Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Zacatenco

Departamento de Ingeniería Eléctrica
Sección de Computación

Multiplicación Escalar en Curvas Elípticas empleando Bisección de Punto: una Arquitectura en Hardware Reconfigurable

Tesis que presenta
Hernández Rodríguez Sabel Mercurio

Para obtener el Grado de
Maestro en Ciencias

En la Especialidad de
Ingeniería Eléctrica

Opción
Computación

Director de la Tesis
Francisco Rodríguez Henríquez

Ciudad de México, Distrito Federal Enero del 2006

A mis padres y a mis hermanos.

Agradecimientos

Agradezco al CINVESTAV por permitirme realizar mis estudios de maestría. Agradezco al CONACYT por su apoyo económico brindado por medio del proyecto número 45306.

Quisiera agradecer a mi familia, en especial a mis padres por su apoyo tanto moral como económico que me brindaron durante el tiempo que estuve realizando la maestría. A mis hermanos por los ánimos que me brindaron para poder salir adelante con este trabajo.

Doy gracias a mi asesor de tesis el Dr. Francisco Rodríguez Henríquez, por su apoyo y su aporte de ideas para la realización de esta tesis. Así como el permitirme trabajar con él dentro del área de criptografía.

Agradezco a los revisores de mi tesis el Dr. Guillermo Morales Luna (CINESTAV-IPN) y al Dr. Arturo Díaz Pérez (CINVESTAV-IPN), por sus observaciones y consejos.

También agradezco al Dr. Julio López por sus sugerencias e ideas para poder mejorar el presente trabajo.

Finalmente agradezco a Sofía Reza y Flor Córdova por su orientación para resolver cuestiones académicas. También quisiera agradecer al personal de servicios académicos y al personal de biblioteca.

Resumen

La criptografía de llave pública que es empleada hoy en día en diversas aplicaciones (sobre todo de transferencia de datos) requiere de gran poder de cómputo. En el caso de Firma Digital con Curvas Elípticas (Elliptic Curve Digital Signature Algorithm-ECDSA) la operación más costosa es la multiplicación escalar, por lo que es necesario poder realizar este tipo de operaciones de manera rápida y eficiente. Knudsen y Schroeppel propusieron reemplazar el doblado de punto por la bisección de punto dentro del algoritmo general de la multiplicación escalar. La bisección de punto es una operación menos costosa que la inversión en la aritmética de campos finitos binarios; además de permitir en los casos generales de una curva elíptica reducir el número de operaciones totales a ejecutar en la multiplicación escalar [1, 2]. El presente trabajo tiene como meta proponer un diseño en hardware reconfigurable que emplee el algoritmo de suma y bisección de punto para realizar la multiplicación escalar de manera rápida y eficiente.

Abstract

Nowadays, security issues have become an important design issue in digital communications. Under this scenario, public key cryptography has emerged as a solution for solving this kind of problems. Indeed, the Elliptic Curve Digital Signature Algorithm (ECDSA) has become one of the most important schemes for performing signature/verification primitives to authenticate people. However this operation demands a great deal of computational resources, quite specially, scalar multiplication which is both, the most important and the most time-consuming ECDSA operation. That is why fast and efficient implementations of this operation are required. Knudsen and Schroeppel [1, 2] proposed a fast algorithm to perform scalar multiplication called half-and-add method. This algorithm requires to perform a new elliptic curve operation namely point halving. The present work proposes an architecture over a reconfigurable hardware platform that employs the half-and-add method in order to perform a fast an efficient scalar multiplication.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este capítulo pretende presentar una perspectiva general de la organización de la tesis. Aquí se presentan los conceptos básicos de la criptografía, así como las operaciones y algoritmos que son empleados comúnmente en implementaciones criptográficas. También se describen las diversas alternativas de implementaciones que se emplean para poder crear estas aplicaciones criptográficas. Además en este capítulo se presentan el objetivo general y los objetivos particulares alcanzados durante el desarrollo de la tesis, así como las contribuciones aportadas por este trabajo. Al final del capítulo se presenta la organización de la tesis.

1.1. Introducción

La red Internet está constituida por un conjunto de redes interconectadas. Esto supone que cada empresa, organismo o institución sea responsable o propietaria de la porción de su red que está conectada a Internet, es decir, su Intranet. En base a este hecho, se puede decir que Internet no es de nadie, y es de todos a la vez, pues cada quien resulta dueño de su propio ámbito local, por lo que cada quien deberá preocuparse de tener un buen canal de acceso al resto de Internet, y de configurar las políticas de seguridad de su sistema.

A diferencia de las tradicionales redes de telefonía, cuando dos usuarios (es decir, dos computadoras) se comunican utilizando Internet, no se establece una conexión física entre ellos, o sea, no se establece un canal de comunicaciones de uso exclusivo para estos usuarios. Por el contrario, la información que fluye entre ellos, dividida en unidades de datos denominadas paquetes o datagramas, pueden cruzar caminos diferentes mientras dura la comunicación, dependiendo de las condiciones de la red en cada momento. Dichos caminos están constituidos por unas secuencias de enlaces, enrutadores, o subredes de múltiples

propietarios. Es por tanto, imposible garantizar de manera general, que la información que se envía a Internet esté a salvo de ser interceptada por terceras personas.

El desarrollo de aplicaciones de envío y transferencia de datos ha creado una nueva necesidad: la seguridad con que se envía dicha información. Ya que el medio habitual para establecer este tipo de comunicaciones es la Internet. Este canal de comunicación es sin embargo un canal inseguro, ya que al tratarse de un canal público de transferencia de información; cualquier persona tiene acceso a este medio de comunicación.

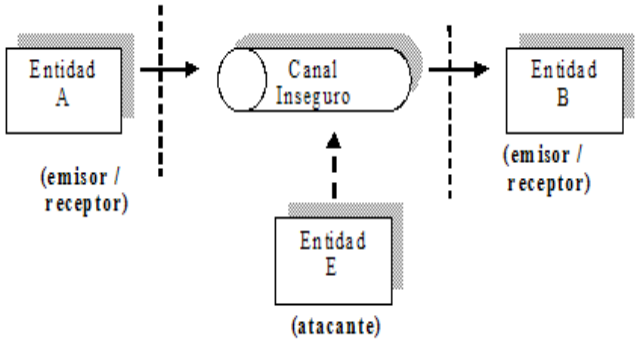

Figura 1.1: Modelo de comunicación básico.

La figura 1.1 ilustra el modelo de comunicación que se establece cuando dos entidades A, B quieren comunicarse entre sí. Se puede observar que el medio de comunicación es fatalmente inseguro y que existe la posibilidad que una tercera entidad tenga acceso a la información que se envía.

La criptografía es la herramienta que comúnmente se emplea para lograr la seguridad en dichas aplicaciones. Del griego kryptos (ocultar) y grafos (escribir), literalmente significa escritura oculta, la criptografía es la ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes van dirigidos.

