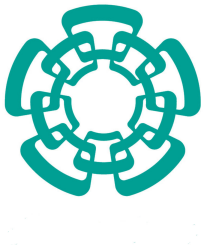

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Paralelización de la multiplicación escalar en curvas elípticas en una arquitectura multinúcleo de Intel

Tesis que presenta:

Gabriel Labrada Hernández

Para obtener el grado de:

**Maestro en Ciencias
en Computación**

Director de la Tesis:
Dr. Francisco José Rambo Rodríguez Henríquez

Ciudad de México, México. Febrero, 2010

Esta investigación fue parcialmente financiada mediante los proyectos: SEP-CONACYT 60240, SEP-CONACYT 90543

This research was partially funded by projects number: SEP-CONACYT 60240, SEP-CONACYT 90543

La tesis presentada por Gabriel Labrada Hernández fue aprobada por:

Dra. Nareli Cruz Cortés

Dr. Amilcar Meneses Viveros

Dr. Francisco José Rambo Rodríguez Henríquez , Director

Ciudad de México, México., 19 de Febrero de 2010

To my beloved family

Agradecimientos

- A mi madre, mis hermanos, gracias por su apoyo incondicional.
- Agradezco infinitamente al Dr. Francisco Rodríguez por todo su apoyo, paciencia y dedicación para orientarme en el trabajo de tesis.
- A mis lectores, Dra. Nareli Cruz Cortés , Dr. Guillermo Morales Luna, Dr. Amilcar Meneses Viveros, por sus valiosas aportaciones a mi trabajo.
- Agradezco mis amigos del CINVESTAV por brindarme su amistad y por estar siempre conmigo en las buenas y en las malas.
- Agradezco especialmente a Bris, Lulu, Lil y Paco, quienes siempre me apoyaron y me alentaron a seguir adelante y fueron como una luz en el obscuro camino.
- Gracias Sofy por alentarnos, estar al pendiente y apoyarnos siempre.
- Al CONACYT por el apoyo económico para la realización mis estudios de maestría.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Paralelización de la multiplicación escalar en curvas elípticas en una arquitectura multinúcleo de Intel

por

Gabriel Labrada Hernández
Maestro en Ciencias del Departamento de Computación
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2010
Dr. Francisco José Rambo Rodríguez Henríquez , Director

La innovación tecnológica en las computadoras ha permitido que la Ley de Moore mantenga su vigencia hasta ahora, a más de 43 años que fuera propuesta por el co-fundador de Intel: Gordon E. Moore. Esta ley expresa que "aproximadamente cada 24 meses se duplica la capacidad de los procesadores digitales" [14]. Este hecho probablemente se ha visto influenciado debido a que desde la creación de la primera computadora, ha habido una demanda general por mayor capacidad de procesamiento.

A medida que aumenta la demanda de velocidad y rendimiento, los diseñadores de procesadores enfrentan el desafío de requerir más energía para aumentar la capacidad de procesamiento. Sin embargo, el incremento en el consumo de energía tiene como consecuencia tener que administrar mayores niveles de disipación de calor y de potencia.

La tecnología multi-núcleos surge como una alternativa para el diseño de procesadores de mayor rendimiento, ya que permite realizar múltiples operaciones en un mismo ciclo de reloj y permite incrementar la capacidad de cómputo sin tener que seguir creciendo en velocidad para lograrlo. Los procesadores "Dual-core" lanzados al mercado en 2005 y los "Quad-core" en 2007, son apenas el comienzo de la estrategia de este tipo de arquitecturas. Actualmente se continúan desarrollando nuevas tecnologías que multiplican la cantidad de núcleos contenidos en un procesador. Debido a esta tendencia tecnológica, resulta conveniente para las aplicaciones y algoritmos en diferentes áreas, tales como la criptografía, utilizar las características que ofrecen; ya que se busca que los algoritmos

se ejecuten de manera rápida y eficiente.

Esta tesis presenta la paralelización de un algoritmo criptográfico que permite utilizar los recursos y las ventajas que ofrecen las arquitecturas multinúcleo. El problema para poder paralelizar la multiplicación escalar en curvas binarias para punto P desconocido, es el hecho que se requiere de una recodificación del escalar k para poder hacer uso simultáneo del doblado y de la bisección de punto. Se realizó la implementación paralela de la multiplicación escalar en curvas elípticas binarias y se obtuvo una versión paralela que permite mejorar los tiempos de ejecución respecto a las versiones secuenciales.

La importancia de los resultados obtenidos es que la multiplicación escalar es la operación básica en curvas elípticas y resulta ser la operación más costosa. Por tal motivo, al reducir los tiempos de ejecución de esta operación se reducen implícitamente los tiempos de ejecución de las aplicaciones que se basan en criptosistemas de curvas elípticas.

Abstract

Paralelization of the scalar multiplication over elliptic curves in an Intel multicore arquitecture

by

Gabriel Labrada Hernández

Master of Science from the Departamento de Computación
Research Center for Advanced Study from the National Polytechnic Institute, 2010
Dr. Francisco José Rambo Rodríguez Henríquez , Advisor

Technological innovation in computers has enabled Moore's Law remain in force until now, more than 43 years it was proposed by the co-founder of Intel: Gordon E. Moore. This law states that "about every 24 months, the capacity of digital processors is doubled." This fact has probably been influenced because since the creation of the first computer, there has been a huge demand for greater processing capacity.

With the increasing need for speed and performance, processor designers are challenged to require more energy to increase processing capacity. However, more energy consumption results in having to manage higher levels of heat dissipation and power.

Multi-core technology is emerging as an alternative for designing higher performance processors, because it allows multiple operations to be executed in one clock cycle and it also allows to increase computing capacity without having to continue increasing clock speed. "Dual-core processors released to the market" in 2005 and "Quad-cores" released in 2007, are just the beginning of a strategy that takes advantage of this kind of architectures. Nowadays processor vendors continue developing new technologies that multiply the number of cores within a processor. Due to this technological trend, it is convenient for applications and algorithms in different areas such as cryptography, to use the new features that they offer, as our goal is that algorithms run quickly and efficiently.

This thesis presents the parallelization of a cryptographic algorithm that let us take advantage of multi-core architecture features. The problem to parallelize the scalar multiplication on binary curves

for unknown point P is the fact that it requires a recoding of the scalar k in order to make it possible to use simultaneously the point doubling and point halving. We implemented a parallel version of the elliptic curve scalar multiplication over binary fields and we improved execution time compared with sequential versions.

The importance of the results is that scalar multiplication is the basic operation in elliptic curves and it is also the most expensive operation. Therefore, reducing execution time of this operation also reduces execution time of applications based on elliptic curve cryptosystems.

1

Introducción

La información se ha convertido en un recurso estratégico en las empresas y actualmente casi para cualquier persona. Dada la creciente tendencia hacia el empleo de la información en formato digital, se requiere de técnicas criptográficas no sólo para poder enviar y recibir información de manera segura, si no que en el intercambio de información entre dos o más entidades se busca ofrecer los servicios de confidencialidad, autenticación, integridad de datos, control de acceso, no repudio, disponibilidad, entre otros.

1.1 Introducción

Los sistemas criptográficos se pueden clasificar en tres tipos: esquemas criptográficos de llave secreta o cifradores simétricos, esquemas de llave pública o cifradores asimétricos y los esquemas híbridos que aprovechan las ventajas de ambos. La criptografía de llave secreta se utiliza para cifrar grandes cantidades de datos debido a su eficiencia, pero presenta la dificultad de la distribución de llaves de manera segura. Además, generalmente la criptografía de llave pública se emplea para acordar una llave de sesión que será utilizada con el método de cifrado simétrico, para realizar firmas electrónicas

y verificaciones de firmas.

Existe una tendencia por la investigación de nuevas técnicas que sean más eficientes para la implementación de criptografía de llave pública. Dos de los esquemas criptográficos más populares de llave pública son el RSA (Rivest-Shamir-Adleman) y la Criptografía de Curvas Elípticas (CCE). La seguridad de RSA se basa en el problema de factorización de números grandes, mientras que CCE garantiza su seguridad en la dificultad de resolver el Problema del Logaritmo Discreto (PLD) para curvas elípticas.

La criptografía de curvas elípticas, desde que fuera propuesta en 1985 por Neal Koblitz [9] y Victor Miller [13], ha demostrado ser capaz de proveer la misma funcionalidad que esquemas tradicionales y ampliamente usados en aplicaciones comerciales como RSA. CCE es capaz de proveer los mismos niveles de seguridad que RSA pero con llaves significativamente más pequeñas. Por ejemplo, se acepta por lo general que una llave de 160 bits en curvas elípticas provee el mismo nivel de seguridad que una llave RSA de 1024 bits.

Las aplicaciones y algoritmos que se encuentran actualmente desarrolladas de manera secuencial, no aprovechan las ventajas que ofrece la tecnología multinúcleo, ya que son ejecutadas por un solo núcleo. Las velocidades de reloj de los nuevos procesadores multinúcleo son incluso menores que la de los procesadores de un núcleo que se encontraban en el mercado. Debido a ello resulta conveniente poder contar con algoritmos y aplicaciones que permitan aprovechar las capacidades de multiprocesamiento de los nuevos procesadores.

La multiplicación escalar, mediante el empleo de algunas técnicas, resulta ser un algorítmo criptográfico paralelizable. Esta operación es fundamental en curvas elípticas, ya que es la operación básica para los criptosistemas basados en curvas elípticas y resulta ser la operación más costosa. La multiplicación escalar es utilizada en las diferentes operaciones como la generación de llaves, cifrado y decifrado, y la firma y verificación de firma. Es por ello que al lograr mejorar los tiempos de ejecución de esta operación se mejora también el desempeño de las aplicaciones cuya seguridad está basada en operaciones de curvas elípticas.

1.2 Antecedentes y motivación

En 1991 N. Koblitz [10] introdujo una familia de curvas que admiten multiplicaciones escalares especialmente rápidas. En 1998 Jerome A. Solinas [17] presenta versiones mejoradas de los algoritmos existentes para multiplicación escalar sobre curvas de Koblitz donde es posible generar una expansión del factor escalar conocida como τNAF la cual reemplaza las operaciones de doblado de puntos por exponenciaciones cuadráticas. En curvas elípticas binarias elevar al cuadrado es una operación muy rápida por lo que trabajar con el operador τ resulta conveniente para mejorar el desempeño de la multiplicación escalar.

El presente trabajo se basa en la propuesta realizada por Omran Ahmadi, Darrel Hankerson y Francisco Rodríguez en 2008 [1], donde se sugiere el uso concurrente del operador de Frobenius τ y del operador τ^{-1} para realizar la multiplicación escalar paralela para curvas de Koblitz basándose en el método clásico $\tau - and - add$. Con el método propuesto por estos tres autores, en la versión realizada en software para dos núcleos, se obtuvo una aceleración por un factor de dos con respecto a la versión secuencial. Este trabajo de tesis se enfoca en la propuesta de trabajo futuro de [1] donde se plantea utilizar la bisección de punto y el doblado de punto de manera análoga al empleo de los operadores τ y τ^{-1}, ya que dicho método aplica para curvas ordinarias.

1.3 Objetivos

El objetivo principal de este trabajo de tesis es diseñar e implementar un algoritmo paralelo de la multiplicación escalar en curvas elípticas binarias que utilice simultáneamente la bisección de punto y el doblado de punto. Para alcanzar el objetivo principal, fueron definidos algunos objetivos particulares.

1. Implementar la multiplicación escalar en curvas elípticas binarias utilizando doblado de punto.

2. Implementar la multiplicación escalar en curvas elípticas binarias empleando bisección de punto.

3. Implementar la multiplicación escalar en curvas elípticas binarias empleando simultáneamente la bisección de punto y el doblado de punto.

1.4 Metodología

En la figura 1.1 se muestra un modelo en capas de la arquitectura sobre la que se basan las implementaciones realizadas para alcanzar los objetivos de este trabajo. Como puede verse en la figura 1.1, se debe construir un modelo basado en capas, donde la capa inferior está dedicada a desarrollar la aritmética de campos finitos y es donde se definen las operaciones básicas del campo. La segunda capa que se implementa es la correspondiente a la aritmética de curvas elípticas, en esta capa se determinan las operaciones que se podrán realizar con los puntos de la curva elíptica. En la tercera capa es donde se tienen las diferentes versiones de la multiplicación escalar para curvas elípticas, esta capa depende de las dos anteriores por lo que para poder obtener una versión eficiente de la multiplicación escalar se debe contar con un buen desempeño de la aritmética de curvas elípticas y la de campos finitos. Finalmente en las capas cuatro y superiores, se encuntran los diferentes protocolos y aplicaciones basados en curvas elípticas.

