], el cual es un algoritmo de dominio público y constituye el estándar utilizado por el gobierno de E.U.

Otra versión podría consistir en generar la ecuación de la curva de manera aleatoria, utilizando los resultados con los que ya se cuenta para identificar curvas con buenas propiedades criptográficas. Lo anterior con el fin de que un criptoanalista no se ocupe únicamente de un tipo de curva.

Finalmente, al momento de estar escribiendo estas líneas se tiene un artículo (Hasan [12]) donde se exponen mejoras específicas para las rutinas en $\mathbb{F}_{2^{233}}$, así que los tiempos que se reportan en este trabajo son ciertamente factibles de mejorar.

Anexo

A continuación se expone la sintaxis para invocar a cada una de las rutinas que conforman la biblioteca producto de este trabajo. El formato para presentar cada rutina es el siguiente.

1. *Nombre de la rutina*
2. *Sintaxis de la rutina*
3. *Descripción de la rutina*
4. *Descripción de los parámetros*
5. *Valores de retorno de la rutina*

generarllaves

Sintaxis:

```
char * generarllaves(char *allavepri, char *allavepub,
                     char *palSecreta);
```

Descripción:

Genera un par de claves.

Tipo	Parámetro	Descripción
char *	allavepri	Nombre del archivo que contendrá la clave privada.
char *	allavepub	Nombre del archivo que contendrá la clave pública.
char *	palSecreta	Contraseña que resguardará a la clave privada. Importante: su longitud deberá ser de 7 caracteres.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

Cifra

Sintaxis:

```
char * cifra(char *pub, char *pla, char *cif,
        unsigned long int  *arreglo, int modo);
```

Descripción:

Realiza el cifrado de un archivo de texto plano (pla), generando un archivo cifrado (cif).

Tipo	Parámetro	Descripción
char *	pub	Nombre del archivo que contiene a la clave pública.
char *	pla	Nombre del archivo de texto plano.
char *	cif	Nombre del archivo que contendrá el texto cifrado.
unsigned long int *	arreglo	Se usa este arreglo para recuperar la clave privada. Sin que se escriba en disco
char *	modo	Es 1 si el modo escribir es en disco y 0 si es en memoria.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

descifra

Sintaxis:

```
char * descifra(char *pri, char *palSec, char *cif,
        char *pla, unsigned long int  *arreglo, int modo);
```

Descripción:

Dado un archivo cifrado, si este se cifró con la clave adecuada, se producirá el archivo de texto plano correspondiente.

Tipo	Parámetro	Descripción
char *	pri	Nombre del archivo que contiene a la clave privada.
char *	palSec	Contraseña que resguarda la clave privada.
char *	cif	Nombre del archivo que contiene el texto cifrado.
char *	pla	Nombre del archivo que contendrá el texto plano.
unsigned long int *	arreglo	Se usa este arreglo para recuperar la clave privada. Sin que se escriba en disco
char *	modo	Es 1 si el modo escribir es en disco y 0 si es en memoria.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

firma

Sintaxis:

```
char * firma(char *pri, char *palSec, char *pla, char *fir);
```

Descripción:

Produce un archivo que contendrá la firma digital de un archivo dado.

Tipo	Parámetro	Descripción
char*	pri	Nombre del archivo que contiene a la clave privada.
char *	palSec	Contraseña que resguarda la clave privada.
char*	pla	Nombre del archivo que será firmado.
char*	fir	Nombre del archivo que contendrá la firma.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

verifica

Sintaxis:

```
char * verifica(char *pub, char *fir, char *pla);
```

Descripción:

Dado un archivo de firma digital, verifica su autenticidad.

Tipo	Parámetro	Descripción
char*	pub	Nombre del archivo que contiene a la clave pública.
char*	fir	Nombre del archivo que contiene la firma.
char*	pla	Nombre del archivo plano que produjo la firma.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

ensobreta

Sintaxis:

```
char * ensobreta(char *pri, char *pub, char *palSec,
                 char *pla, char *sob);
```

Descripción:

Dado un archivo de texto plano, se cifra con DES y posteriormente se firma, la clave que se creó para el cifrado, el texto cifrado y la firma conforman el sobre que produce esta rutina.

Tipo	Parámetro	Descripción
char *	pri	Nombre del archivo de clave privada.
char *	pub	Nombre del archivo de clave pública.
char *	palSec	Contraseña que resguarda la clave privada.
char *	pla	Nombre del archivo que contiene el texto plano.
char *	sob	Nombre del archivo que contendrá el sobre.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

verificasobre

Sintaxis:

```
char * verificasobre(char *pri, char *pub, char *palSec,
                     char *sobre);
```

Descripción:

Se encarga de abrir el sobre y verificar la firma que contiene.

Tipo	Parámetro	Descripción
char *	pri	Nombre del archivo de clave privada.
char *	pub	Nombre del archivo de clave pública.
char *	palSec	Contraseña que resguarda la clave privada.
char *	sob	Nombre del archivo que contiene el sobre.

Valores de Retorno:

Regresa una cadena que notifica si tuvo o no una terminación normal la invocación a esta rutina.

Bibliografía

[1] Angel J. J., *Criptografía y Curvas Elípticas*, Tesis de Maestría, UAM Iztapalapa, 1998.

[2] AES, Advanced Encryption Standard Development Effort, http://www.nist.gov/aes

[3] De Win E., Bosselaers A., Vandenberghe S., De Gersem P. and Vandewalle J., "A fast software Implementation for Arithmetic Operations in $GF(2^n)$", *Advances in Cryptology, Proceedings Asiacrypt'96*, LNCS 1163, Springer-Verlag, 1996, 65-76.

[4] U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Federal Information Processing Standards Publication 186-2,*FIPS PUB 186-2, Data Signature Standard*, 2000 January 27.