En la criptografía, el termino *cifrar*, se emplea cuando se trata de codificar un mensaje de tal manera que la información original es escondida o modificada tal que a simple vista no pueda ser comprendida. El procedimiento inverso a cifrar se denomina *descifrar*, el cual es un proceso que recupera la información original a partir del texto cifrado. En criptografía clásica, el método de codificación que se usa para cifrar y descifrar un mensaje hace uso de una *llave*. Esta llave sirve tanto para cifrar como descifrar un mensaje.

La finalidad de la criptografía es pues, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.), y, en segundo lugar, asegurar

que la información que se envía es auténtica en un doble sentido: que el emisor sea realmente quien dice ser; y que el contenido del mensaje enviado (habitualmente denominado cifra) no haya sido modificado en su tránsito.

Dependiendo del tipo de actividad o del servicio que se desee ofrecer, habrá que garantizar la seguridad desde un punto de vista u otro. Entre otras cuestiones, deben implantarse medidas para garantizar que:

1. La información enviada no sea leída por un tercero (Servicio de Confidencialidad). Este servicio garantiza que los datos transmitidos por la red, si llegase a ser interceptados por un tercero, no puedan ser descifrados o interpretados, a menos que el oponente conozca la llave privada del destinatario.

2. La información enviada no sea manipulada por un tercero (Servicio de Integridad de los datos). Esto permite, por ejemplo saber si ocurre un error de transmisión, o si alguna persona maliciosa ha interceptado el mensaje y lo ha alterado antes de llegar a su destino.

3. El emisor sea quien dice ser (Servicio de Autenticación y Control de acceso). Este servicio tiene como objetivo que un usuario que envía un documento electrónico, pueda identificarse ante el receptor, teniendo éste garantía de que el emisor es el auténtico, y no un impostor. Así, cuando se realiza una transacción electrónica, una compra por Internet, un pago por tarjeta, debe garantizarse que el usuario que solicita el servicio es realmente quien dice ser y no un impostor.

4. El emisor pueda justificar que ha enviado la información o que el destinatario no pueda negar que no lo haya recibido (Servicio de No Repudio). Este servicio tiene importancia para el emisor, ya que se puede garantizar que el destinatario ha recibido el mensaje y no puede negarse a ello. El procedimiento es sencillo: cuando se envía un mensaje firmado digitalmente, puede exigir acuse de recibo, firmado digitalmente por el destinatario. Un acuse de recibo firmado y contrastado con la llave pública del otro usuario, garantiza que el mensaje ha sido recibido por el destinatario.

La criptografía se divide en dos grandes bloques: criptografía de llave privada (o criptografía simétrica), y en criptografía de llave pública (o criptografía de llave asimétrica). Sin embargo la criptografía de llave pública ha cobrado mayor fuerza en últimas fechas ya que se emplea ampliamente en aplicaciones tales como el comercio electrónico, transferencias bancarias, dinero digital, etc.

1.2. Antecedentes y motivación

La criptografía estudia la solución a los problemas de identificación, autenticación y privacidad en los sistemas basados en computadoras. Los problemas de protección de privacidad y autenticación en grandes redes fueron abordados teóricamente por Whitfield Diffie y Martin Hellman en 1976 cuando publicaron su concepto para un novedoso método de intercambio de mensajes secretos sin intercambio de llaves secretas. La idea se realiza en la práctica un año después en 1977 con la invención del Criptosistema de llave pública RSA, por Ronald Rivest, Adi Shamir y Len Adleman (en ese entonces profesores en el Massachusetts Institute of Technology). En 1985 Neal Koblitz y Víctor Miller propusieron independientemente; el Criptosistema de Curvas Elípticas (Elliptic Curve Cryptosystem-ECC), que también es un sistema basado en criptografía de llave pública. En 1991 el Instituto Nacional de Estándares y Tecnología (National Institute of Standars and Technology-NIST) propone el esquema de llave pública: Algoritmo de Firma Digital (Digital Signature Algorithm-DSA), el cual se convierte en el estándar de firma digital (DSS). Posteriormente el NIST propone en el año 2000 la segunda versión del DSS donde incluye tanto el DSA como el Algoritmo de Firma Digital con Curvas Elípticas (Elliptic Curve Digital Signature Algorithm-ECDSA).

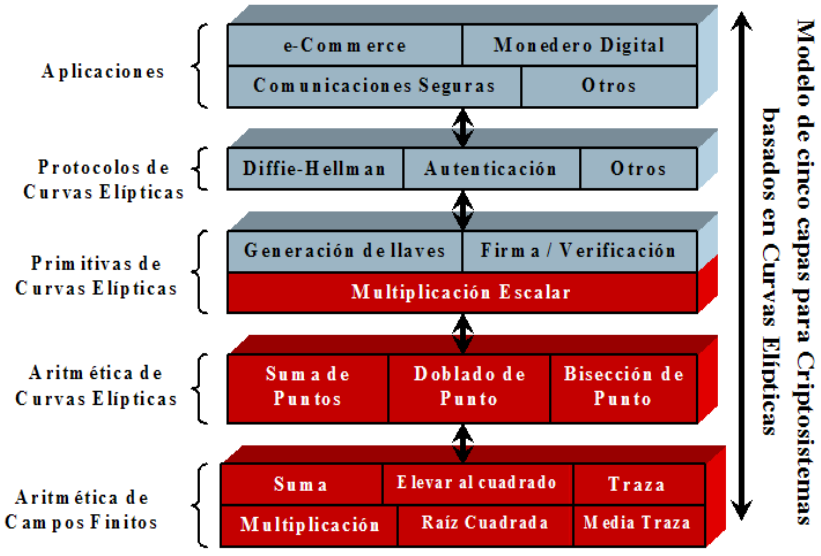

Figura 1.2: Modelo por capas de un Criptosistema de Curvas Elípticas.

La figura 1.2, permite observar cómo se encuentra compuesto una aplicación que utiliza

criptografía de llave pública con curvas elípticas. La capa superior representa la aplicación que se desarrolla, los dos siguientes niveles representan los protocolos de seguridad que se emplean dentro de la aplicación y las primitivas de la firma/verificación. Los siguientes niveles representan las primitivas de curvas elípticas, y las primitivas de la aritmética de campos finitos. Estas últimas capas representan la base para realizar un criptosistema de curvas elípticas. Es en las dos primeras capas, y parte de la tercera en lo que se enfoca el presente trabajo.

El empleo de criptografía de llave pública como medio de comunicación seguro, es cada vez más usado en todo tipo de comunicaciones a través de la red. Con el crecimiento acelerado de tecnologías móviles (celulares, PDA's, entre otros); se ha vuelto importante (sino es que indispensable) el empleo de transmisiones seguras en este tipo de medios.

La figura 1.2, es un modelo de capas que permite observar cómo está compuesta una aplicación que emplea criptografía de curvas elípticas. Es importante remarcar que las capas inferiores del modelo, componen la base de un sistema de firma/verificación. Es sobre estas capas que se realiza el mayor número de operaciones aritméticas de campo finito, y operaciones aritméticas de curvas elípticas.