1.5 Organización de la tesis

El contenido de la tesis ha sido organizado en 5 Capítulos. El Capítulo 1 presenta los antecedentes y motivación del presente trabajo. El capítulo 2 presenta la introducción a criptografía de llave pública y de llave privada. También se da una breve explicación sobre campos finitos, con el objetivo de comprender algunas operaciones realizadas durante el desarrollo de la tesis. Se presentan los algoritmos necesarios para efectuar las operaciones aritméticas de campos finitos binarios, así como las operaciones de traza, media traza y raíz cuadrada que son requeridas para poder llevar a cabo la implementación en software del algoritmo de bisección de punto. El Capítulo 3 presenta las bases matemáticas necesarias para la definición de las operaciones requeridas para operar con los puntos pertenecientes a la curva elíptica. Se define la operación de multiplicación escalar y la bisección de

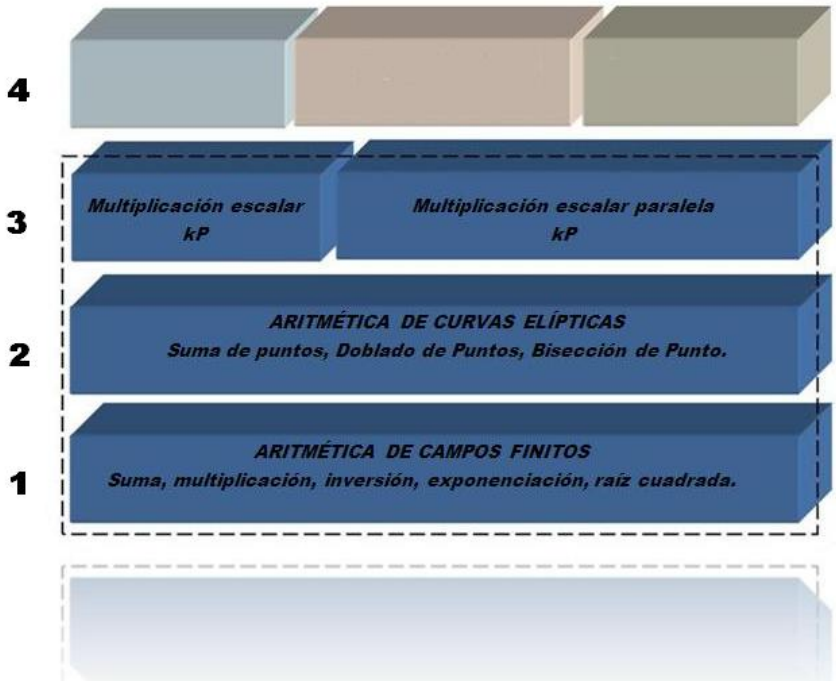

Figura 1.1: Modelo de la arquitectura en la que se basa la implementación.

punto en curvas elípticas. En el Capítulo 4 se describen las tecnologías multinúcleo, su arquitectura y se plantean las diferencias contra otros tipos de multiprocesamiento. El Capítulo 5 presenta la formulación paralela de la multiplicación escalar en curvas binarias. La paralelización de la multiplicación requiere de una recodificación del escalar k para poder hacer uso simultáneo del doblado y de la bisección de punto. Se presenta también la evaluación de los resultados obtenidos y tiempos resultantes para el cálculo de la multiplicación escalar. Finalmente se plantean las posibles extensiones futuras del trabajo realizado.

2

Conceptos básicos

En este capítulo se presentan algunos conceptos básicos de criptografía, así como la teoría matemática.

La criptografía es la ciencia de cifrar y descifrar datos mediante técnicas matemáticas [7]. Tiene el objetivo de permitir la comunicación de dos o más entidades de forma segura a través de un canal de comunicación, para cumplir con este objetivo se definen servicios de seguridad que deben satisfacer las aplicaciones, por ejemplo: confidencialidad, integridad, disponibilidad, autenticación, firmas digitales, estampas de tiempo, no repudio [12]. Las cuales se definen informalmente a continuación:

- **Confidencialidad.** Este servicio tiene como objetivo permitir que los datos puedan ser vistos o modificados únicamente por las entidades que están autorizadas para ello.
- **Integridad.** Este servicio protege la información contra modificaciones, alteraciones, borrado, inserción y, en general, contra todo tipo de acción que atente contra la integridad de los datos. Se debe aclarar que, estrictamente hablando, la integridad como tal no se puede garantizar. Lo que sí se puede garantizar es si la información sufre alguna alteración, ésta pueda ser detectada.
- **Disponibilidad.** Consiste en que los datos siempre se encuentren disponibles para su uso.

- **Autenticación.** Este servicio consiste en garantizar que las partes o entidades participantes en una comunicación sean las que dicen ser. Es decir, consiste básicamente en el proceso de identificación de una parte ante las demás, de una manera no controversial y demostrable [5]. Estrictamente hablando existen dos tipos de autenticación: autenticación de identidad o identificación, y autenticación de origen de datos. Este último tipo de autenticación se refiere a la certeza de que los datos hayan salido de donde se supone que deben de haber salido y no exista la posibilidad de haber suplantado el origen y que en realidad los datos tengan un origen distinto al supuesto. La autenticación, extendida como proceso de identificación, se clasifica en tres tipos: de acuerdo a la naturaleza de los elementos en que se basa su implementación:

  - En algo que se sabe.
  - En algo que se tiene.
  - En algo que se es.

  En el primer caso la autenticación pueden basarse en algo que se aprende o memoriza, tal como una contraseña o palabra clave. Al ser revelado a la parte ante la que se desea identificarse, se demuestra ser quien se dice ser. En el segundo caso, la autenticación se basa en algo que se posee como un generador de números aleatorios, una clave física o cualquier otro objeto tangible. En el tercer caso, la identidad se demuestra comparando patrones relacionados a alguna característica inherente a la naturaleza de la entidad que se autentica. Si se trata de una persona, una característica inherente a su naturaleza podría ser sus huellas digitales, su voz, etc. Este tipo de autenticación también se conoce como biométrica.

  La autenticación puede ser directa o indirecta. Es directa si en el proceso de autenticación sólo intervienen las partes interesadas o que se van a autenticar. Es decir, no interviene ninguna tercera parte actuando como juez. Es indirecta si en el proceso interviene una tercera parte confiable que actúa como autoridad o juez que avala la identidad de las partes.

  También la autenticación puede ser unidireccional o mutua. Es unidireccional si basta que una de las partes se autentique ante la otra y no es necesario que la otra se autentique, a su vez,

ante la primera. Es mutua cuando se requiere que ambas partes se autentiquen entre sí. El servicio de autenticación está íntimamente relacionado al de control de acceso.

- **Firma Digital.** Un método criptográfico que permite vincular la identidad de una entidad con respecto a un mensaje dado.
- **Estampas de tiempo.** Es un identificador simple que sirve para identificar cada transacción de manera única.
- **No repudio.** El no repudio proporciona protección contra la posibilidad de que algunas de las partes involucradas en una comunicación niegue haber enviado o recibido un mensaje u originado o haber sido el destinatario de una acción. Normalmente, para implementar este servicio se utilizan esquemas de clave pública tales como las firmas digitales, pero no se restringe a ellas; también se pueden utilizar técnicas de cifrado de clave pública y de clave secreta pero, en esta última, siempre que se utilice una tercera parte confiable.

Por lo general para poder hacer uso de los servicios de seguridad es necesario el uso de esquemas criptográficos, los cuales, se basan en la existencia de una clave. La criptografía clásica se basa en el uso algoritmos simétricos que consisten de una clave privada, la criptografía moderna hace uso de algoritmos asimétricos que son los que utilizan un par de llaves, una clave privada y una clave pública. En las siguientes secciones se da un breve descripción de ambos esquemas.

2.1 Criptografía simétrica

Se trata de un esquema criptográfico que utiliza una misma clave para cifrar y descifrar los datos [12]. En la figura 2.1 se ilustra el modelo de la criptografía simétrica que consiste en una clave en común entre dos entidades; Alicia (emisor) y Beto (receptor). Alicia y Beto previamente se deben poner de acuerdo para establecer la clave, Alicia toma la clave y cifra los datos que son enviados a Beto, quien realiza el proceso de descifrar los datos haciendo uso de la misma clave privada.

La seguridad de este tipo de esquemas se basa en la clave, es decir, no es ventaja para el atacante conocer el tipo de algoritmo utilizado sin que éste conozca a clave.

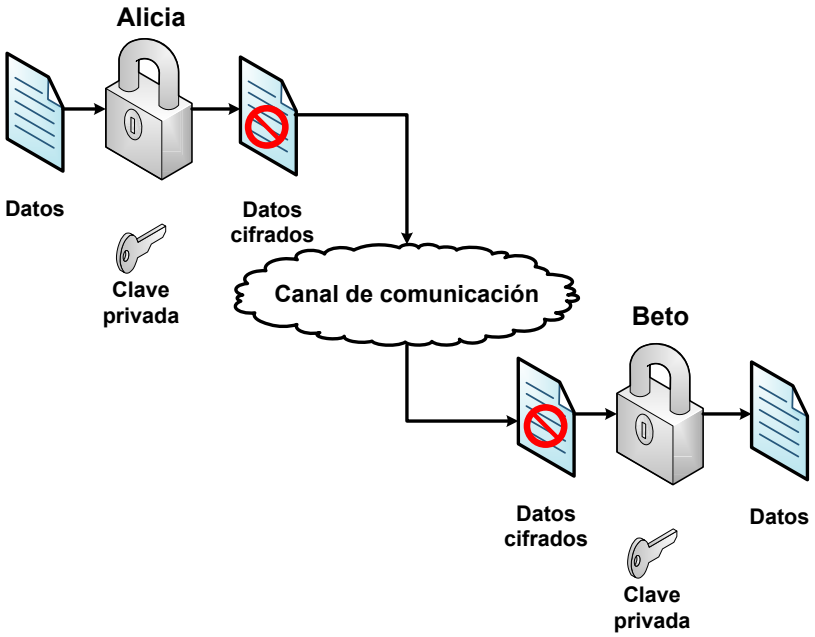

Figura 2.1: Modelo de la criptografía simétrica.

Los esquemas de cifrado simétrico a su vez se dividen en dos tipos: cifradores por flujo y cifradores por bloque. Los cifradores por flujo son algoritmos que realizan el cifrado bit a bit, son utilizados en aplicaciones en donde el flujo de datos se producen en tiempo real y en pequeños fragmentos [15]. Ejemplos de éste tipo de cifradores son RC4, A5, entre otros.

Los cifradores por bloques son algoritmos que procesan los datos por grupos de bits de logitud fija, llamados bloques, a los cuales se les aplican una tranformación invariante. DES, AES, CAST, DEAL, son algunos ejemplos de cifradores por bloques.

2.2 Criptografía asimétrica

También llamada criptografía de clave pública, fue propuesta por los investigadores Whit Diffie y Martin Hellman en 1976 y constituye la base de la criptografía moderna.

Es el método criptográfico que utiliza un par de claves, llamadas clave pública y clave privada. La clave privada es conocida únicamente por la entidad propietaria mientras que la clave pública puede ser conocida por todos los usuarios.

La figura 2.2 ilustra el modelo de clave pública. Alicia y Beto poseen sus pares de claves respectivamente. Alicia cifra los datos con la llave pública de Beto con el fin que sólo Beto pueda conocer el contenido de la información enviada. Los datos se envían a Beto, quien, los recibe cifrados y utiliza su clave privada para descifrarlos.

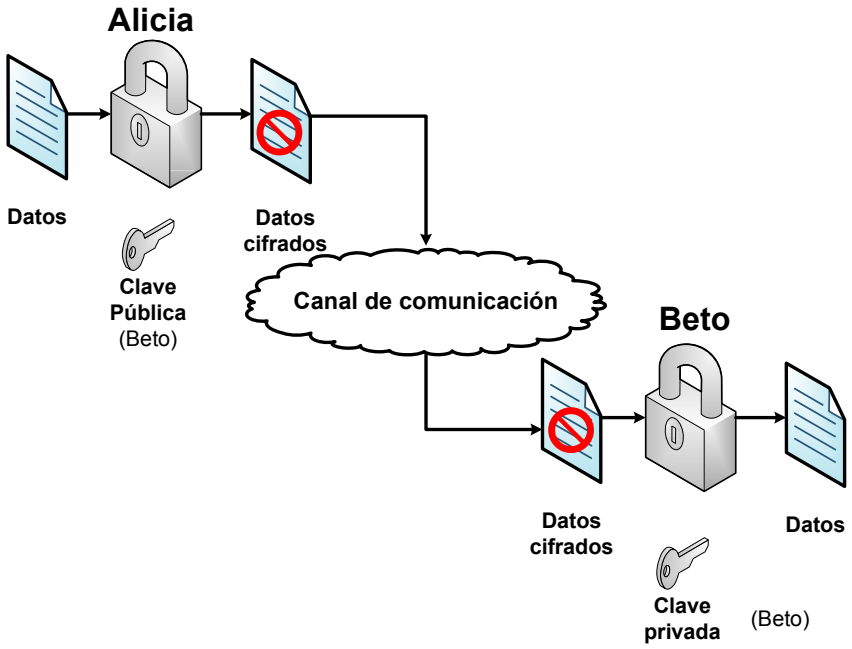

Figura 2.2: Modelo de la criptografía asimétrica.

El funcionamiento de este tipo de esquemas se basa en funciones matemáticas cuyo cálculo directo es simple, pero el cálculo inverso es muy complejo computacionalmente hablando. Un ejemplo típico es la multiplicación de números primos. En la siguiente operación se supone que p y q son dos primos.

$$(p, q) \rightarrow m = p \bullet q$$

La operación anterior es muy rápida, pero la operación inversa, es decir, dado m hallar p y q, es de complejidad exponencial en la longitud de m.

En la criptografía asimétrica se tienen tres ejemplos muy comunes de familias de algoritmos: los que se basan en la factorización de números primos muy grandes que es el caso de los sistemas RSA y el Rabin-Williams, en el problema del logaritmo discreto como los sistemas DSA ('Digital Signature

Algorithm'), DH ('Diffie-Hellman'), ElGamal y Nyberg-Rueppel, y por último en el problema del logaritmo discreto en curvas elípticas.

La criptografía asimétrica, aunque solventa desventajas de la criptografía simétrica como por ejemplo la distribución de llaves, también tiene puntos débiles como por ejemplo:

1. El tiempo de ejecución de los algoritmos de cifrado es mayor, es decir, son más lentos que los algoritmos de criptografía simétrica.

2. El tamaño de la llave es mayor que las correspondientes a criptografía simétrica.

A continuación se mostrará más a detalle el protocolo criptográfico para el intercambio de clave el cual es utilizado en la tesis en parte de comprobación de resultados.

2.3 Intercambio de claves Diffie-Hellman

El algoritmo Diffie-Hellman permite que dos entidades se pongan de acuerdo en una clave en común mediante un canal de comunicación no cifrado [4]. El acuerdo se lleva a cabo sin que una tercera entidad que tiene acceso completo a los datos pueda conocerlo o calcularlo, en un tiempo razonable.

En la figura 2.3 se muestra el protocolo de Diffie-Hellman, en donde Alicia y Beto son las entidades que quieren realizar el acuerdo.