[5] Dieudonné J., *History of algebraic geometry*, Wadsworth Inc., 1985.

[6] Diffie W., Hellman M. E. (1976), "New directions in cryptography", *IEEE Trans. Information Theory* **22**, 644-654.

[7] Fraleigh J. B., *A first Course in Abstract Algebra*, 3rd edition, Addison-Wesley Publishing Co.,1982.

[8] Fulton W., *Algebraic curves*, New York-Amsterdam (W. A. Benjamin), 1969.

[9] Gaudry P., Hess F., "Smart N. P. Constructive and Destructive Facets of Weil Descent on Elliptic Curves".

Technical report HPL 2000-10. Visitar la dirección electrónica http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html, 2000.

[10] Griffiths P. and Harris J., *Principles of algebraic geometry*, Wiley-Interscience, New York, 1978.

[11] Hartshorne R., *Algebraic geometry*, Berlin-Heidelberg-New York (Springer), 1977.

[12] Hasan M. A., "Efficient Computation of Multiplicative Inverses for Cryptographic Applications", *15th IEEE Symposium on Computer Arithmetic*, June 11-13, 2001. Vail, Colorado.

[13] Herstein I. N., *Topics in Algebra*, Blaisdell Publishing Co., 1969.

[14] Housemöller D., *Elliptic Curves*, Springer Verlag GTM 111, 1987.

[15] Johnson G., *Computers and Intractability, a guide to the theory of NP-Completeness*, Freeman & Co., 1979.

[16] Kahn D., *The Codebreakers*, Macmillan Publishing Company, New York, 1967.

[17] Koblitz N., "Elliptic curve cryptosystems", *Math. Comp. 48*, 1987, 203-209.

[18] Koblitz N., "CM-Curves with Good Cryptographic Properties", *Advances in Cryptology Crypto '91*, LNCS 576, 1992, 279-287.

[19] Koblitz N., *Algebraic Aspects of Cryptography*, Springer-Verlag Vol 3., 1998.

[20] Koblitz N., "Constructing Elliptic Curve Cryptosystems in Characteristic 2", *Advances in Cryptology Crypto '90*, LNCS 537, 1991, 156-167.

[21] Koblitz N., *Introduction to Elliptic Curves and Modular Forms*, Springer Verlag GTM 97, 1984.

[22] Koblitz N., Menezes A., and Vanstone S., "The State of Elliptic Curve Cryptography", *Designs, Codes and Cryptography, 19*, 2000, 173-193.

[23] Koc C. K., "High-Speed RSA Implementation", *RSA Laboratories, RSA Data Security, Inc.*, 100 Marine Parkway City, CA 94065, 1995.

[24] Lidl R., Niederreiter H., *Finite Fields*, Encyclopedia of Mathematics and its Applications Vol. 20, Addison Wesley Publishing Company, 1983.

[25] López J., Dahab R.,"Performance of elliptic curve cryptosystems", *Technical report, IC-00-08*, May 2000.

[26] MacWilliams F. J. And Sloane N. J. A., "The Theory of Error-Correcting Codes", North-Holland, Amsterdan, 1977 (fifth printing:1986).

[27] Miller V. "Uses of elliptic curves in cryptography", *Advances in Cryptology - Crypto '85*, Springer-Verlag, 1986, 417-426.

[28] Menezes A., Okamoto T. and Vanstone S., "Reducing elliptic curve logarithms in a finite field", *IEEE Transactions on Information Theory*, 39 (1993), 1639-1646.

[29] Menezes A., Van Oorschot P., Vanstone S. A., *Handbook of Applied Cryptography*, CRC press, 1996.

[30] Pollard J. M., "Theorems on factorization and primality testing", *Proceedings of the Cambridge Philosophycal Society*, 76 (1974), 521-528.

[31] Schoof R. J., "Elliptic curves over finite fields and the computation of square roots mod p", *Math. Comp.*, 44 (1985). 483-494.

[32] Schoof R. J., "Nonsingular plane cubic curves over finite fields", *J. Combinatorial Theory* 46 (1987), 183-211.

[33] Schroeppel R., Orman H., O'Malley S. and Spatscheck O., "Fast key exchange with elliptic curve systems", *Advances in Cryptology, Proc. Crypto '95*, LNCS 963, D. Coppersmith, Ed., Springer-Verlag, 1995, 43-56.

[34] Semple J. G. and Roth L., *Introduction to algebraic geometry*, Oxford (Clarendon Press), 1949.

[35] Silverman J. H., *The Arithmetic of Elliptic Curves*, Springer Verlag GTM 106, 1986.

[36] Silverman J. H., *Rational Points on Elliptic Curves*, Springer Verlag UTM, 1992.

[37] Vanstone S., "The future of Elliptic Curve Cryptography", *Sixth International Conference on Finite Fields and Applications*, May 21-25, 2001, Oaxaca, México.

[38] Walker R. J., *Algebraic curves*, New York, Dover Publications Inc., 1949.

[39] Woodbury A. D., Bailey D. V., Paar Christof, "Elliptic Curve Cryptography on Smart Cards Without Coprocessors", *The 4th Smart Card Research and Advanced Applications (CARDIS 2000) Conference*, September 20-22, 2000, Bristol, UK.

[40] Zierler N., "Primitive Trinomial Whose Degree is a Mersenne Exponent", *Information and Control Vol. 15*, 1969, 67-69.

[41] Zierler N., "On x^n+x+1 over $GF(2)$", *Information and Control Vol. 16*, 1970, 502-505.

[42] Zivkovic M., "Table of Primitive Binary Polinomials II", *Mathematics of Computation Vol. 63 No. 207*, 1994, 301-306.