La pregunta abordada en la tesis es: ¿es posible crear una arquitectura en hardware que permita efectuar estas operaciones de bajo nivel de forma rápida y eficiente? Al crear una arquitectura que efectúe estas operaciones de bajo nivel, es posible crear una capa superior (en software) que permita efectuar el esquema de firma/verificación de manera eficiente, puesto que al reducir los tiempos de ejecución de las capas inferiores del modelo de la figura 1.2, es posible acelerar el procesamiento de las capas superiores. Incluso sería posible crear un circuito integrado de propósito específico.

1.3. Objetivos

1.3.1. General

El presente trabajo tiene como finalidad proponer un diseño en hardware reconfigurable que permita realizar de manera rápida y eficiente la multiplicación escalar en curvas elípticas, empleando el método de bisección de punto.

1.3.2. Particulares

Se tienen como objetivos particulares del trabajo los siguientes puntos:

1. Realizar el análisis del diseño a implementar

2. Hacer una implementación del algoritmo de bisección de punto en VHDL, tres módulos principales:

   a) Calcular la traza de un elemento del campo.

   b) Obtener la raíz cuadrada de un elemento del campo.

   c) Resolver una ecuación cuadrática.

3. Realizar la implementación del método de suma y bisección en VHDL, que calcule la multiplicación escalar kP.

4. Realizar el análisis y comparación de resultados obtenidos con otros diseños (en FPGA) que realicen multiplicación escalar

1.4. Estado del arte

Desde la propuesta hecha por Whitfield Diffie y Martin Hellman en 1976 [5], de ocupar un criptosistema de llave pública para el intercambio de información de manera segura, se han realizado diversos trabajos e investigaciones para poner en práctica estos principios. En 1977 Ronald Rivest, Adi Shamir y Len Adleman implementan el primer criptosistema que ocupa el principio de llave pública usando el algoritmo de su creación, RSA [6]. Más tarde Koblitz [7] y Miller [8] proponen (de forma independiente) emplear curvas elípticas como base matemática para el diseño de criptosistemas de llave pública. A finales de los años '90, los sistemas de seguridad que emplean curvas elípticas comienzan a recibir una mayor aceptación comercial, por lo que son propuestos y aceptados estándares y protocolos que tienen como base las curvas elípticas [9, 10, 11, 12].

El estándar propuesto por el Instituto Nacional de Estándares y Tecnología [9] (National Institute of Standards and Technology-NIST), explica en detalle los procedimientos necesarios para realizar las operaciones de campos finitos primos y binarios, así como las operaciones de curvas elípticas que permiten realizar las operaciones de firma/verificación. Este estándar también contiene los algoritmos que se deben de emplear en la creación de un criptosistema de llave pública.

La criptografía de llave pública ha inspirado el desarrollo de algoritmos que permite realizar firma y verificación de documentos electrónicos de una forma eficiente (López, Dahab [13]), en implementaciones en software [14, 15, 16], en hardware [17, 18], así como hardware reconfigurable [19, 20, 21, 22, 23]. La importancia de la criptografía de llave pública ha tomando tanto auge, que en la actualidad se propone crear un procesador dedicado [18] que realice todas las operaciones básicas de curvas elípticas, y sobre esta arquitectura crear en software, las aplicaciones que se requieran para realizar dichas operaciones aritméticas.

En 1999 Knudsen [1] y Schroeppel [2], de forma independiente realizan un estudio sobre las operaciones que se pueden realizar en curvas elípticas, de tal forma que encontraron nuevas maneras de realizar la multiplicación escalar; en vez de emplear el método de suma y doblado de puntos se propone un nuevo método el de suma y bisección de puntos. En este nuevo método para calcular la multiplicación escalar, se reemplaza el doblado de punto por una operación que tiene menor costo computacional: la bisección de punto.

Fong, Hankerson, López y Menezes [24], realizaron comparaciones de este nueva propuesta con respecto al método de suma y doblado ("double-and-add"). El atractivo del método de suma y bisección ("half-and-add") sobre el método de suma y doblado se aprecia en el hecho que en la multiplicación escalar se tienen que realizar un mayor número de inversiones de campo. Un análisis hecho por Knudsen [1], indica que si el costo de una inversión es aproximadamente igual al de tres multiplicaciones de campo, el método de suma y bisección tiene una ventaja de 39 % sobre el método de suma y doblado. Sin embargo Fong, Hankerson, López y Menezes [24] mencionan que la ventaja proporcionada por Knudsen es demasiado optimista, puesto que el análisis que ellos realizaron sobre plataformas SPARC y Pentium indica que los mejores tiempos se obtienen si el costo de una inversión es inferior a ocho multiplicaciones de campo, i.e. el costo de multiplicaciones a realizar es menor al costo de una inversión; por lo que método de suma y bisección es mejor que el método de suma y doblado. Y la reducción en tiempo de ejecución es aproximadamente de un 25 %.

1.5. Organización de la tesis

La organización del resto de este documento es como sigue: en el capítulo §2 se da una introducción a la criptografía de llave pública, y se mencionan sus ventajas y desventajas con respecto de la criptografía de llave privada. El capítulo §3 incluye una introducción tanto a la aritmética de campos finitos como a la aritmética de curvas elípticas, la cual forma base de toda implementación de criptografía de curvas elípticas. Posteriormente en el capítulo §4 se describe la Unidad Aritmético-lógica (ALU) que realiza las operaciones básicas de aritmética de campos finitos. En seguida en el capítulo §5 se presentará la arquitectura final propuesta para calcular la multiplicación escalar. Finalmente en el capítulo §6 se presentarán los resultados y las conclusiones del trabajo, así como el trabajo a futuro.

Capítulo 2

Criptografía de Llave Pública

En este capítulo se da una introducción a los conceptos de la criptografía de llave pública. Al mismo tiempo que se muestran las ventajas que presenta con respecto al esquema de criptografía de llave privada.

Como se menciona en la introducción, los problemas de identificación, autenticidad y privacidad pueden ser resueltos con la ayuda de la criptografía. Estos problemas se acentúan sobre todo en las comunicaciones digitales que se realizan por un canal público (por ejemplo la Internet), por lo que hay que garantizar de alguna forma la seguridad de los datos que se envían. Hoy en día la criptografía de llave pública ha mostrado ser una manera eficiente de brindar seguridad a documentos digitales así como una técnica que permite realizar un intercambio de llaves en un medio público de una forma segura.

2.1. Seguridad en comunicaciones digitales

Desde tiempos remotos la criptografía de llave privada ha sido el método utilizado para enviar mensajes cifrados. Métodos de cifrado clásicos como el de substitución, de corrimiento, el método de Vigenère, entre otros, y los métodos modernos de cifrado por bloques (DES, AES) fueron y han sido empleados para el cifrado de datos. La criptografía de llave privada es un método criptográfico que usa la misma llave tanto para cifrar como para descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la llave o secreto compartido. Una vez que ambas partes tienen acceso a esta llave, el remitente cifra un mensaje usándola, lo envía al destinatario, y éste lo descifra con esa misma llave (ver figura 2.1).