1. Alicia y Beto escogen un grupo cíclico G y un elemento generador $g \in G$.

2. Alicia escoge un número natural, que llamaremos a.

3. Beto escoge otro número natural, denominado b.

4. Alicia calcula g^a y se la envía a Beto mediante un canal de comunicaciones no cifrado.

5. Beto calcula g^b y se la envía a Alicia mediante un canal de comunicaciones no cifrado.

6. Alicia entonces calcula el nuevo número mediante la fórmula: $C_a = ((g^b)^a) mod\ p$

7. De igual forma, Beto calcula su nuevo número $C_b = ((g^a)^b) mod\ p$

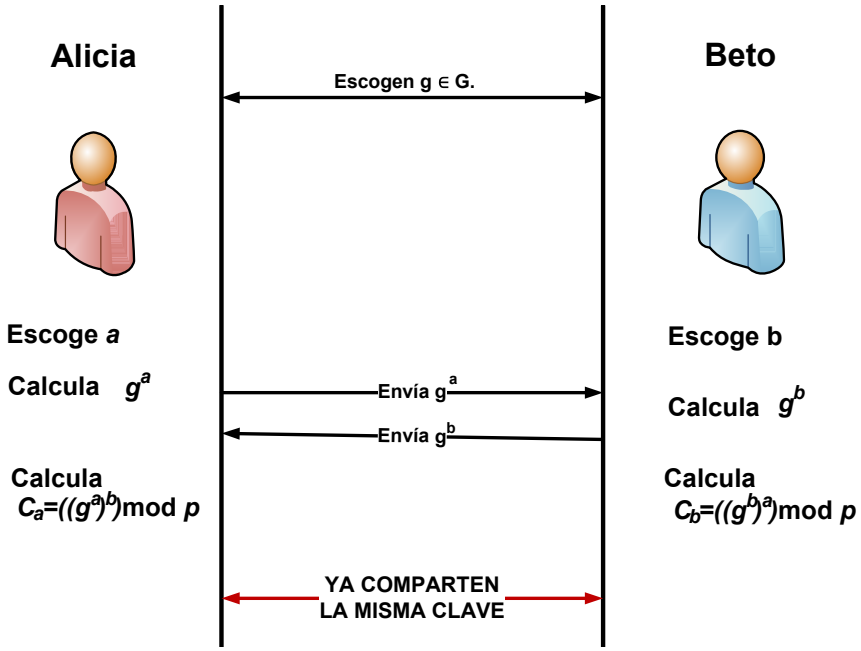

Figura 2.3: Protocolo de Diffie-Hellman.

Los números C_a y C_b son el mismo, en este momento Alicia y Beto tienen ya un secreto compartido, que un tercero no puede conocer, pese a haber tenido acceso a todo el intercambio de información.

2.4 Campos finitos

2.4.1 Grupos

Un grupo G, algunas veces denotado por $G, \bullet$; es un conjunto de elementos con una operación binaria $\bullet$, que asocia a cada par ordenado $(a,\ b)$ de elementos en G un elemento $(a\ \bullet\ b)$, de tal forma que los siguientes axiomas se cumplan:

1. Cerradura. Si a y b pertenecen a G, entonces $a\ \bullet\ b$ también se encuentra en G

2. Asociativa. $a\ \bullet\ (b\ \bullet\ c) = (a \bullet b) \bullet c$ para toda a, b, c en G.

3. Elemento de identidad. Existe un elemento e en G tal que $a \bullet e = e \bullet a = a$ para toda a en G.

4. Elemento inverso. Para cada a en G existe un elemento a' en G tal que $a \bullet a' = a' \bullet a = e$

2.4.2 Anillos

Un anillo A algunas veces denotados por $\{A, +, \bullet\}$ es un conjunto de elementos con dos operaciones binarias, llamadas adición y multiplicación, de tal forma para todas $\{a, b, c\} \in A$ los siguientes axiomas se cumplen:

1. A es un grupo abeliano con respecto a la adición si satisface los axiomas relacionados con dicho grupo. En el caso de un grupo aditivo, el elemento identidad es 0 y la inversa de a es $-a$.

2. Cerradura bajo multiplicación: Si a y b pertenecen a A, entonces ab también están en A.

3. Asociativa para la multiplicación: $a(bc) = (ab)c$ para toda $a, b, c \in A$.

4. Leyes distributivas: $a(b + c) = ab + ac$ para toda $a, b, c \in A$. $(a + b)c = ac + bc$ para toda $a, b, c \in A$.

2.4.3 Campos

Un campo $\mathbb{F}$ es un anillo conmutativo con elemento unidad en el que todo elemento distinto de cero tiene inverso multiplicativo [8]. A veces denotado como $\{\mathbb{F}, +, \bullet\}$ el conjunto de elementos con dos operaciones binarias,llamadas adición y multiplicación, de tal forma para todas $a, b, c \in \mathbb{F}$ los siguientes axiomas se cumplen:

1. $a + b$ está en $\mathbb{F}$

2. $a + b = b + a$

3. $(a + b) + c = a + (b + c)$

4. Existe un elemento 0 en $\mathbb{F}$ tal que $a + 0 = a$

5. Existe un elemento $-a$ en $\mathbb{F}$ tal que $-a + a = 0$

6. $a\ b$ está en $\mathbb{F}$

7. $a(bc) = (ab)c$ para toda $a, b, c\ \in \mathbb{F}$.

8. Leyes distributivas:

   $a(b + c) = ab + ac$ para toda $a, b, c\ \in \mathbb{F}$.

   $(a + b)c = ac + bc$ para toda $a, b, c\ \in \mathbb{F}$.

- Inverso multiplicativo: para cada a en $\mathbb{F}$, excepto $\{0\}$, existe un elemento a^{-1} en $\mathbb{F}$ tal que $aa^{-1} = (a^{-1})a = 1$.

2.4.4 Campos finitos

Un campo finito o campo de Galois denotado por $GF(q = p^n)$ o $\mathbb{F}_{p^n}$, es un campo de característica p, con un número q de elementos y también puede ser denotado por $\mathbb{F}_q$. Un campo finito es un sistema algebraico que consiste del conjunto GF junto con los operadores binarios $+$ y $\bullet$ definidos en GF que satisfacen los siguientes axiomas:

1. Los elementos 0 y 1 se encuentran en GF.

2. GF es un grupo abeliano respecto a la operación $+$.

3. GF excepto $\{0\}$ es un grupo abeliano respecto a la operación $\bullet$.

4. Para todo x, y y z en GF, $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$ y $x + (y \bullet z) = (x + y) \bullet (x + z)$.

Los campos finitos tienen gran importancia en criptografía ya que varios algoritmos criptográficos se basan en operaciones aritméticas realizadas en estos campos. Los campos son estructuras matemáticas que cuentan con operaciones aritméticas básicas tales como la suma, resta, multiplicación y la inversión de elementos no cero. La división está definida como el producto de un número por el inverso de otro.

2.4.5 Campos finitos binarios

Los campos $GF(2^m)$ son también llamados campos binarios o campos finitos de característica dos. Una forma de representar sus elementos es usando una base polinomial binaria de grado a lo más $(m-1) > 0$, donde sus coeficientes se encuentran en el campo $\mathbb{F}_2 = \{0, 1\}$. Así $\forall a, b \in GF(2^m)$ se tiene que:

$$a = \sum_{i=0}^{m-1} a_i x_i$$
$$b = \sum_{i=0}^{m-1} b_i x_i$$
donde $a_i, b_i \in \{0, 1\}$

2.5 Aritmética de campos finitos binarios

En esta sección se presentan los algoritmos necesarios para llevar a cabo las operaciones aritméticas básicas en campos finitos. Es fundamental contar con algoritmos que permitan realizar las operaciones aritméticas de manera eficiente, ya que en este tipo de aritmética descansan las operaciones aritméticas de puntos en la curva elíptica.

Por lo tanto, si se pretende obtener una implementación eficiente de la multiplicación escalar en curvas elípticas, resultará vital que las operaciones aritméticas de campos finitos: suma, multiplicación, elevar al cuadrado, raíz cuadrada, traza y media traza, sean computables de la manera más eficiente posible.

Sea $f(z)$ un polinomio binario irreducible de grado m, que se puede escribir como $f(z) = z^m + r(z)$. Los elementos de $\mathbb{F}_{2^m}$ son polinomios binarios de grado $(m-1)$.

La multiplicación se hace módulo $f(z)$. Un elemento de campo $a(z) = a_{m-1}Z^{m-1} + \ldots + a_2 z^2 + a_1 z + a_0$ se asocia con el vector binario $a = (a_{m-1}, \ldots, a_2, a_1, a_0)$ de longitud m. Sea $t = \lceil m/W \rceil$, y sea $s = tW - m$. En software, a puede ser almacenada en una matriz de t $W-$bit palabras: $A = (A[t-1], \ldots, A[2], A[1], A[0])$, donde el último bit de $A[0]$ es a_0, y más a la izquierda de s bits de $A[t-1]$ no han sido utilizados (siempre 0).

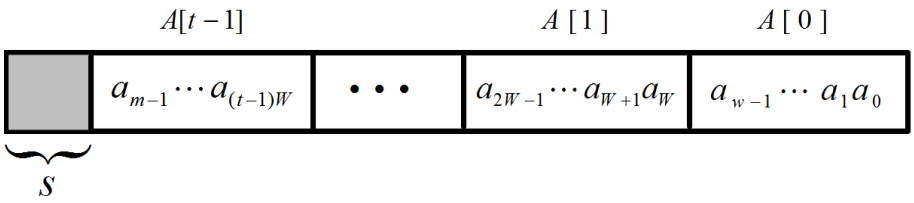

Figura 2.4: Representación binaria de $a \in \mathbb{F}_{2^m}$ como un arreglo A de un tamaño de W bit.

2.5.1 Suma

La suma es una de las operaciones más sencillas y rápidas debido a que en campos finitos binarios no hay acarreos, por lo que las sumas simplemente se hacen mediante operadores XOR.

A continuación se muestra el algoritmo para efectuar la suma de dos elementos del campo donde se tienen representados con t palabras a los elementos del campo:

Algoritmo 2.1 Suma en $\mathbb{F}_{2^m}$.
Require: Dos polinomios binarios $a(z)$ y $b(z)$ hasta un grado $m-1$.
Ensure: $c(z) = a(z) + b(z)$.
1: **for** $i = 0$ to $t-1$ **do**
2: $\quad C[i] \leftarrow A[i] \oplus B[i]$
3: **end for**
4: Retorna (c)

2.5.2 Multiplicación

El Algoritmo 2.2 muestra el método de multiplicación de regla de Horner de izquierda a derecha que está basado en lo siguiente.

$$a(z) \bullet b(z) = a_{m-1}z^{m-1}b(z) + \ldots + a_2 z^2 b(z) + a_1 z b(z) + a_0 b(z).$$

En cada iteración i el algoritmo calcula $z^i b(z)$ mod $f(z)$ y suma el resultado con lo acumulado en c si $a_i = 1$. Si $b(z) = b_{m-1}z^m + \ldots + b_2 z^2 + b_1 z + b_0$, entonces

$$b(z) \bullet z = b_{m-1}z^m + b_{m-2}z^{m-1} + \ldots + b_2 z^3 + b_1 z^2 + b_0 z$$
$$\equiv b_{m-1}r(z) + (b_{m-2}z^{m-1} + \ldots + b_2 z^3 + b_1 z^2 + b_0 z)(\text{ mod } f(z)).$$

Así $b(z) \bullet z$ mod $f(z)$ es calculado haciendo un corrimiento a la izquierda del vector $b(z)$, seguido de la suma de $r(z)$ con $b(z)$ si el bit de orden superior es 1.

Algoritmo 2.2 Multiplicación en $\mathbb{F}_2^m$ por el método "Corrimiento y Suma".

Require: Dos polinomio binario $a(z)$ y $b(z)$ hasta un grado $m-1$.
Ensure: $c(z) = a(z) \bullet b(z)$ mod $f(z)$

```
1: if a_0 = 1 then
2:     c ← b
3: else
4:     c ← 0
5: end if
6: for i = 1 to m − 1 do
7:     b ← b • z mod f(z)
8:     if a_i = 1 then
9:         c ← c + b
10:    end if
11: end for
12: Retorna (c)
```

2.5.2.1. Karatsuba-Ofman

La multiplicación de Karatsuba-Ofman para polinomios en $GF(2^m)$ Se trata de una técnica de divide y vencerás [11]. Es considerado como uno de los algoritmos más rápidos para multiplicar números de gran tamaño. Para la multiplicación de polinomios, ambos operandos tienen que ser expresados como la suma de otros dos. Si la longitud de los operandos es impar que tienen que ser rellenados con '0', por lo tanto requiere $log_2(m)$ pasos de la iteración de polinomios de grado $(m-1)$.

Sea $A = (a_0, a_1, \ldots, a_{m-1})$ y $B = (b_0, b_1, \ldots, b_{m-1})$, la representación binaria de dos polinomios de grado $(m-1)$. Los operandos A y B se pueden descomponer en dos partes de igual tamaño A_1A_0 y B_1B_0, que representan los $m/2$ bits más significativos y menos significativos de A y B, respectivamente. Se puede expresar como la suma de otros dos como se muestra a continuación:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = \sum_{\frac{m}{2}}^{m-1} a_i x^i + \sum_{i=0}^{\frac{m}{2}-1} a_i x^i$$

$$x^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} a_{i+\frac{m}{2}} x^i + \sum_{i=0}^{\frac{m}{2}-1} a_i x^i = x^{\frac{m}{2}} A_1 + A_0$$

$$B(x) = \sum_{i=0}^{m-1} b_i x^i = \sum_{i-\frac{m}{2}}^{m-1} b_i x^i + \sum_{i=0}^{\frac{m}{2}-1} b_i x^i$$

$$x^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} b_{i+\frac{m}{2}} x^i + \sum_{i=0}^{\frac{m}{2}-1} b_i x^i = x^{\frac{m}{2}} B_1 + B_0$$

Entonces el producto $P(x) = A(x) \bullet B(x)$ puede ser calculado de la siguiente manera:

$$A(x)B(x) = A_0B_0 + A_1B_1x^m + (A_1B_0 + A_0B_1)x^{\frac{m}{2}}$$

La ecuación anterior necesita cuatro multiplicaciones de $m/2$ bits para calcular el producto de $P(x)$. Esta ecuación puede ser modificada de la siguiente forma:

$$A_1B_0 + A_0B_1 = (A_1 + A_0)(B_0 + B_1) - A_1B_1 - A_0B_0$$

$$A(x)B(x) = A_0B_0 + A_1B_1x^m + (A_0B_0 + A_1B_1 + (A_1 + A_0)(B_1 + B_0))x^{\frac{m}{2}}$$

De esta manera se puede realizar la multiplicación de $A(x)B(x)$ con solo 3 multiplicaciones y 4 sumas:

1. Calcular la multiplicación: A_0B_0
2. Calcular la multiplicación: A_1B_1
3. - Sumar: $(A_1 + A_0)$
   - Sumar: $(B_1 + B_0)$
   - Multiplicar los resultados de las sumas anteriores: $(A_1 + A_0)(B_1 + B_0)$
   - Sumar los productos que previamente se habían calculado:

     (A_1B_1) y (A_0B_0)

Cabe señalar que los operandos de cada una de las tres multiplicaciones que se efectuan son aproximadamente de la mitad de bits comparado con los de la multiplicación original. De esta

manera, podría continuarse aplicando el método recursivamente hasta tener un tamaño conveniente de operandos como para efectuar los productos de manera sencilla mediante el uso del multiplicador del sistema o algún otro método. Las cuatro sumas resultan insignificantes comparadas contra el tiempo requerido para la operación de multiplicación de números grandes.