Sin embargo, la criptografía de llave privada presenta dos grandes desventajas:

- Distribución de llaves. Es decir que el canal para poder distribuir las diferentes llaves

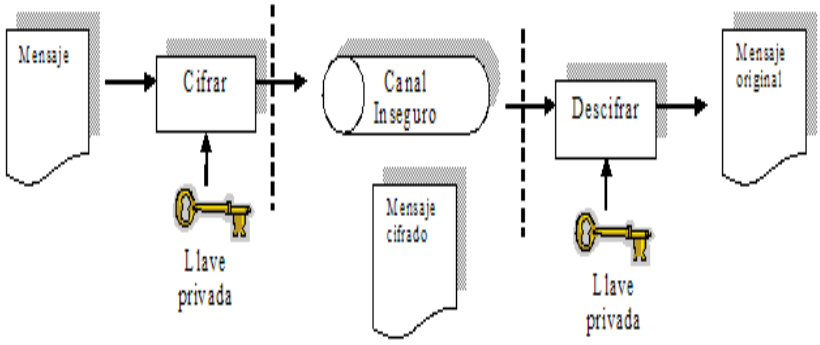

Figura 2.1: Modelo de criptografía de llave privada.

entre personas debe de ser seguro y confiable. En ciertos escenarios, la distribución de llaves se puede llevar a cabo por medio de un canal físico seguro, o se puede emplear un servidor seguro que se encargue de dicha distribución de llaves (por ejemplo un servidor que emplee el servicio de Kerberos). Este tipo de soluciones pueden ser prácticas en algunos casos (en pequeña escala como una pequeña empresa), sin embargo presenta una gran desventaja en medios masivos como la Internet.

- Manejo de llaves. Se refiere a que en una red que tiene N estaciones de trabajo, cada estación deba de poseer una llave distinta por cada una de las $N - 1$ estaciones de trabajo restantes. Al igual que en el caso anterior se puede emplear un servidor común que se encargue de realizar dicho trabajo, para así evitar que cada una de la entidades de la red tenga que almacenar varias llaves. Pero al igual que en el caso del problema de distribución de llaves, este tipo de solución no es práctica en ámbitos como el de la Internet.

2.2. Criptografía de llave pública

El concepto de criptografía de llave pública, descrito en la figura 2.2, fue propuesto por W. Diffie y M. Hellman [5], esto con el propósito de evitar las desventajas que presenta la criptografía de llave privada mencionadas en la sección previa.

W. Diffie y M. Hellman proponen un protocolo criptográfico denominado “intercambio de llaves Diffie-Hellman” (*Diffie-Hellman key exchange*), el cual permite a dos entidades que no tienen conocimiento previo entre ellas, de poder crear un secreto compartido (o una llave compartida), sobre un canal inseguro de comunicación (como la Internet). Este secreto

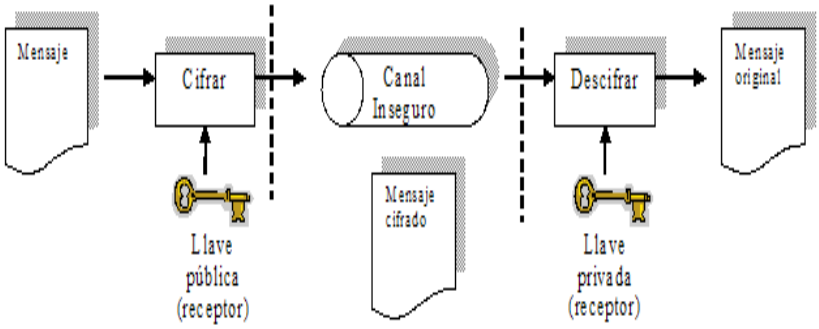

Figura 2.2: Modelo de criptografía de llave pública.

o llave puede ser empleado posteriormente para cifrar mensajes empleando un cifrador de llave simétrica.

Dicho protocolo emplea un grupo multiplicativo de enteros módulo un número primo. Dadas dos entidades A, B, se puede describir el protocolo de la siguiente forma (ver figura 2.3):

1. A, B seleccionan un grupo cíclico G y un elemento generador $g \in G$, para poder describir el grupo multiplicativo de G.

2. La entidad A escoge al azar un número natural a y envía g^a hacia la entidad B.

3. La entidad B escoge al azar un número natural b y envía g^b hacia la entidad A.

4. A calcula $(g^b)^a$

5. B calcula $(g^a)^b$

Cada entidad A, B posee el valor g^{ab}, g^{ba} respectivamente, los cuales son iguales. Cabe señalar que los valores a, b, g^{ab}, g^{ba} son secretos (o dicho de otra forma, son desconocidos para otras entidades externas), ya que solamente los valores g^a, g^b son conocidos para todo mundo. Como $g^{ab} = g^{ba}$, este elemento del grupo G puede servir como llave compartida entre A, B, puesto que sólo ellos conocen dicho valor. Además esta llave compartida sirve como llave de cifrado para poder enviar mensajes de una forma segura sobre canales públicos de comunicación.

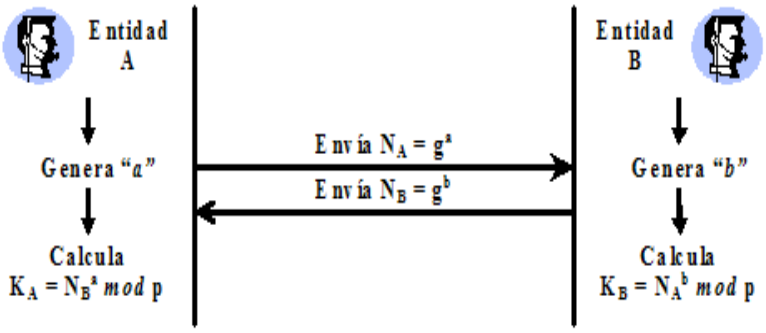

Figura 2.3: Protocolo de Diffie-Hellman.

A diferencia de los esquemas basados en criptografía de llave privada, los esquemas de criptografía de llave pública sólo necesitan que se intercambien las llaves públicas respectivas de cada entidad; dichas llaves son auténticas y no secretas, evitando así el intercambio de información secretas por un canal inseguro.

Es gracias al concepto de criptografía de llave pública que nace el concepto de firma digital.

2.2.1. Firma Digital

Se puede definir la firma digital como un método que se emplea para autenticar información digital, esta firma digital es análoga a la firma manuscrita. Para su implementación se emplean técnicas de criptografía de llave pública.

La firma digital en la transmisión de mensajes electrónicos, sirve como método criptográfico que asegura:

- Autenticidad: Un criptosistema de llave pública permite que cualquier persona envíe un mensaje empleando criptografía de llave pública. Una firma permite al receptor de un mensaje estar seguro de que la entidad que envió el mensaje, es en verdad quien dice ser. Esta propiedad es importante sobre todo en ámbitos financieros, supóngase que una oficina de un banco realiza una transferencia de dinero (por ejemplo un depósito) hacia las oficinas centrales, si no se puede comprobar la identidad de la entidad que realiza dicha operación, cualquier persona podría depositar dinero en las oficinas centrales del banco.