2.5.3 Elevar al cuadrado

Elevar al cuadrado se trata de una operación lineal, además de ser mucho más rápida que una multiplicación arbitraria de dos polinomios: Por ejemplo, si $a(z) = a_{m-1}z^{m-1} + \ldots + a_2z^2 + a_1z + a_0$

$$a(z)^2 = a_{m-1}z^{2m-2} + \ldots + a_2z^4 + a_1z^2 + a_0$$

Es decir, la representación binaria de $a(z)^2$ se obtiene insertando un bit de valor 0 entre cada bit subsecuente de la representación de $a(z)$. La figura 2.5 ilustra este proceso mientras que el algoritmo 2.3 muestra como se podría llevar a cabo esta operación con un polinomio de t palabras de longitud W de 32 bits.

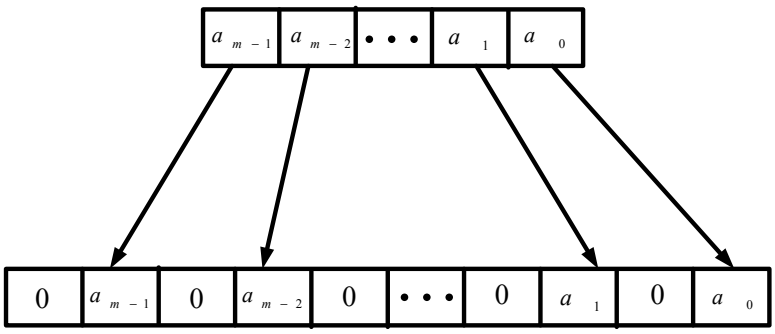

Figura 2.5: Elevar al cuadrado polinomial $a(z) = a_{m-1}z^{m-1} + \ldots + a_2z^2 + a_1z + a_0$

Algoritmo 2.3 Elevar al cuadro polinomial (Tamaño de palabra $W = 32$).

Require: Un polinomio binario $a(z)$ hasta un grado $m-1$.
Ensure: $c(z) = a(z)^2$
1: Precomputo. Para cada byte $d = (d_7, \ldots, d_1, d_0)$, calcular $T(d) = (0, d_7, \ldots, 0, d_1, 0, d_0)$.
2: **for** $i = 0$ to $t-1$ **do**
3: $A[i] = u_3, u_2, u_1, u_0$ donde cada u_j es un byte.
4: $c[2i] \leftarrow (T(u_1), T(u_0)),\ c[2i+1] \leftarrow (T(u_3), T(u_2))$
5: **end for**
6: Retorna (c)

2.5.4 Raíz cuadrada

Una implementación eficiente de la función raíz cuadrada se deriva de la observación de que $\sqrt{c}$ puede ser expresada en términos de la raíz cuadrada de un elemento z [16].

Si $c = \sum_{i=0}^{m-1} c_i z^i \ \in \mathbb{F}_{2^m},\ c_i \ \in \{0,1\}$, entonces debido a que elevar al cuadrado es una operación lineal en $\mathbb{F}_{2^m}$, la raíz cuadrada de c puede ser escrita de la siguiente forma:

$$\sqrt{c} = \left(\sum_{i=0}^{m-1} c_i z^i\right)^{2^{m-1}} = \sum_{i=0}^{m-1} c_i (z^{2^{m-1}})^i$$

Separando a c en potencias pares e impares:

$$\begin{aligned}\sqrt{c} &= \sum_{i=0}^{(m-1)/2} c_{2i} (z^{2^{m-1}})^{2i} + \sum_{i=0}^{(m-3)/2} c_{2i+1} (z^{2^{m-1}})^{2i+1} \\ &= \sum_{i=0}^{(m-1)/2} c_{2i} z^i + \sum_{i=0}^{(m-3)/2} c_{2i+1} z^{2^{m-1}} z^i \\ &= \sum_{i\ par} c_i z^{i/2} + \sqrt{z} \sum_{i\ impar} c_i z^{(i-1)/2}\end{aligned}$$

Las ecuaciones anteriores permiten utilizar un método eficiente para calcular la raíz cuadrada de la siguente forma.

1. El vector c es expresado en dos vectores de la mitad de longitud cada uno: $c_{par} = (c_{m-1}, \ldots, c_4, c_2, c_0)$ y $c_{impar} = (c_{m-2}, \ldots, c_5, c_3, c_1)$, donde m es impar,
2. Se realiza la multiplicación de campo entre el vector c_{impar} con el valor precalculado de $\sqrt{z}$
3. Finalmente se suma el resultado con c_{par}

En el caso que el polinomio irreducible f sea un trinomio, el cálculo de $\sqrt{c}$ puede ser más rápido debido que la fórmula para $\sqrt{z}$ se deriva de f, donde $f(z) = z^m + z^k + 1$ el cual es un polinomio irreducible de grado m, con m primo y $m > 2$

Considerando que el valor de k es impar, notése que $1 \equiv z^m + z^k \pmod{f(z)}$.

$$\sqrt{z} \equiv z^{\frac{m+1}{2}} + z^{\frac{k+1}{2}} \pmod{f(z)}$$

De esta forma, el producto de $\sqrt{z} \bullet c_{odd}$ requiere dos operaciones de corrimiento a la izquierda y una reducción modular.

Ahora supongamos que k es par. Se observa que $z^m \equiv z^k + 1$ (mod $f(z)$). Luego, dividiendo por z^{m-1} y haciendo la raíz cuadrada, se obtiene

$$\sqrt{z} \equiv z^{-\frac{m-1}{2}}(z^{\frac{k}{2}} + 1) \pmod{f(z)}$$

Para calcular z^{-s} modulo $f(z)$, donde $s = \frac{m-1}{2}$, se hacen uso de congruencias $z^{-t} \equiv z^{k-t} + z^{m-t}$ (mod $f(z)$ para $1 \leq t \geq k$ para escribir a z^{-s} como la suma de algunas potencias positiva de z. Por lo tanto el producto de $\sqrt{z} \bullet c_{impar}$ puede ser realizado por medio de pocas operaciones de corrimiento a las izquierda y una reducción modular.

2.5.5 Traza

Para poder implementar la bisección de punto en curvas elípticas binarias se emplea la función traza. En campos finitos binarios $\mathbb{F}_{2^m}$ la función traza $Tr : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_2$ se define como:

$$Tr(c) = c + c^2 + c^{2^2} + +c^{2^3} + \cdots + +c^{2^{m-1}}$$

Cabe señalar que:

$$Tr(c) \in \{0, 1\}$$

La función traza tiene las siguientes propiedades:

Suponiendo que $c, d \in \mathbb{F}_{2^m}$

1. $Tr(c) = Tr(c^2) = Tr(c)^2$

2. La función traza es lineal, es decir, $Tr(c+d) = Tr(c) + Tr(d)$

3. Si $(u, v) \in \mathbb{G}$, entonces $Tr(u) = Tr(a)$

Una manera directa de calcular la función traza podría ser utilizando la definición de la función, pero no sería la mejor manera de hacerlo si lo que se busca es obtener una implementación eficiente. Basándose en la propiedad de linealidad de esta función, se toma un elemento aleatorio $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$, donde $c_i \in \{0, 1\}$.

$$Tr(c) = Tr(\sum_{i=0}^{m-1} c_i z^i) = \sum_{i=0}^{m-1} c_i Tr(z^i)$$

De tal manera que para calcular la traza se precomputan los valores $Tr(z^i)$ y solo se opera con los valores de $Tr(z^i)$ que son diferentes de cero. Para las curvas utilizadas en este trabajo resulta muy conveniente debido a que casi todos los valores de $Tr(z^i)$ son cero.

2.5.6 Media traza

Suponiendo que m es un entero impar. La función de media traza $H : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ está definida por:

$$H(c) = \sum_{i=0}^{\frac{m-1}{2}} c^{2^{2i}}$$

Propiedades de la función de media traza.

(i) $H(c+d) = H(c) + H(d)$ para todo $c, d \in \mathbb{F}_{2^m}$

(ii) $H(c)$ es la solución de la ecuación $x^2 + x = c + Tr(c)$

(iii) $H(c) = H(c^2) + c + Tr(c)$ para todo $c \in \mathbb{F}_{2^m}$

Suponiendo que $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$ con $Tr(c) = 0$; $H(c)$ es una solución de $x^2 + x = c$. Para encontrar $H(c)$ directamente de la definición se requiere $m-1$ elevaciones al cuadrado y $(m-1)/2$ sumas. Si el almacenamiento para $H(z^i) : 0 \leq i < m$ lo permite, entonces las propiedades de la

Media Traza pueden ser empleadas para obtener:

$$H(c) = H\left(\sum_{i=0}^{m-1} c_i z^i\right) = \sum_{i=0}^{m-1} c_i H(z^i).$$

Por lo que se requiere almacenar m elementos del campo, y el método necesita en promedio de $m/2$ sumas.

Tomando en cuenta la propiedad (iii)

$$H(c) = H(c^2) + c + Tr(c) \text{ para todo } c \in \mathbb{F}_{2^m}$$

Se puede hacer que $H(c) = H(c^{'}) + s$ donde $c^{'}$ tiene menos elementos diferentes de cero que c.

Para los elementos i pares: $H(z^i) = H(z^{\frac{i}{2}}) + z^{\frac{i}{2}} + Tr(z^i)$

Es decir, se puede eliminar el tamaño de la tabla de consulta a la mitad, debido a que se evita tener que almacenar los elementos i pares de $H(z^i)$.

El algoritmo 2.4 muestra cómo se obtiene la solución de la ecuación cuadrática

Algoritmo 2.4 Versión básica para resolver $x^2 + x = c$

Require: $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$ donde m es impar y $Tr(c) = 0$.

Ensure: La solución s de $x^2 + x = c$

1: Precomputo $H(z^i)$ para impar i, $1 \leq i \leq m-2$
2: $s \leftarrow 0$
3: **for** $i = (m-1)/2$ to 1 **do**
4: **if** $c_{2i} = 1$ **then**
5: $c \leftarrow z + z^i$
6: $s \leftarrow s + z^i$
7: **end if**
8: **end for**
9: $s \leftarrow + \sum_{i=1}^{(m-1)/2} c_{2i-1} H(z^{2i-1})$
10: Retorna (s)

3

Curvas elípticas

Desde la publicación de los artículos de Koblitz [9] y Miller [13] donde se propone el uso de las curvas elípticas sobre campos finitos dentro de la criptografía, se han diseñado varios criptosistemas que ayudan a solventar los inconvenientes de las criptografía basada en el problema de factorización ó del algoritmo discreto, manteniendo la misma seguridad computacional, e.g. la disminución del tamaño de claves.

En este capítulo se muestra la definición de curva elíptica, se presentan los diferentes tipos de curvas y se plantean las primitivas para poder realizar operaciones con los puntos pertenecientes a la curva elíptica.

3.1 Definición

Una curva elíptica denotada por E sobre un campo K, es definida por la ecuación de Weiestrass:

$$E : y^2 + a_1xy + a_3y = x^3 + a2x^2 + a_4x + a_6$$

Donde a_1, a_2, a_3, a_4, $a_6 \in K$ y $\Delta \neq 0$, donde Δ es el discriminante de E y es definido como sigue:

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6$$

$$d_2 = a_1^2 + 4a_2$$

$$d_4 = 2a_4 + a_1 a_3$$

$$d_6 = a_3^2 + 4a_6$$

$$d_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2$$

Las constantes d_2, d_4, d_6, d_8 definen el valor del discriminante de la curva E, estas constantes dependen del valor de las constantes a_1, a_2, a_3, a_4, a_6

Si L es una extensión del campo K, entonces el conjunto de puntos racionales L sobre E es

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x - a_6\} \cup \{\mathcal{O}\}$$

Donde $\{\mathcal{O}\}$ se define como el punto al infinito.

3.1.1 Curva elíptica en $GF(2^m)$

El campo sobre el que esté definida la curva elíptica, así como el valor de las constantes a_1, a_2, a_3, a_4, a_6, determinan el tipo de curva elíptica que se está manejando. Mediante un cambio admisible de variables se pueden encontrar formas reducidas de la ecuación de Weiestrass. Para campos finitos binarios y para el caso de curvas elípticas no-supersingulares, se tiene una ecuación reducida definida de la siguiente manera:

Sea E una curva elíptica sobre un campo finito binario $GF(2^m)$ definida por la siguiente equación:

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

donde a, $b \in GF(2^m)$, $b \neq 0$. Un punto P de coordenadas (x, y), pertenece a E si satisface la ecuación anterior.

3.2 Orden de la curva

Sea E una curva elíptica definida sobre el campo finito $\mathbb{F}_q$. El orden de E sobre $\mathbb{F}_q$ se trata del número de elementos en $E(\mathbb{F}_q)$ y es denotado por $\#E(\mathbb{F}_q)$. De la ecuación de Weierstrass se tienen

a lo más dos soluciones para cada $x \in \mathbb{F}_q$, entonces $\#E(\mathbb{F}_q) \in [1, 2q+1]$.

El teorema de Hasse sirve para calcular el orden de una curva sobre un campo finito.

Teorema. (Hasse)[12]. Sea E una curva elíptica definida sobre $\mathbb{F}_q$, entonces:

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

El intervalo $[q+1-2\sqrt{q},\ q+1+2\sqrt{q}]$ es conocido como el intervalo de Hasse. Una formulación alternativa del teorema de Hasse es $\#E(\mathbb{F}_q) = q + 1 - t$ con $|t| \leq 2\sqrt{q}$, donde t es llamado traza de E sobre $\mathbb{F}_q$, $2\sqrt{q}$ es pequeño en relación a q, entonces $\#E(\mathbb{F}_q) \approx q$.

3.3 Orden del punto

El menor número de veces que hay que sumar un punto P consigo mismo para obtener el cero de la curva E, es conocido como orden de un punto. Se trata básicamente del mínimo entero k tal que $kP = \mathcal{O}$. El orden de cualquier punto siempre divide el orden de la curva $\#E(\mathbb{F}_q)$, esto garantiza que si r y l son enteros, entonces $rP = lP$ si y sólo si $r \equiv l \mod q$.

3.4 Operaciones sobre curvas elípticas

3.4.1 Suma de puntos

Para las curvas no-supersingulares $E/\mathbb{F}_{2^m} : y^2 + xy = x^3 + ax^2 + b$. La suma de puntos se expresa de la siguiente manera:

Si $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ y $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, $P \neq \pm Q$, entonces para obtener el punto de la curva elíptica (x_3, y_3) correspondiente a la suma de $P + Q = (x_3, y_3)$ se deben efectuar las siguientes operaciones:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

donde:

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

3.4.2 Doblado de puntos

Para las curvas no-supersingulares $E/\mathbb{F}_{2^m} : y^2 + xy = x^3 + ax^2 + b$. El doblado de punto se define de la siguiente manera: Si $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, $P \neq -P$. Entonces para obtener el punto de la curva elíptica (x_3, y_3) correspondiente al doblado de P, es decir, $2P = (x_3, y_3)$, se efectúan las siguientes operaciones:

$x_3 = \lambda^2 + \lambda + a$

$y_3 = x_1^2 + \lambda x_3 + x_3$

donde:

$\lambda = (x_1 + y_1)/x_1$

3.4.3 Bisección de puntos

Suponiendo que $P = (x, y)$ es un punto de la curva elíptica E donde $P \neq -P$ y que para efectuar el doblado de punto $Q = 2P = (u, v)$, en coordenadas afines, se realizan las siguientes operaciones:

$Q = 2P$ donde $P = (x, y)$ y $Q = (u, v)$

- $\lambda = x + y/x$
- $u = \lambda^2 + \lambda + a$
- $v = x^2 + u(\lambda + 1)$

La bisección de punto se define de la siguiente manera, dado un punto $Q = (u, v)$ se debe encontrar el punto P de coordenadas (x, y), tal que $Q = 2P$ [6]. Por lo que la bisección de punto podría considerarse como la operación inversa al doblado.

La idea para la bisección de punto consta de 3 pasos básicos, primero se debe resolver la ecuación cuadrática para lambda:

$$u = \lambda^2 + \lambda + a$$

Una vez que se tiene el valor de λ se puede resolver la siguiente ecuación para x:

$$v = x^2 + u(\lambda + 1)$$

Finalmente, se resuelve la siguiente ecuación para obtener el valor de y:

$$\lambda = x + y/x$$

A continuación se muestran los pasos para calcular la bisección de punto.

1. $\hat{\lambda}^2 + \hat{\lambda} = u + a$
2. $t = v + u\hat{\lambda} = u(u + u + \frac{v}{u}) + u\hat{\lambda} = u(u + \lambda_Q + \hat{\lambda})$
3. If $Tr(t) = 0$, then $\lambda_P \leftarrow \hat{\lambda}$, $x \leftarrow \sqrt{t + u}$
4. else $\lambda_P \leftarrow \hat{\lambda} + 1$, $x \leftarrow \sqrt{t}$
5. Return (x,λ_P)

La salida del algoritmo de bisección regresa (x,λ_P) que son las coordenadas en representación lambda equivalentes a las coordenadas (x, y) del punto P. La representación lambda es útil cuando se deben efectuar varias bisecciones de punto, como para el caso de la multiplicación escalar que utiliza bisección de punto, donde se bisecta un punto m veces.

Dado un punto P de coordenadas (x, y) se puede representar también como un punto P de coordenadas (x, λ_P), donde:

$$\lambda_P = x + y/x$$

para volver a coordenadas afines se obtiene nuevamente el valor de y con una multiplicación y una suma:

$$y = x(\lambda_P + x)$$

3.4.4 Multiplicación escalar

La multiplicación escalar es la operación básica para los criptosistemas basados en curvas elípticas y esta operación resulta ser la más costosa en cuanto a desempeño. De tal manera que resulta importante implementar la multiplicación escalar de una manera eficiente.

La multiplicación escalar kP donde k es un entero en el intervalo $[1, n-1]$ y P es un punto en la curva elíptica E definida sobre el campo finito binario $\mathbb{F}_{2^m}$, es el resultado de sumar $P+P+\cdots+P$, k veces. Existen varios métodos para poder calcular esta operación, algunos de ellos son para los casos donde se conoce el punto P a priori y se puede efectuar precómputo, otros métodos aplican para cuando P no se conoce previamente.

El método básico para efectuar la multiplicación escalar para cuando P es un punto desconocido es el método de "suma y doblado". Existen dos tipos de multiplicadores de suma y doblado, el que va de derecha a izquierda y el de izquierda a derecha:

Algoritmo 3.1 Multiplicación Doblado-y-Suma de izquierda a derecha.
Require: $k = (k_{t-1}, ..., k_1, k_0)_2, P \in E(\mathbb{F}_{2^m})$.
Ensure: kP.
1: $Q \leftarrow P$
2: **for** $i = t-2$ to 0 do **do**
3: $Q \leftarrow 2Q$
4: **if** $k_i = 1$ entonces **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: **end for**
8: Regresar (Q)

El algoritmo 3.1 realiza en promedio $\frac{m-1}{2}$ sumas y $m-1$ doblados, esto es debido a que el número esperado de unos en la representación binaria del escalar k es $\frac{t}{2} \approx \frac{m}{2}$, donde t es el número de bits del escalar k. Como este método va de izquierda a derecha y se supone el bit más significativo del escalar k es uno, se inicializa el acumulador Q con P y se comienza a iterar desde $t-2$ hasta 0, esto es con el objetivo de ahorrar una suma y un doblado.

El algoritmo 3.2 realiza en promedio $\frac{m}{2}$ sumas y m doblados. Como este método va de derecha

Algoritmo 3.2 Multiplicación Suma-y-Doblado de derecha a izquierda.

Require: $k = (k_{t-1}, ..., k_1, k_0)_2, P \in E(\mathbb{F}_{2^m})$.
Ensure: kP.
1: $Q \leftarrow \mathcal{O}$
2: **for** $i = 0$ to $t - 1$ **do**
3: **if** $k_i = 1$ **then**
4: $Q \leftarrow Q + P$
5: **end if**
6: $P \leftarrow 2P$
7: **end for**
8: Return (Q)

a izquierda, se debe iterar desde 0 hasta $t - 1$. A diferencia del algoritmo 3.1 donde los doblados se hacen en el acumulador Q, para el algoritmo 3.2 los doblados se aplican en el punto P.

Como puede verse en los algoritmos 3.1 y 3.2, se dobla incondicionalmente y se realiza la suma en caso de que el bit k_i se encuentra encendido. Existen varios métodos para mejorar el tiempo de ejecución de la multiplicación escalar. Algunos de ellos se enfocan en disminuir el número de sumas que se realizan mediante la reducción de la cantidad de unos del escalar k. Estos métodos como por ejemplo el NAF y wNAF, recodifican k de tal manera que se incrementan los simbolos con los que se puede representar k y se reduce la densidad de los dígitos diferentes de cero.

3.4.4.1. Multiplicación binaria NAF

El método NAF (Non-Adjacent Form) forma no adyacente de un entero positivo k es una expresión de la forma $k = \sum_{i=0}^{l-1} k_i 2^i$ donde $k_i \in \{0, \pm 1\}, k_{l-1} \neq 0$ y al menos dos bits consecutivos k_i son dígitos diferentes de cero. Donde la longitud de la representación NAF es l.

Propiedades de la representación NAF:

- k tiene una única representación NAF denotada por $NAF(k)$
- $NAF(k)$ tiene menos dígitos diferentes de cero que la representación binaria de k
- La longitud de $NAF(k)$ es a lo más un bit mayor que la longitud de la representación binaria de k

- La densidad promedio de dígitos diferentes de cero de $NAF(k)$ es aproximadamente $1/3$.

El algoritmo 3.3 muestra como obtener la representación NAF.

Algoritmo 3.3 Algoritmo para obtener la representación NAF

Require: Entero positivo k.
Ensure: $NAF(k)$.

$i \leftarrow 0$
while $k \geq 1$ **do**
  if k es impar **then**
    $k_i \leftarrow 2 - (k \mod 4)$, $k \leftarrow k - k_i$
  else
    $k_i \leftarrow 0$
    $k \leftarrow k/2, i \leftarrow i + 1$
  end if
end while
Return $(k_{i-1}, k_{i-2}, \cdots, k_1, k_0)$

Una vez que se tiene la representación NAF(k) se puede emplear por ejemplo el algoritmo 3.4 de multiplicación de izquierda a derecha.

Algoritmo 3.4 Multiplicación binaria NAF de izquierda a derecha.

Require: $k, P \in E(\mathbb{F}_{2^m})$.
Ensure: kP.

Obtener la representación $NAF(k) = \sum_{i=0}^{l-1} k_i 2^i$
$Q \leftarrow P$
for $i = l - 2$ to 0 do **do**
  $Q \leftarrow 2Q$
  Si $k_i = 1$ entonces $Q \leftarrow Q + P$
  Si $k_i = -1$ entonces $Q \leftarrow Q - P$
end for
Return (Q)

El algoritmo 3.4 realiza en promedio $\frac{m-1}{3}$ sumas o restas y $m - 1$ doblados, esto es debido a que la densidad de dígitos diferentes de cero es ahora de dos ceros por cada dígito diferente cero. Ahora se tienen dos casos, para cuando se trata de 1 ó -1. Para el caso negativo, basta con obtener $-P = (x, x + y)$ y sumarlo al acumulador Q. Como el método va de izquierda a derecha, también se supone que el bit más significativo de la representación $NAF(k)$ es uno.

3.4.4.2. Multiplicación binaria wNAF

Mediante uso del método wNAF (window Non-Adjacent Form) se reduce aún más la densidad de dígitos diferentes de cero del escalar k, por lo que se decrementa el número de sumas requeridas para multiplicacion escalar.

Propiedades de la representación wNAF:

- k tiene una única representación wNAF denotada por $NAF_w(k)$
- $NAF_2(k) = NAF(k)$
- La longitud de $wNAF(k)$ es a lo más un dígito mayor que la longitud de la representación binaria de k
- La densidad promedio de dígitos diferentes de cero de $wNAF(k)$ es aproximadamente $1/(w+1)$.

El algoritmo 3.5 especifica como obtener la representación wNAF.

Algoritmo 3.5 Algoritmo para obtener la representación wNAF.

Require: Entero positivo k.
Ensure: $wNAF(k)$.
1: $i \leftarrow 0$
2: **while** $k \geq 1$ **do**
3: Si k es impar, entoces $k_i \leftarrow k$ mods 2^w, $k \leftarrow k - k_i$
4: Si no $k_i \leftarrow 0$
5: $k \leftarrow k/2, i \leftarrow i + 1$
6: **end while**
7: Return $(k_{i-1}, k_{i-2}, \cdots, k_1, k_0)$

A continuación se muestra el algoritmo de multiplicación de izquierda a derecha que utiliza la representación wNAF:

Este algoritmo 3.6 realiza en promedio [$\frac{m}{w+1}$ Sumas $+ m$ Doblados], y para el precómputo de los P_i requiere [1 Doblado $+(2^{w-2} - 1)$ Sumas].

Algoritmo 3.6 Multiplicación wNAF de izquierda a derecha:.

Require: ancho w, entero positivo $k, P \in E(\mathbb{F}_{2^m})$
Ensure: kP.

1: Obtener la representación $wNAF(k) = \sum_{i=0}^{l-1} k_i 2^i$
2: Precalcular $P_i = iP$ para $i \in \{1, 3, 5, \cdots, 2^{w-1} - 1\}$
3: $Q \leftarrow \mathcal{O}$
4: **for** $i = l - 1$ hasta 0 hacer **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad$ **if** $k_i \neq 0$ **then**
7: $\quad\quad$ **if** $k_i > 0$ **then**
8: $\quad\quad\quad Q \leftarrow Q + P_{k_i}$
9: $\quad\quad$ **else**
10: $\quad\quad\quad Q \leftarrow Q - P_{-k_i}$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: **end for**
14: Regresar (Q)

3.4.4.3. Multiplicación utilizando Bisección de punto

El método de multiplicación escalar utilizando bisección de punto, reemplaza las operaciones de doblado por la bisección, ya que esta última resulta ser una operación potencialmente más rápida.