- Integridad: Se refiere a que tanto el emisor como el receptor puedan estar seguros que la información que se envía no ha sido alterada por una tercera persona. El cifrado de

datos podría proporcionar esta propiedad; ya que la información no puede ser leída, sin embargo la información puede ser alterada de alguna forma.

- No repudio: En el ámbito criptográfico se refiere, a que si un mensaje es enviado, el emisor no pueda negar el envío de dicho mensaje. Esta propiedad es importante ya que puede prevenir el envío de mensajes apócrifos, o prevenir fraudes electrónicos.

Los esquemas de firma digital son la contraparte de las firmas escritas, ya que una firma digital puede ser empleada para proveer a un documento digital: autenticidad del origen del documento, integridad del documento y no repudio del mismo. Los esquemas de firma digital son comúnmente usados por autoridades de certificación para firmar certificados que conjuntan la entidad con su llave pública.

Un esquema de firma digital consiste principalmente de cuatro algoritmos [3]:

1. Un algoritmo generador de parámetros de dominio, el cual genera el conjunto D.

2. Un algoritmo de generación de llaves, el cual tiene como entrada el dominio de parámetros D, y crea como salida el par de llaves (Q, d) (pública y privada respectivamente).

3. Un algoritmo de firma digital, el cual tiene como parámetros de entrada el conjunto D de parámetros, la llave privada d, y el mensaje m a enviar, y produce como salida la firma σ.

4. Un algoritmo de verificación, que toma como entradas el conjunto D de parámetros, la llave pública Q, el mensaje recibido m, así como la firma adjunta σ, y da como salida la aceptación o negación de la firma.

Un esquema de firma digital se dice que es seguro (según Goldwasser [25]), si a pesar que el atacante tiene todos los recursos y medios necesarios para tratar de obtener la información que desea, no pueda obtenerla. Es decir, que aun si el atacante pueda obtener la firma de algún documento de su elección, no por ello pueda ser capaz de generar la misma firma válida para otro tipo de documento que el atacante quiera enviar.

Una firma digital se crea al ejecutar un texto de mensaje a través de un algoritmo de resumen (función hash). Esto genera un resumen de mensaje. El resumen de mensaje es posteriormente cifrado usando la llave privada del individuo que está enviando el mensaje, convirtiéndolo en una firma digital (ver figura 2.4). La firma digital sólo puede ser descifrada por la llave pública del mismo individuo.

Para realizar la verificación de la firma digital, el receptor del mensaje descifra la firma digital y luego recalcula el resumen del mensaje. El valor de este resumen de mensaje

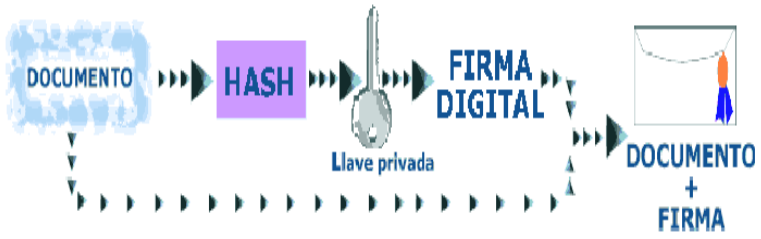

Figura 2.4: Firma digital.

recién calculado se compara con el valor del resumen del mensaje encontrado en la firma (ver figura 2.5). Si los dos coinciden, quiere decir que el mensaje no ha sido alterado. Ya que la llave pública del remitente fue utilizada para verificar la firma, el texto debe haber sido firmado con la llave privada conocida sólo por el remitente. El proceso total de autenticación será incorporado dentro de cualquier aplicación que use seguridad.

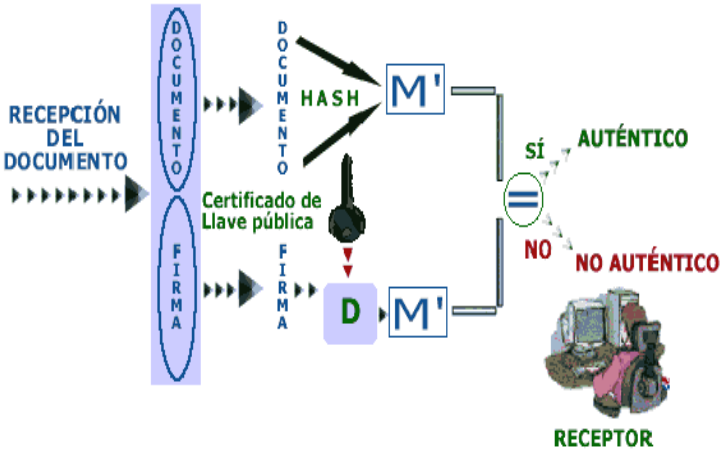

Figura 2.5: Verificación de la firma digital.

2.3. Criptosistemas de llave pública

Como se observó en la sección previa, en la firma digital basada en llave pública, cada entidad crea un par de llaves (e, d), que consiste de una llave pública (e) y una llave privada

(d) que se debe de conservar de manera secreta. Este par de llaves deben estar enlazadas de tal forma que sea computacionalmente difícil determinar la llave secreta conociendo la llave pública. En base a problemas matemáticos que son considerados difíciles de resolver, se han creado esquemas de criptografía de llave pública, los más comúnmente empleados son:

1. El problema de factorización de enteros, que debido a su dificultad es la base de los esquemas de cifrado y firma digital de tipo RSA.

2. El problema del logaritmo discreto, que es la base del esquema de cifrado y firma digital de tipo ElGamal, así como sus variantes como DSA (Digital Signature Algorithm).

3. El problema del logaritmo discreto en curvas elípticas, el cual es la base de los esquemas de criptografía de curvas elípticas.

2.3.1. Sistemas de tipo RSA

Este tipo de sistemas llevan el nombre de sus inventores Ronald Rivest, Adi Shamir y Leonard Adleman ([6]), es de los primeros sistemas de llave pública que fueron propuestos e implementados.

La fortaleza de este tipo de sistemas criptográficos, reside en la intratabilidad matemática del problema de la factorización de enteros (IFP). El cual consiste en encontrar un divisor no trivial de un número compuesto, por ejemplo dado el número 91, el reto está en encontrar el número 7, el cual divide dicho número. En la práctica los sistemas de RSA poseen un tamaño de llave de al menos 1024 bits, lo que es equivalente al nivel de seguridad proporcionado por una llave de 80 bits de un sistema simétrico.

2.3.1.1. Generación de llaves

Un par de llaves de tipo RSA se pueden generar empleando el algoritmo descrito en la tabla 2.1. La llave pública consiste del par de enteros (n, e), donde n es el producto de dos enteros primos p, q (escogidos aleatoriamente), los cuales tienen la misma longitud en bits. El exponente e es un entero que satisface $1 < e < \phi$ y $mcd(e, \phi) = 1$, donde $\phi = (p-1)(q-1)$. La llave privada d es un entero que satisface $1 < d < \phi$ y $ed \equiv 1 (\text{mod } \phi)$. Se ha demostrado que el problema para calcular la llave privada d a partir de la llave pública (n, e) es computacionalmente equivalente a calcular los factores p, q de n, es decir a resolver el problema de la factorización de enteros (IFP).