Para poder hacer uso de bisecciones en lugar de doblados, se debe recodificar k,ya que el escalar en su expansión binaria está expresado en potencias positivas de 2 como se muestra a continuación.

$$k = k_{t-1}2^{t-1} + k_{t-2}2^{t-2} + ... + k_1 2^1 + k_0 2^0$$

Se requiere que el escalar quede expresado en potencias negativas de 2:

$$k' = k'_{t-1}2^0 + k'_{t-2}2^{-1} + ... + k'_1 2^{-(t-2)} + k'_0 2^{-(t-1)}$$

Para lograr esto, se efectúa la siguiente operación modulo n, donde n es el orden del punto generador.

$$k' = 2^{t-1}k \mod n$$

Así que se obtiene un nuevo escalar $k^{'}$ que es con el que se opera en la multiplicación usando bisecciones.

$$k^{'} = k^{'}_{t-1}2^{t-1} + k^{'}_{t-2}2^{t-2} + ... + k^{'}_{1}2^{1} + k^{'}_{0}2^{0}$$

Para poder recuperar el escalar k original, bastaría con dividir por el factor 2^{t-1} por el cual se ha multiplicado anteriormente.

$$k = \frac{k^{'}}{2^{t-1}} \text{ mod } n$$

Si se expresa el escalar $k^{'}$ en su forma binaria y se divide cada uno de los términos de la expansión por 2^{t-1}, entonces $k^{'}$ queda de la siguiente manera:

$$k = \frac{k^{'}_{t-1}2^{t-1}}{2^{t-1}} + \frac{k^{'}_{t-2}2^{t-2}}{2^{t-1}} + ... + \frac{k^{'}_{1}2^{1}}{2^{t-1}} + \frac{k^{'}_{0}2^{0}}{2^{t-1}}$$

Es decir,

$$k = k^{'}_{t-1}2^{0} + k^{'}_{t-2}2^{-1} + ... + k^{'}_{1}2^{-(t-2)} + k^{'}_{0}2^{-(t-1)}$$

Sólo que en lugar de efectuar la división, se realizan las bisecciones. De manera análoga al método de doblado y suma, se van recorriendo los bits del escalar $k^{'}$ de tal manera que para el bit más significativo de $k^{'}_{t-1}2^{0}$ no se bisecta y en caso de que esté encendido solo se acumula el valor del punto P, para el siguiente bit $k^{'}_{t-2}2^{-1}$ se realiza la bisección de P y en caso de que el bit $k^{'}_{t-2}$ esté encendido se acumula el resultado de la bisección, se continúa hasta que finalmente para el bit menos significativo $k^{'}_{0}2^{-(t-1)}$ se habrán efectuado $(t-1)$ bisecciones.

El algoritmo 3.7 muestra el método de bisección y suma para realizar la multiplicación escalar donde se reemplazan los doblados por bisecciones.

Este algoritmo 3.7 realiza en promedio $\frac{m^{'}-1}{2}$ sumas y $m^{'}-1$ bisecciones. Como este método va de izquierda a derecha y se supone el bit más significativo del escalar $k^{'}$ es uno, se inicializa el acumulador Q con P y se comienza a iterar desde $t-2$ hasta 0. Cabe señalar que en este caso a diferencia del método de "Doblado y Suma"de izquierda a derecha donde se dobla a Q, para este

Algoritmo 3.7 Multiplicación Bisección-y-Suma de izquierda a derecha.

Require: $(k^{'} = k^{'}_{t-1}2^0 + k^{'}_{t-2}2^{-1} + ... + k^{'}_1 2^{-(t-2)} + k^{'}_0 2^{-(t-1)}), P \in E(\mathbb{F}_{2^m})$
Ensure: kP.
1: $Q \leftarrow P$
2: **for** $i = t-2$ to 0 **do**
3: $P \leftarrow P/2$
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: **end for**
8: Regresar (Q)

Algoritmo 3.8 Multiplicación Suma-y-Bisección de derecha a izquierda

Require: $(k^{'} = k^{'}_{t-1}2^0 + k^{'}_{t-2}2^{-1} + ... + k^{'}_1 2^{-(t-2)} + k^{'}_0 2^{-(t-1)}), P \in E(\mathbb{F}_{2^m})$
Ensure: kP.
1: $Q \leftarrow \mathcal{O}$
2: **for** $i = 0$ to $t-2$ **do**
3: **if** $k_i = 1$ **then**
4: $Q \leftarrow Q + P$
5: **end if**
6: $Q \leftarrow Q/2$
7: **end for**
8: $Q \leftarrow Q + P$
9: Regresar (Q)

caso a quien se bisecta es a P y Q solo va acumulando la suma.

El algoritmo 3.8 realiza en promedio $\frac{m}{2}$ sumas y $m-1$ bisecciones. Como este método va de derecha a izquierda y suponiendo que el bit más significativo de $k^{'}$, $k^{'}_{t-1} = 1$ se debe iterar desde 0 hasta $t-2$.

Como puede verse en los algoritmos 3.7 y 3.8, de manera análoga a los métodos de doblado, se bisecta incondicionalmente y se realiza la suma en caso de que el bit $k^{'}_i$ se encuentre encendido. Los métodos NAF y wNAF que recodifican el escalar, en este caso $k^{'}$, también pueden aplicarse para reducir la densidad de dígitos diferentes de cero de $k^{'}$ y así decrementar el número de sumas.

Como puede verse en el algoritmo 3.9, las bisecciones $P \leftarrow P/2$ son realizadas en el punto P y se debe tener en cuenta que la bisección de punto recibe como parametros el punto en coordenadas

Algoritmo 3.9 Multiplicación wNAF utilizando Bisección de punto (derecha a izquierda).

Require: Ancho w, $k^{'} = 2^{t-1}k \mod n$, $P \in E(\mathbb{F}_{2^m})$
Ensure: kP.

1: Obtener la representación $wNAF(k^{'}) = \sum_{i=0}^{l-1} k_i^{'} 2^i$
2: Inicializar $Q_i \leftarrow \mathcal{O}$ para $i \in \{, 3, 5, \cdots, 2^{w-1} - 1\}$
3: Si $k_t^{'} = 1$ entonces $Q_1 \leftarrow 2P$
4: **for** $i = t - 1$ hasta 0 hacer **do**
5: **if** $k_i^{'} \neq 0$ **then**
6: **if** $k_i^{'} > 0$ **then**
7: $Q_{k_i^{'}} \leftarrow Q_{k_i^{'}} + P$
8: **else**
9: $Q_{-k_i^{'}} \leftarrow Q_{-k_i^{'}} - P$
10: **end if**
11: **end if**
12: $P \leftarrow P/2$
13: **end for**
14: $Q \leftarrow \sum_{i \in I} iQ_i$
15: Regresar (Q)

afines o en notación lambda. De tal forma que las sumas que se realizan de P con los acumuladores $Q_{k_i^{'}}$ deberán efectuarse en coordenadas mixtas. En caso de que el dígito $k_i^{'}$ sea negativo, se debe obtener $-P$ y se realiza la respectiva suma con el acumulador $Q_{-k_i^{'}}$.

El algoritmo 3.9 realiza t bisecciones de punto, un promedio de $t/(w+1) - 2^{w-2}$ sumas en coordenadas mixtas (López-Dahab), 2^{w-2} sumas del postcomputo y finalmente $2^{w-1} - 1$ sumas de la sumatoria final.

A continuación se muestra el algoritmo 3.10 de multiplicación escalar de izquierda a derecha que utiliza bisección en lugar de doblados de punto y donde se utiliza recodificación wNAF.

Como puede apreciarse en el algoritmo 3.10, las bisecciones ahora se efectúan en el acumulador Q por lo que Q no puede estar en coordenadas proyectivas, ya que si no habría que estar convirtiendo a coordenadas afines o notación lambda para poder efectuar las bisecciones. En este caso se debe efectuar precómputo de los $P_{k_i^{'}}$ y de los $P_{-k_i^{'}}$, a diferencia del algoritmo de derecha a izquierda donde se efectúa postcomputo.

Algoritmo 3.10 Multiplicación wNAF utilizando Bisección de punto (izquierda a derecha).

Require: Ancho w, $k^{'} = 2^{t-1}k \mod n$, $P \in E(\mathbb{F}_{2^m})$
Ensure: kP.

1: Obtener la representación $wNAF(k^{'}) = \sum_{i=0}^{l-1} k_i^{'} 2^i$
2: Precalcular $P_i = iP$ para $i \in \{1, 3, 5, \cdots, 2^{w-1} - 1\}$
3: $Q \leftarrow \mathcal{O}$
4: **for** $i = 0$ hasta $t - 1$ hacer **do**
5: $\quad Q \leftarrow Q/2$
6: $\quad$ **if** $k_i^{'} \neq 0$ **then**
7: $\quad\quad$ **if** $k_i^{'} > 0$ **then**
8: $\quad\quad\quad Q \leftarrow Q + P_{k_i^{'}}$
9: $\quad\quad$ **else**
10: $\quad\quad\quad Q \leftarrow Q - P_{-k_i^{'}}$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: **end for**
14: **if** $k_t^{'} = 1$ **then**
15: $\quad Q \leftarrow Q + 2P$
16: **end if**
17: Regresar (Q)

El algoritmo 3.10 realiza t bisecciones de punto, un promedio de $t/(w+1)$ sumas de punto. Para el precómputo, un doblado y $2^{w-2} - 1$ sumas.

4

Tecnologías Multinúcleo

Existe una creciente demanda por computadoras de menor tamaño, mayor capacidad de almacenamiento, portátiles más ligeras y servidores de aplicaciones más compactos, que impulsa el desarrollo de nuevas tecnologías. Los procesadores multinúcleo han surgido como una opción para resolver algunos desafíos de computación al no poder crecer más en velocidad, como se venía haciendo.

Proveedores de hardware y microprocesadores como por ejemplo, Intel, IBM, Sun y AMD han cambiado la estrategia de ser el primero en manejar velocidades mayores a 10 Gigahertz y han optado por lograr tener un mayor número de núcleos dentro de un mismo procesador [2].

4.1 Arquitecturas Multinúcleo

El hecho de que los principales proveedores de microprocesadores han cambiado la estrategía en el diseño de tecnología con mayor rendimiento, presenta nuevas oportunidades para los desarolladores de software. Las plataformas precedentes se basan en un modelo de programación secuencial. Los sistemas operativos y otras aplicaciones simulan ambientes multitarea tomando ventaja de la velocidad del procesador para poder atender varias tareas, asignando determinado tiempo de procesador a cada

una de ellas. Como resultado la tecnología multi-hilos dio buen resultado para poder aprovechar los tiempos de espera del procesador cuando, por ejemplo, ejecutaba operaciones de lectura y escritura a memoria.

Con la nueva arquitectura multi-núcleo se cuenta ahora con plataformas realmente paralelas en un mismo procesador, de tal manera que los desarrolladores de software cuentan con mayores capacidades y mayor versatilidad en el desarrollo y diseño de aplicaciones. De igual forma este cambio tecnológico presenta nuevos retos en el desarrollo de aplicaciones, ya que se deben tomar en cuenta ciertas consideraciones y afrontar nuevos problemas cuando se pretenden desarrollar aplicaciones paralalelas.

4.2 Arquitecturas de microprocesadores

4.2.1 Arquitectura de un solo núcle

Existen diferentes arquitecturas de microprocesadores, en la figura 4.1 se muestra la arquitectura de un solo núcleo que se utilizaba comúnmente. En esta arquitectura se cuenta con una sola unidad de ejecución, la memoria cache y el estado del CPU. En este tipo de procesadores las instrucciones se ejecutan una a la vez.

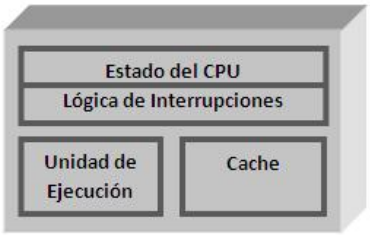

Figura 4.1: Modelo de la arquitectura de un solo núcleo

4.2.2 Arquitectura multiprocesador

Dada la demanda por equipos de mayor capacidad de cómputo, surge la arquitectura multiprocesador. Este tipo de tecnología puede ejecutar múltiples instrucciones en un mismo ciclo de reloj. La capacidad de ejecución depende del número de procesadores con los que se cuente, ya que cada uno de los procesadores al ser independientes tienen su propia memoria cache, lógica de interrupciones y unidad de ejecución.

La desventaja que presenta esta tecnología es obviamente el alto costo, ya que se cuenta con varios procesadores en la misma placa base. En la figura 4.2 se muestra un ejemplo de la arquitectura multiprocesador donde se aprecia que los recursos del microprocesador se encuentran duplicados, cabe señalar que los buses, memoria RAM y dispositivos periféricos se comparten entre los diferentes procesadores.

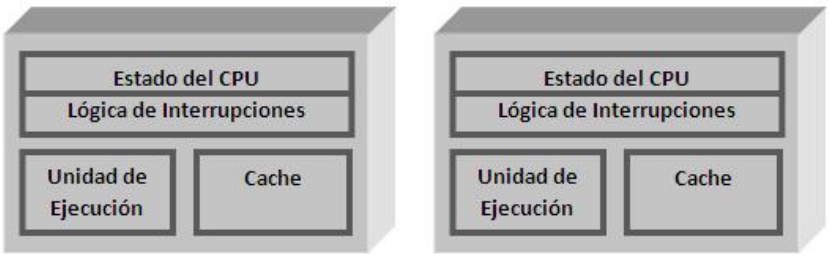

Figura 4.2: Modelo de la arquitectura multiprocesador.

4.2.3 Arquitectura Hyper-Threading

En la tecnología Hyper-Threading figura 4.3, algunas partes del procesador son compartidas, mientras que otras se encuentran duplicadas. Una de las partes más importantes que se encuentran compartidas es la unidad de ejecución, esto implica que un solo hilo será atendido mientras que el otro espera a que se le asigne la unidad de ejecución. Esto permite que se aprovechen mejor los recursos, ya que se evita el tiempo de ocio que se presenta cuando algún proceso tiene que esperar respuesta de memoria o algún dispositivo de entrada/salida.

La ganancia en rendimiento obtenida mediante el uso de esta tecnología varía dependiendo de las aplicaciones y de la plataforma de hardware, pero en promedio se obtiene una ganancia del 30 por ciento. Esto se debe a que entre mayor latencia existe en las aplicaciones, mayor será la ganancia mediante el empleo de la tecnología Hyper-Threading.

Figura 4.3: Modelo de la arquitectura Hyper-Threading.