RSA, generación de llaves.
Entradas: Parámetro de seguridad l.
Salida: Llave pública (n, e) y llave privada d.

1. Seleccionar aleatoriamente dos números primos p, q con la misma longitud de $l/2$ bits.
2. Calcular $n = p$ y $\phi = (p-1)(q-1)$.
3. Seleccionar un entero aleatorio e, tal que $1 < e < \phi$ y $mcd(e, \phi) = 1$.
4. Calcular d tal que $1 < d < \phi$ y $ed \equiv 1 (\text{mod } \phi)$.
5. Devolver (n, e, d).

Tabla 2.1: Generación de llaves en RSA.

2.3.1.2. Esquema de firma digital y verificación

Los procedimientos para realizar las operaciones de firma y verificación son mostrados en las tablas 2.2 y 2.3. La entidad que envía un mensaje m debe primero calcular el hash $h = H(m)$ empleando una función hash H, donde h funciona como una marca única de m. Luego quien envía el mensaje, ocupa su propia llave privada d para calcular $s = h^d \bmod n$. En seguida quien envía el mensaje transmite el mensaje m, así como la firma s al receptor. El receptor debe entonces calcular el hash $h = H(m)$, y además debe de calcular $h' = s^e \bmod n$ a partir de s; la firma s de m será válida si y solo si $h' = h$. La seguridad de este esquema recae en la intratabilidad matemática de calcular la llave privada d a partir de la firma s y la llave pública (n, e).

La operación más costosa en RSA es la exponenciación modular, i.e. calcular $m^d \bmod n$; para la firma, y $c^e \bmod n$ en la verificación. Por lo que normalmente se emplea un exponente chico e (del orden de $e = 3$ ó $e = 2^{16} + 1$), para tratar de agilizar la exponenciación modular. Así en el esquema de RSA las operaciones de cifrado y verificación con un exponente chico e es más rápido que realizar las operaciones de firma y descifrado de mensajes.

2.3.2. Sistemas basados en el DLP

El primer sistema basado en logaritmos discretos (DL), fue propuesto para ser empleado en un protocolo de intercambio de llaves, es decir como un sistema de llave pública (el

RSA, generación de firma digital.
Entradas: Llave pública (n, e), llave privada d, mensaje.
Salida: Firma s.

1. Calcular $h = H(m)$, siendo H una función hash.
2. Calcular $s = h^d \bmod n$.
3. Devolver s.

Tabla 2.2: Firma digital empleando RSA.

RSA, verificación de firma digital.
Entradas: Llave pública (n, e), mensaje m, firma s.
Salida: Aceptación o denegación de la firma.

1. Calcular $h = H(m)$.
2. Calcular $h' = s^e \bmod n$.
3. Si $h = h'$, entonces aceptar firma; si no desechar firma.

Tabla 2.3: Verificación de firma empleando RSA.

cual fue ideado por W. Diffie y M. Hellman en 1976). En 1984, fue propuesto el sistema ElGamal [26], el cual describe un sistema de llave pública para cifrado de datos, y para firma digital de documentos; que está basado en logaritmos discretos. A partir de este esquema es que se propuso el algoritmo de firma digital (DSA) [9].

Supóngase que se tiene el grupo cíclico multiplicativo $(G, \cdot)$, de orden n con un generador g. Los parámetros del dominio son g, n, se determina una llave privada al azar seleccionando una x entera aleatoria dentro del intervalo $[1, n-1]$, y su respectiva llave pública se calcula como $y = g^x$. El problema de calcular x dados g, n, y se conoce como el problema del logaritmo discreto en G (DLP).

Problema DLP: *Dados un grupo cíclico finito G de orden n, un generador $g \in G$, y un elemento $y \in G$, encontrar el entero x, $0 \leq x \leq n-1$, tal que $g^x = y$.*

DL, generación de parámetros.
Entradas: Parámetros de seguridad l, t.
Salida: Parámetros (p, q, g).

1. Seleccionar un número primo q de longitud de t bits y un entero primo p de longitud l bits tal que q divida $p - 1$.
2. Seleccionar un elemento g de orden q:
   a) Seleccionar un elemento arbitrario $h \in [1.p - 1]$ y calcular $g = h^{(p-1)/q} \bmod p$.
   b) Si $g = 1$ entonces ir al paso **2.a**
3. Devolver (p, q, g)

Tabla 2.4: Generación de parámetros públicos en DL.

2.3.2.1. Generación de llaves

En los sistemas basados en logaritmos discretos, el par de llaves que se genera está asociado con un conjunto de parámetros públicos (p, q, g). Donde p es un número primo, q es un primo que es divisor de $p - 1$, y $g \in [1, p - 1]$, el cual tiene un orden de q (es decir, $t = q$ es el entero positivo más pequeño tal que $g^t \equiv 1(\bmod p)$). La llave privada es un entero x que es seleccionado aleatoriamente dentro del intervalo $[1, q - 1]$, denotado $x \in R_{[1,q-1]}$, y su llave pública correspondiente es $y = g^x \bmod p$. La dificultad de este esquema radica en determinar x, dados el conjunto de parámetros públicos (p, q, g) y y; esto se conoce como el problema del logaritmo discreto (DLP). Los procedimientos para generar los parámetros públicos y el par de llaves (pública/privada) son descritos en las tablas 2.4 y 2.5

2.3.2.2. Esquema de firma digital y verificación

El Algoritmo de Firma Digital (DSA) fue propuesto en 1991 [27] por el NIST (U. S. National Institute of Standards and Technology) y fue especificado por el FIPS [9] (U. S. Government Federal Information Processing Standard), el cual se denominó como Estándar para Firma Digital (DSS). Las operaciones de firma y verificación son mostrados en las tablas 2.6 y 2.7.

Una entidad A que posee una llave privada x, firma un mensaje escogiendo aleatoria-

DL, generación de llaves.
Entradas: Parámetros públicos (p, q, g).
Salida: Llave pública y y llave privada x.

1. Seleccionar $x \in R_{[1,q-1]}$.
2. Calcular $y = g^x \bmod p$.
3. Devolver x, y.

Tabla 2.5: Generación de llaves en DL.

mente un entero k dentro del intervalo $[1, q-1]$, y calculando:

$$T = g^k \bmod p \tag{2.1}$$

$$r = T \bmod q \tag{2.2}$$

y

$$s = k^{-1}(h + xr) \bmod q \tag{2.3}$$

donde $h = H(m)$, siendo h el resumen (hash) del mensaje. La firma del mensaje m es el par de enteros (r, s). Para verificar la firma, el receptor del mensaje debe de comprobar que (r, s) satisfacen la ecuación 2.3. Puesto que el receptor desconoce la llave privada x de A, así como el entero k, esta ecuación no puede ser comprobada directamente de la ecuación 2.3. Sin embargo, cabe señalar que la ecuación 2.3 es equivalente a la siguiente ecuación:

$$k \equiv s^{-1}(h + xr) \bmod q \tag{2.4}$$

Al elevar g al exponente k, da como resultado la siguiente congruencia:

$$\begin{aligned} g^k &\equiv g^{s^{-1}(h+xr)} (\bmod q) \\ g^k &\equiv (g^{s^{-1}h} g^{s^{-1}xr}) (\bmod q) (\bmod p) \\ T &\equiv g^{hs^{-1}} y^{rs^{-1}} (\bmod p) \end{aligned} \tag{2.5}$$

por lo que el receptor puede calcular T y así poder comprobar que $r = T \bmod q$.

DSA, generación de firma digital.
Entradas: Parámetros públicos (p, q, g), llave privada x, mensaje m.
Salida: Firma (r, s).

1. Escoger k tal que $k \in R_{[1,q-1]}$.
2. Calcular $T = g^k \bmod p$.
3. Calcular $r = T \bmod q$. Si $r = 0$, entonces regresar al paso **1**.
4. Calcular $h = H(m)$.
5. Calcular $s = k^{-1}(h + xr) \bmod q$. Si $s = 0$, entonces regresar al paso **1**.
6. Devolver (r, s).

Tabla 2.6: Firma digital empleando DSA.

2.4. Curvas Elípticas contra otros Métodos de Criptografía de Llave Pública

Existen ciertos factores y criterios a considerar cuando se trata de implementar un sistema que emplee criptografía de llave pública, entre los principales factores están: la funcionalidad, es decir qué tanta capacidad tiene la aplicación; la seguridad, se refiere a que tanta confianza se tiene en el esquema; y también hay que tomar en consideración el desempeño, es decir si la aplicación cumple las expectativas esperadas de ella. Hay otros factores que influyen en la decisión de qué esquema de llave pública utilizar como los estándares que hay disponibles, las aplicaciones y productos existentes, entre otros.

Los esquemas RSA, ElGamal y ECC proveen todos el funcionamiento básico esperado de los esquemas de criptografía de llave pública esto es: cifrado / descifrado, firma / verificación, intercambio y generación de llaves. La seguridad de estos sistemas como ya se ha mencionado previamente recae en la intratabilidad y complejidad de los problemas matemáticos en que están basados (factorización de enteros, problema del logaritmo discreto, y el problema del logaritmo discreto en curvas elípticas). Es debido a la dificultad y complejidad matemática que presentan estos problemas que los criptosistemas que están basados en estos esquemas son seguros, y tienen gran resistencia para soportar ataques de

DSA, verificación de firma digital.
Entradas: Parámetros públicos (p, q, g),llave pública x, mensaje m, firma (r, s).
Salida: Aceptación o denegación de la firma.

1. Verificar que r, s sean dos enteros dentro del intervalo $[1, q-1]$. Si la verificación falla, entonces la firma no es válida.
2. Calcular $h = H(m)$.
3. Calcular $w = s^{-1} \bmod q$.
4. Calcular $u_1 = hw \bmod q$ y $u_2 = rw \bmod q$.
5. Calcular $T = g^{u_1} y^{u_2} \bmod p$.
6. Calcular $r' = T \bmod q$.
7. Si $r = r'$, entonces la firma es válida, en caso contrario la firma es inválida.

Tabla 2.7: Verificación de firma empleando DSA.

criptoanálisis. El tiempo de ejecución que emplean las operaciones y cálculos que se requieren para realizar las operaciones de cifrado o firma, dependen del tamaño del dominio y del tamaño de la llave que se emplea. Por lo que si el dominio y el tamaño de llave es demasiado grande; el desempeño de las operaciones aritméticas no es del todo eficiente, y esto afecta en el desempeño de las aplicaciones que emplean primitivas criptográficas.

En la actualidad existen algoritmos que son capaces que resolver tanto el problema de factorización, como los problemas basados en logaritmos discretos, en un tiempo subexponencial. El algoritmo de Number Field Sieve (NFS) [28] permite resolver el IFP, los algoritmos de Number Field Sieve (NFS) [29] y el algoritmo Rho-Pollard [30] permiten resolver los problemas de logaritmos discretos.

Para el caso de RSA, el mayor número conocido en la actualidad que ha sido factorizado con el método NFS es un número de 576 bits (un entero de 160 dígitos) en diciembre del 2003 [31]. Y para ECDSA, el mayor ejemplo conocido que ha sido resuelto con el método de Pollard es (en el caso de números primos) es el de una curva elíptica sobre un campo finito primo de 109 bits (en noviembre del 2002 [32]).

Los tamaños de llaves que se van a describir proveen un nivel de seguridad equiparable

	Nivel de seguridad (en bits)				
	80 (SKIPJACK)	112 (Triple-DES)	128 (AES-Small)	192 (AES-Medium)	256 (AES-Large)
DL parámetro q EC parámetro n	160	224	256	384	512
RSA parámetro n DL parámetro p	1024	2048	3072	8192	15360

Tabla 2.8: Tabla de comparación para distintos tamaños de llave (tomada de [3]).

para los esquemas de RSA, ElGamal y ECC. Lo que anteriormente se mencionó como los parámetros de los algoritmos, son también conocidos como el tamaño de llave que se emplea dentro de cada esquema de criptografía de llave pública. En la tabla 2.8 se puede observar que se emplean tamaños de llave de 80, 112, 128, 192 y 256 (en criptografía de llave privada) tienen un nivel de seguridad que es considerado igual a un tamaño de llave en criptografía de llave pública descritos en la tabla 2.8, esta tabla fue tomada del libro [3].

Las comparaciones que se observan en la tabla 2.8, muestran que los tamaños de llave más pequeñas que se emplean para un determinado nivel de seguridad, los proporciona el esquema de ECC. Esta diferencia se vuelve más marcada con tamaños de llaves grandes (es decir para un nivel mayor de seguridad). Un tamaño de llave pequeño implica una mayor velocidad de ejecución, ya que requiere un menor poder de cómputo para realizar las operaciones. En general las operaciones de criptografía de llave pública realizadas con ECC son más rápidas y eficientes que en el caso de RSA. Este tipo de ventajas brindadas por ECC, pueden ser importantes en ambientes de desarrollo donde las capacidades de cómputo son menores y limitadas, donde también el ancho de bando es limitado y el consumo de potencia es mínimo.

Una de las aplicaciones más comunes en que se emplea la criptografía de llave pública es la firma digital de documentos electrónicos.

Capítulo 3

Aspectos Matemáticos

Este capítulo introduce en los conceptos más relevantes de campos finitos [33]. Este tipo de campos son relevantes en criptografía de curvas elípticas, ya que sobre ellos se definen las operaciones aritméticas de curvas elípticas [34, 3]. En particular se explican los operaciones aritméticas sobre campos finitos binarios (los cuales son de sumo interés en implementaciones de hardware), que a su vez forman el bloque básico para calcular la multiplicación escalar en curvas elípticas.