4.2.4 Arquitectura multinúcleo

La tecnología multinúcleo, figura 4.4, cuenta con dos o más unidades de ejecución en el mismo procesador. En este tipo de arquitectura, n procesos se pueden ejecutar simultaneamente, donde n es el número de núcleos contenidos dentro de un mismo procesador. A diferencia de la tecnología Hyper-Threading donde se simulan varios procesadores, en este caso realmente se cuenta con capacidad de ejecución paralela.

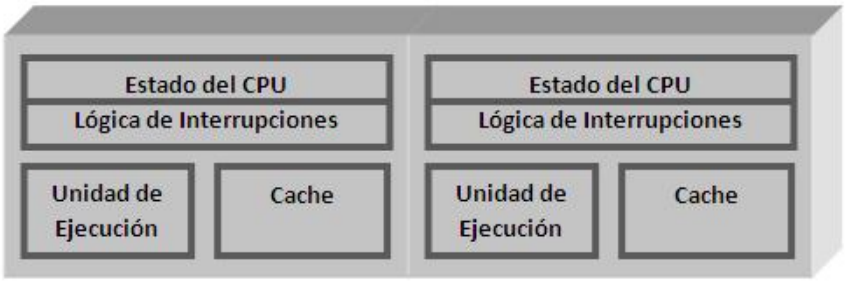

Figura 4.4: Modelo de la arquitectura multinúcleo.

Existe otro tipo de arquitectura multinúcleo donde se comparte la memoria cache, figura 4.5, este tipo de tecnología presenta ventajas en cuanto al costo pero tiene un menor rendimiento que la arquitectura con memorias cache independientes.

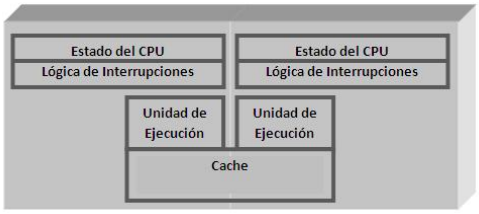

Figura 4.5: Modelo de la arquitectura multinúcleo con cache compartida.

4.2.5 Arquitectura multinúcleo con tecnología Hyper-Threading

Los nuevos procesadores que combinan la arquitectura multinúcleo con la tecnología Hyper-Threading, figura 4.6, permiten utilizar las ventajas de la ejecución simultanea de instrucciones y aprovechan también la latencia de los procesos cuando deben esperar respuesta de algún dispositivo de baja velocidad dentro de la placa base o dispositivo periférico.

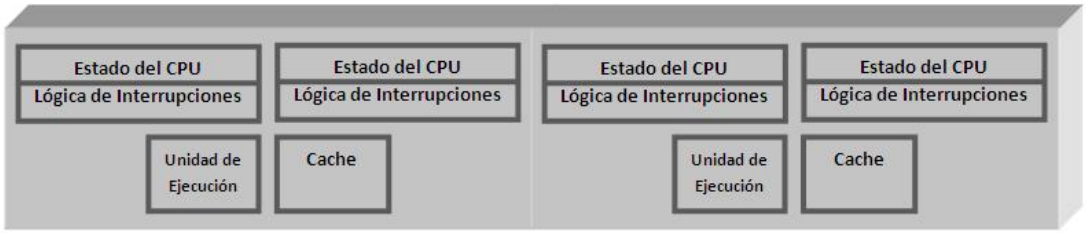

Figura 4.6: Modelo de la arquitectura multinúcleo con tecnología Hyper-Threading.

4.3 Consideraciones para la programación en arquitecturas multinúcleo

4.3.1 Paradigmas de programación paralela

Para poder pasar de un modelo lineal de programación a un modelo paralelo, se debe descomponer el problema en las distintas tareas que lo integran e identificar las dependencias que existen entre ellas. Es necesario identificar las tareas que son independientes y que potencialmente pueden ser ejecutadas de manera simultanea. Existen varios estilos para la construcción de programas paralelos. Se puede llevar a cabo una paralelización por datos, por tareas o por flujo de datos.

4.3.1.1. Paralelismo en datos

En este tipo de paralelismo, se descompone el problema dividiendo los datos a ser procesados en varios subconjuntos que son destinados a cada uno de los subprocesos. Cada hilo realiza las mismas operaciones pero con un subconjunto distinto de datos, de tal manera que en conjunto se encargan de procesar la totalidad de los datos. En otras palabras, se divide el trabajo a realizar entre los n procesos o hilos.

4.3.1.2. Paralelismo por tareas

Este es el método más sencillo de implementar debido a que se paralelizan las tareas que son independientes entre sí y que procesan sus propios datos, por lo que el nivel de sincronización y de dependencia entre los hilos es mínimo.

4.3.1.3. Paralelismo por flujo de control

En este tipo de paralelismo se tienen dependencias entre los distintos subprocesos, por lo que se debe manejar muy bien la sincronía y los tiempos de latencia. Es común que en ocasiones los datos deban ser preprocesados para luego poder ser divididos y asignados a distintas instancias de código

o que se tenga que esperar a que todos los subprocesos hayan terminado determinada etapa para poder recombinar los resultados y pasar a una siguiente parte del proceso o terminar.

4.3.2 Consideraciones fundamentales en cómputo paralelo

Existe una relación muy estrecha entre el diseño de un programa paralelo y la arquitectura de la computadora paralela donde se ejecutará. También es importante conocer muy bien el problema e identificar como poder descomponerlo en las diferentes tareas y partes independientes para determinar cuales pueden ser ejecutadas de manera simultánea.

4.3.2.1. Sincronización

En un ambiente donde se tienen varios subprocesos ejecutándose en paralelo, se debe tener un buen control en el orden de ejecución, acceso a los datos compartidos y a los tiempos de espera de los procesos que tienen dependencias unos con otros. Se debe tratar de balancear los tiempos de ejecución de los hilos que se ejecutan simultaneamente. Existen dos tipos de sincronización, por exclusión mutua y sincronización por condición. La exclusión mutua sirve para evitar el uso simultáneo de recursos comunes como variables globales o fragmentos de código conocidos como secciones críticas. En la sincronización por condición, los hilos deben esperar hasta que determinada condición se cumple y entonces pueden continuar.

4.3.2.2. Secciones críticas

Se denomina sección crítica al fragmento de código de un programa en la cual se accede a un recurso compartido que no debe ser utilizado por más de un subproceso en ejecución. La sección crítica por lo general es liberada en un tiempo determinado y el hilo o hilos deben esperar cierto tiempo para entrar, por lo que se necesita un mecanismo de sincronización, por ejemplo un semáforo, en la entrada y en la salida de la sección crítica para asegurar el uso exclusivo del recurso.

4.3.2.3. Bloqueo mutuo

El bloqueo mutuo que también es conocido como: interbloqueo, traba mortal, deadlock o abrazo mortal, ocurre cuando un hilo o subproceso queda a la espera de un recurso que nunca llega. Es decir, es el bloqueo permanente de uno o un conjunto de subprocesos en ejecución que esperan la disponibilidad de algún recurso del sistema, o bien alguna respuesta en la comunicación entre ellos. A diferencia de otros problemas de concurrencia de procesos, en este caso se debe tener especial cuidado en el diseño; ya que no existe una solución general para los interbloqueos.

4.4 Cálculo de la aceleración obtenida con el paralelismo

La ventaja obtenida al paralelizar un algoritmo o una aplicación se conoce como la aceleración, y esta define que tan rápido es un algoritmo paralelo respecto a la mejor versión secuencial del mismo.

$$\text{Aceleración} = \frac{Tiempo_{mejor\_algoritmo\_secuencial}}{Tiempo_{algoritmo\_paralelo}(n_t)}$$

La aceleración se determina dependiendo el número de hilos (n_t) utilizados en la implementación del algoritmo paralelo.

4.4.1 Ley de Amdahl

Comunmente cuando se paraleliza un algoritmo o aplicación, existen porciones de código que son inherentemente secuenciales y otras que sí son paralelizables. La ley de Amdahl establece que la aceleración estará determinada por la fracción del código que es mejorada y por que tanto es acelerada dicha fracción.

$$\text{A} = \frac{1}{(1-F_m)+(\frac{F_m}{A_m})}$$

donde:

- A es la aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas.
- A_m es el factor de mejora que se ha introducido en el subsistema mejorado.

- F_m es la fracción de tiempo que el sistema utiliza el subsistema mejorado.

Por ejemplo:

Si en un programa el tiempo de ejecución de un cierto algoritmo A utiliza un 40 % del tiempo total de ejecución del programa, y se logra hacer que este algoritmo A se ejecute en la mitad de tiempo, se tendría que:

- $A_m = 2$
- $F_m = 0.4$

$$\text{A} = \frac{1}{(1-0.4)+(\frac{0.4}{2})}$$

Es decir, se habría mejorado la velocidad de ejecución del programa en un factor de 1.25

La ley de Amdahl se mide en unidades adimensionales. Cuando se paraleliza un algoritmo, finalmente se llega a cierto límite donde no se puede paralelizar más, debido a características del propio algoritmo y no a la disponibilidad de un mayor número de hilos o núcleos.

4.4.2 Ley de Gustafson

La ley de Gustafson, también conocida como la ley de Gustafson-Barsis considera que la cantidad de paralelismo se escala con el tamaño de los datos (vectores o matrices) de la aplicación. Esta ley es una función alternativa a la ley de Amdahl para el cálculo de la aceleración, debido a que la ley de Amdahl presenta algunas consideraciones que no son del todo apropiadas para aplicaciones del mundo real:

- La ley de Amdahl supone que se toma el mejor algoritmo serial y que está limitado por las características del CPU. En una arquitectura multi-núcleo con cache independiente para cada núcleo, se podrían separar los datos y manejarse independientemente en cada cache por lo que se reduciría el tiempo de latencia de acceso a la memoria.

- La ley de Amdahl supone que se toma el mejor algoritmo secuencial, pero existen algunos programas que por naturaleza son más eficientes en su versión paralela.
- La ley de Amdahl modela adecuadamente muchos programa de flujo de control pero no modela adecuadamente programas del modelo paralelo en datos dado que no toma en cuenta el tamaño del problema.

La ley de Gustafson determina el límite en la aceleración que puede obtenerse cuando se paraleliza un programa, y plantea que un programa suficientemente grande puede ser paralelizado de manera eficiente y obtener una aceleración lineal con el número de procesadores o núcleos:

$$A = N + (1 - N) * s$$

donde:

- N es el número de procesadores o núcleos
- s es la relación del tiempo secuencial y el tiempo total de ejecución
- A es la aceleración obtenida

En este caso se puede ver que la aceleración es resulta ser lineal. Esta ley es equivalente a la ley de Amdahl pero con la diferencia que es más realista para el caso del cómputo paralelo en procesadores multi-núcleo.

5

Formulaciones paralelas de la multiplicación escalar en curvas binarias

En este capítulo se presenta la formulación paralela de la multiplicación escalar para una arquitectura multinúcleo. La implementación paralela se efectuó con el objetivo de hacer más eficiente la multiplicación escalar al aprovechar las características de multiprocesamiento que ofrecen los nuevos procesadores multinúcleo.

Finalmente se presentan los resultados obtenidos y se comparan contra los tiempos obtenidos por otras implementaciones de la multiplicación escalar en curvas elípticas.

5.1 Paralelización híbrida con doblado de punto y bisección de puntos

En el método de Doblado y Suma, se dobla incondicionalmente y se efectúa la suma dependiendo si el bit k_i está encendido. Esto es debido a que el escalar k se ha expandido en su forma binaria:

$$k = k_{t-1}2^{t-1} + k_{t-2}2^{t-2} + ... + k_1 2^1 + k_0 2^0$$

Cuando se reemplazan los doblados por bisecciones, se debe recodificar k de tal manera que quede expresado de manera binaria pero con potencias negativas:

$$k' = k'_{t-1}2^0 + k'_{t-2}2^{-1} + ... + k'_1 2^{-(t-2)} + k'_0 2^{-(t-1)}$$

para lograr esto, el escalar k se multiplica por un factor 2^{t-1} donde t es el número de bits de k.

$$k' = 2^{t-1}k \text{ mod } n$$

Finalmente se obtiene el módulo n, donde n es el orden del punto generador. De tal manera que el escalar k' queda expresado nuevamente en su expansión binaria y con potencias positivas pero al efectuar las $(t-1)$ bisecciones se estaría realizando la división por el factor 2^{t-1}.

$$k = \frac{k'}{2^{t-1}} \text{ mod } n$$

De esta manera el resultado de la multiplicación utilizando bisecciones en lugar de doblados, es equivalente a la multiplicación que utiliza Doblado y Suma.

Cuando se pretende utilizar un método de multiplicación que utilice simultaneamente la Bisección y el Doblado de punto, se debe recodificar el escalar k de tal manera que una parte quede expresada en potencias negativas y otra en potencias positivas. Así el método de multiplicación con bisecciones puede procesar los bits con potencias negativas del escalar k', donde k' es la recodificación de k, y el método de multiplicación con doblados procesa los bits con potencias positivas.

En la siguiente sección se describe como realizar la recodificación para poder utilizar simultaneamente bisecciones y doblados.

5.1.1 Recodificación del escalar k para la versión paralela con Bisección y Doblado

Para poder obtener una k expresada en su expansión binaria y que tenga potencias de 2 positivas y negativas, básicamente lo que se hace es que se reemplaza el factor 2^{t-1} por 2^I, donde:

$$I = INDEX = \frac{t}{d} - 1$$

t es el número de bits del escalar k.

d es un número en el rango (1,m).

Por ejemplo si $d = 1$, entonces

$$I = \frac{t}{d} - 1 = \frac{t}{1} - 1 = (t - 1)$$

por lo que al multiplicar por el factor $2^I = 2^{(t-1)}$, se estaría realizando la multiplicación únicamente con bisecciones, ya que k' quedaría representada con potencias negativas de 2.

$$k' = k'_{t-1}2^0 + k'_{t-2}2^{-1} + ... + k'_1 2^{-(t-2)} + k'_0 2^{-(t-1)}$$

Ahora, suponiendo que $d = 2$, entonces

$$I = \frac{t}{d} - 1 = \frac{t}{2} - 1$$

En este caso se tendría una k' codificada con la mitad de los bits en potencias positivas y la otra mitad con potencias negativas.

$$k' = k'_{t-1}2^{I-1} + k'_{t-2}2^{I-2} + ... + k_{I+1}2^1 + k_I 2^0 + ... + k'_1 2^{-(I-1)} + k'_0 2^{-(I)}$$

Suponiendo que t tiene m bits y para el caso que $d = m$, se tendría que:

$$I = \frac{t}{d} - 1 = \frac{m}{m} - 1 = 0$$

Al realizar la multiplicación por el factor $2^I = 2^0 = 1$, únicamente se estarían realizando doblados, ya que al no multiplicar por ningún factor el escalar k' queda representado sólo con potencias positivas.

5.1.2 Multiplicación usando Bisección de punto y Doblado de punto

El algoritmo de multiplicación 5.1 utiliza simultáneamente la Bisección y Doblado de punto.

Algoritmo 5.1 Multiplicación wNAF utilizando Bisección y Doblado de punto.

Require: Ancho w, $INDEX = t/d - 1$, $NAF_w(2^{INDEX}k \bmod n), P \in E(\mathbb{F}_{2^m})$
Ensure: kP.
1: Obtener la representación $wNAF(k') = \sum_{i=I-1}^{t-1} k'_i 2^i + \sum_{i=0}^{I} k'_i 2^{-i}$
2: Inicializar $Q_i \leftarrow \mathcal{O}$ para $i \in \{1, 3, 5, \cdots, 2^{w-1} - 1\}$
3: **if** $k'_t = 1$ **then**
4: $Q_1 \leftarrow 2P$
5: **end if**
6: **for** $i = t - 1$ to $INDEX + 1$ **do**
7: **if** $k'_i > 0$ **then**
8: $Q_{k'_i} \leftarrow Q_{k'_i} + P$
9: **end if**
10: **if** $k'_i < 0$ **then**
11: $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
12: **end if**
13: $P_D \leftarrow 2P_D$
14: **end for**
15: **for** $i = INDEX$ to 0 **do**
16: **if** $k'_i > 0$ **then**
17: $Q_{k'_i} \leftarrow Q_{k'_i} + P$
18: **end if**
19: **if** $k'_i < 0$ **then**
20: $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
21: **end if**
22: $P \leftarrow P/2$
23: **end for**
24: $Q \leftarrow \sum_{i \in In} iQ_i$
25: Regresar (Q)

En el algoritmo 5.1, las bisecciones $P \leftarrow P/2$ son realizadas en el punto P y se utilizan coordenadas mixtas para efectuar las sumas de punto. Para el caso del módulo que utiliza doblados $P_D \leftarrow 2P_D$, debe notarse que se utiliza un acumulador distinto P_D, y que los doblados se hacen en coordenadas afines, de tal forma que las sumas que se realizan de P_D con los acumuladores $Q_{k'_i}$ deberán efectuarse en coordenadas mixtas. En caso de que el dígito k'_i sea negativo, en ambos casos

se debe obtener $-P$ y se realiza la respectiva suma con el acumulador $Q_{-k_i'}$.

Este algoritmo realiza t/d bisecciones de punto, un promedio de $t/(w+1) - 2^{w-2}$ sumas en coordenadas mixtas (López-Dahab), 2^{w-2} sumas del postcómputo y finalmente $2^{w-1} - 1$ sumas de la sumatoria final.

En este algoritmo 5.1 los bucles que procesan las multiplicaciones parciales utilizando bisecciones y doblados de punto son independientes, por lo que podrían generarse dos subprocesos que se ejecuten de manera paralela. En la suma final se deben integrar los resultados parciales de los dos subprocesos para obtener el resultado final.

Algoritmo Multiplicación wNAF utilizando Bisección y Doblado de punto.

Require: Ancho w, $INDEX = t/d - 1$, $NAF_w(2^{INDEX} k \bmod n), P \in E(\mathbf{F}_{2^m})$
Ensure: kP.

Obtener la representación $wNAF(k') = \sum_{i=l-1}^{t-1} k_i' 2^i + \sum_{i=0}^{l} k_i' 2^{-i}$
Inicializar $Q_i \leftarrow \mathcal{O}$ para $i \in \{1, 3, 5, \cdots, 2^{w-1} - 1\}$
if $k_t' = 1$ **then**
  $Q_1 \leftarrow 2P$
end if

for $i = t - 1$ hasta $INDEX + 1$ hacer **do**	**for** $i = INDEX$ hasta 0 hacer **do**
if $k_i' > 0$ **then**	**if** $k_i' > 0$ **then**
$Q_{k_i'} \leftarrow Q_{k_i'} + P$	$Q_{k_i'} \leftarrow Q_{k_i'} + P$
end if	**end if**
if $k_i' < 0$ **then**	**if** $k_i' < 0$ **then**
$Q_{-k_i'} \leftarrow Q_{-k_i'} - P$	$Q_{-k_i'} \leftarrow Q_{-k_i'} - P$
end if	**end if**
$P_D \leftarrow 2P_D$	$P \leftarrow P/2$
end for	**end for**

$Q \leftarrow \sum_{i \in In} iQ_i$
Regresar (Q)

Figura 5.1: Multiplicación escalar utilizando Bisección y Doblado de punto.

Es importante que la carga de trabajo para cada uno de los subprocesos se encuentre bien balanceada y que los dos hilos terminen aproximadamente al mismo tiempo para que se obtenga el menor tiempo de ejecución posible. De no ser así, uno de los procesos tendría que estar inactivo esperando a que el otro termine para finalmente conjuntar los resultados.

5.1.2.1. Ley de Amdahl

Como se vio en el capítulo anterior, la ley de Amdahl establece que la aceleración está determinada por la fracción del código que es mejorada y por que tanto es acelerada dicha fracción.

$$\mathrm{A} = \frac{1}{(1-F_m)+(\frac{F_m}{A_m})}$$

Para nuestro caso los bucles que calculan la multiplicación utilizan un 80 % del tiempo total de ejecución del programa, y dado que se ejecutan dos hilos, se logra hacer que este algoritmo se ejecute en aproximadamente la mitad de tiempo, por lo que se tiene que:

- $A_m = 2$
- $F_m = 0.7$

$$\mathrm{A} = \frac{1}{(1-0.7)+(\frac{0.7}{2})}$$

Es decir, se ha mejorado la velocidad de ejecución del programa en un factor de 1.53

5.1.2.2. Ley de Gustafson

La ley de Gustafson también determina el límite en la aceleración que puede obtenerse cuando se paraleliza un programa y plantea que un programa suficientemente grande puede ser paralelizado de manera eficiente y obtener una aceleración lineal con el número de procesadores o núcleos. Pero para nuestro caso aunque se contara con un mayor número de nucleos, como estamos en el caso de punto P desconocido, se debería de poder particionar el escalar k en un mayor número de subpartes para poder disponer de más núcleos.

Para el caso de punto desconocido, sí existe dependencia en los ciclos del algoritmo debido a que se debe doblar o bisectar incondicionalmente el punto P para poder calcular el siguiente ciclo.

Para el caso de punto P conocido, se podría tener el caso de contar con una tabla donde se tienen precalculados los m valores resultantes de los doblados o de las bisecciones figura 5.2.

En este caso solo se tendría que ir acumulando el resultado de las sumas de los puntos para los cuales el bit k_i del escalar k este encendido.

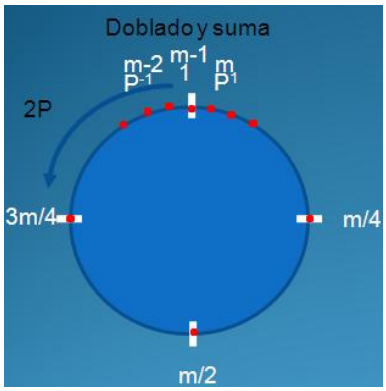

Figura 5.2: Doblado y suma para punto conocido.

Como puede verse en la figura 5.3, para el caso de punto P conocido aplican otras técnicas de paralelización como el truco de Shamir y utilizar también biseccion y suma.

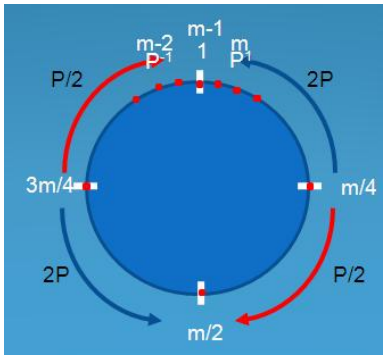

Figura 5.3: Particionamiento para punto conocido.

5.2 Resultados

Los tiempos que se obtuvieron en las diferentes versiones de multiplicación escalar se muestran en la siguiente tabla:

Método de Multiplicación:	Tiempo(ms)
Double-and-Add	4.1
Halve-and-Add	5.3
Mixed Halve and Double	3.3

Tabla 5.1: TABLA DE RESULTADOS

La implementación de los algoritmos fue realizada en lenguaje C, y se utilizó openMP para efectuar la programación multi-hilos. El procesador utilizado para obtener los tiempos fue un Pentium Dual Core a 1.73Ghz.

Como puede verse en la tabla de resultados, el tiempo de ejecución resultante en el algoritmo de Bisección y Suma es mayor a del algoritmo de Doblado y Suma, esto se debe a que en la implementación de la función de bisección de punto se obtuvo un tiempo ligeramente mayor que el doblado.

6

Conclusiones

En este trabajo de tesis se realizó un estudio y análisis del algoritmo de multiplicación en curvas elípticas binarias. Se revisaron las diferentes versiones que se emplean para calcular la multiplicación escalar kP, como por ejemplo la de suma y doblado o la que utiliza bisección y suma. Esto llevó a cabo con el objetivo el diseñar e implementar un algoritmo paralelo de la multiplicación escalar en curvas elípticas que utiliza simultaneamente la bisección de punto y el doblado de punto.

Para poder lograr el objetivo principal, se desarrollaron versiones como:

- Multiplicación escalar en curvas elípticas binarias utilizando doblado de punto.
- Multiplicación escalar en curvas elípticas binarias empleando bisección de punto.
- Multiplicación escalar en curvas elípticas binarias empleando simultaneamente la bisección de punto y el doblado de punto.

Para esta última versión se propuso una manera de recodificar el escalar k para que se pudiese emplear simultaneamente la bisección y el doblado de punto.

La actual tendencia tecnológica apunta a que en un futuro, incluso los dispositivos móviles dispondrán de varios nucleos con el objetivo de aumentar la capacidad de procesamiento, evitar el

sobrecalentamiento y reducir el consumo de potencia. Es por ello que resulta relevante el poder migrar algunos algoritmos a versiones que permitan aprovechar las capacidades de los nuevos procesadores.

Es importante también conocer las ventajas de las diferentes arquitecturas de hardware, debido a que el diseño de los algoritmos y la plataforma en la que se ejecutan guardan una estrecha relación que debe optimizarse para poder obtener un buen desempeño.

6.1 Trabajo futuro

- El algoritmo de multiplicación propuesto permite utilizar simultáneamente la Bisección y el Doblado de punto. Sin embargo, resulta necesario lograr particionar con una mayor granularidad la multiplicación escalar para poder aprovechar las ventajas de procesadores con un mayor número de núcleos.

- Como una posible estrategia de paralelización para un mayor número de núcleos, podría utilizarse el método de paralelización propuesto por B. Ansari y Huapeng Wu [3]. Este método sugiere utilizar un esquema productor consumidor para separar los doblados y las sumas, ya que se dobla incondicionalmente y la suma se realiza sólo cuando el bit k_i se encuentra encendido. De manera similar se podría separar la bisección de las sumas y seguir el esquema antes mencionado.

- Algunas de las funciones de la implementación realizada pueden ser mejoradas mediante otras estrategias de programanción para poder obtener mejores tiempos de ejecución.

Bibliografía

[1] O. Ahmadi, D. Hankerson, and F. Rodríguez-Henríquez. Parallel formulations of scalar multiplication on koblitz curves. *J.UCS: Journal of Universal Computer Science*, 14(3):481–504, 2008.

[2] S. Akhter and J. Roberts. *Multi-Core Programming: Increasing Performance through Software Multi-threading*. Intel Press, United States of America, 2006.

[3] B. Ansari and H. Wu. Parallel scalar multiplication for elliptic curve cryptosystems. *IEEE Trans. Computers*, 1:71 – 73, 2005.

[4] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[5] G. Mallén J. Vázquez E. Daltabuit, L. Hernández. *La seguridad de la información*. Editorial Limusa, Balderas 95, México, D.F., 2007.

[6] J. Lopez Fong, D. Hankerson and A. Menezes. Field inversion and point halving revisited. *IEEETC: IEEE Transactions on Computers*, 53, 2004.

[7] Oded Goldreich. Foundations of cryptography: a primer. *Found. Trends Theor. Comput. Sci.*, 1(1):1–116, 2005.

[8] I.Ñ. Herstein. *Algebra Moderna: grupos, anillos, campos, teoría de Galois*. Trillas, México, D.F., 2006.

[9] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[10] N. Koblitz. CM-curves with good cryptographic properties. *Lecture Notes in Computer Science*, 576:279–287, 1991.

[11] W.El hadj youssef B. Bouallegue A. Baganne M. Machhout, M. Zeghid and R.Tourki. Efficient large numbers karatsuba-ofman multiplier designs for embedded systems. *International Journal of Electronics, Circuits and Systems*, 3:123–133, 2009.

[12] A. Menezes, A. Scott Vanstone, and P. C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[13] V. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology—CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1986, 18–22 August 1985.

[14] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[15] M. J. B. Robshaw. Stream ciphers. rsa laboratories technical report tr-701. Technical Report 43, RSA Laboratories, http://islab.oregonstate.edu/koc/ece575/rsalabs/tr-701.pdf, May 1995.

[16] F. Rodríguez-Henríquez, G. Morales-Luna, and J. López. Low-complexity bit-parallel square root computation over $GF(2^m)$ for all trinomials. *IEEE Trans. Computers*, 57(4):472–480, 2008.

[17] J. A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In Burton S. Kaliski Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer-Verlag, 17–21 August 1997.