El material que se presenta en este capítulo está basado en [35, 3, 36, 14, 37]

3.1. Campos finitos

3.1.1. Anillos

Definición 1 *Un anillo R es un conjunto donde los elementos que lo conforman pueden ser sumados o multiplicados entre ellos, y que satisfacen las siguientes condiciones:*

- *Bajo la adición, el anillo R es un grupo aditivo (Abeliano).*
- *Para todos $x, y, z \in R$, se tiene que $x(y + z) = xy + xz; (y + z)x = yx + zx$*
- *Para todos $x, y \in R$, se tiene que $(xy)z = x(yz)$*
- *Existe un elemento $e \in R$, tal que $ex = xe = x$ para todo $x \in R$*

Los números enteros, racionales, reales y complejos son ejemplos de anillos. Se dice que un elemento $x \in \mathbb{R}$ tiene un inverso si x tiene un inverso multiplicativo en $\mathbb{R}$, si existe un elemento único $u \in \mathbb{R}$, tal que $xu = ux = 1$. Este elemento 1 es denominado la unidad del anillo.

3.1.2. Campos

Definición 2 *Un campo F es un anillo en el cual la multiplicación es conmutativa y para todo elemento e ∈ F, exceptuando el elemento nulo denotado* 0*, existe un inverso multiplicativo. El campo F tiene las siguientes características:*

- *Sean* $x, y \in F$ *entonces* $x + y = y + x$
- *Sean* $x, y \in F$ *entonces* $x \cdot y = y \cdot x$
- *Para todo* $x, y, z \in F$*, se tiene que* $x(y + z) = xy + xz; (y + z)x = yx + zx$

3.1.3. Campos finitos

Definición 3 *Un campo finito o campo de Galois, denotado como* $GF(p^n)$*, es un campo de característica p y con un numero* $q = p^n$ *de elementos. Un campo finito es un sistema algebraico que consiste del conjunto GF junto con los operadores binarios + y · definidos en GF; y que satisfacen los siguientes axiomas:*

- *GF es un grupo abeliano con respecto a la suma*
- $GF \setminus \{0\}$ *es un grupo abeliano con respecto a la multiplicación*
- *Para todos* $x, y, z \in GF$*, tenemos* $x \cdot (y + z) = (x \cdot y) + (x \cdot z), (x + y) \cdot z = (x \cdot z) + (y \cdot z)$

3.1.4. Campos finitos binarios

Un campo de la forma $GF(2^m)$ es conocido como un campo finito de característica dos o como campo finito binario. Este campo tiene $2^m - 1$ elementos no cero, más el elemento nulo denotado ϑ. Un elemento arbitrario $A \in GF(2^m)$, es un polinomio de grado menor a m tal que:

$$A = a(z) = a_{m-1}z^{m-1} + \cdots + a_1 z + a_0 = \sum_{i=0}^{m-1} a_i z^i, \text{ donde } a_i \in \{0, 1\} \tag{3.1}$$

Donde $a(z)$ se conoce como la representación en base polinomial de los elementos que pertenecen a $GF(2^m)$. Normalmente este polinomio se puede representar como un vector $(a_{m-1}, \cdots, a_1, a_0)$.

3.2. Aritmética de campos finitos

Una vez descrito el espacio de trabajo, se definen el conjunto de operaciones que se pueden realizar dentro del campo finito binario $GF(2^m)$.

3.2.1. Suma

Dados $A = a(z) = (a_{m-1}, \cdots, a_1, a_0)$ y $B = b(z) = (b_{m-1}, \cdots, b_1, b_0) \in GF(2^m)$, sea $C = c(z)$ la suma de esos dos polinomios, entonces:

$$A + B = C = c(z) = (c_{m-1}, \cdots, c_1, c_0) \tag{3.2}$$

donde $c_i = (a_i + b_i) \bmod 2$.

3.2.2. Multiplicación

Dados $A = a(z) = (a_{m-1}, \cdots, a_1, a_0)$ y $B = b(z) = (b_{m-1}, \cdots, b_1, b_0) \in GF(2^m)$, sea $C = c(z)$ la multiplicación de esos dos polinomios, entonces:

$$A \cdot B = C = c(z) = (c_{m-1}, \cdots, c_1, c_0) \tag{3.3}$$

donde $c(z) \in GF(2^m)$ es el elemento resultante de la operación $a(z) \cdot b(z) \bmod p(z)$, donde $p(z)$ es un polinomio irreducible de grado m.

Para poder obtener $c(z)$, se puede calcular primero el producto polinomial $c'(z)$ resultante que es un polinomio de grado a lo más de $2m - 2$, mediante la operación:

$$c'(z) = a(z) \cdot b(z) = \left(\sum_{i=0}^{m1} a_i z^i \right) \left(\sum_{i=0}^{m1} b_i z^i \right) \tag{3.4}$$

Para después en un segundo paso realizar la operación de reducción para poder obtener un polinomio $c(z)$ de grado a lo más de $m - 1$, el cual podemos definir como:

$$c(z) = c'(z) \bmod p(z) \tag{3.5}$$

La forma en que se calcula $c(z)$ por métodos tradicionales (en hardware), emplea multiplicadores paralelos con complejidad espacial $O(m^2)$. Se han hecho diversas investigaciones para poder realizar la multiplicación de campos finitos binarios de una manera rápida y eficiente, por lo que diversas estrategias en paralelo han sido propuestas [38, 39, 40, 41]. Sin embargo todas estas estrategias también tienen la misma complejidad cuadrática. Existe una estrategia que puede reducir el costo de área: el algoritmo de Karatsuba-Ofman.

3.2.3. Multiplicadores de Karatsuba-Ofman

Este algoritmo, fue propuesto en 1962, fue el primero en realizar la operación de multiplicación en paralelo de campos finitos binarios con menos de $O(m^2)$ operaciones [35]. Los multiplicadores de Karatsuba-Ofman están basados en la siguiente observación:

Sea el campo finito $GF(2^m)$, construido a partir del polinomio irreducible $p(z)$, de grado $m = rn$, con $r = 2^k$, siendo k un entero positivo. Sean A, B dos elementos que pertenecen al campo $GF(2^m)$, ambos elementos pueden ser representados en base polinomial como:

$$A = \sum_{i=0}^{m-1} a_i z^i = \sum_{i=\frac{m}{2}}^{m-1} a_i z^i + \sum_{i=0}^{\frac{m}{2}-1} a_i z^i = z^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} a_{i+\frac{m}{2}} z^i + \sum_{i=0}^{\frac{m}{2}} a_i z^i = z^{\frac{m}{2}} A^H + A^L \tag{3.6}$$

y

$$B = \sum_{i=0}^{m-1} b_i z^i = \sum_{i=\frac{m}{2}}^{m-1} b_i z^i + \sum_{i=0}^{\frac{m}{2}-1} b_i z^i = z^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} b_{i+\frac{m}{2}} z^i + \sum_{i=0}^{\frac{m}{2}} b_i z^i = z^{\frac{m}{2}} B^H + B^L \tag{3.7}$$

Usando las ecuaciones 3.6 y 3.7, el producto polinomial está dado por:

$$C = z^m A^H B^H + (A^H B^L + A^L B^H) z^{\frac{m}{2}} + A^L B^L \tag{3.8}$$

El algoritmo de Karatsuba-Ofman está basado en la idea de que el producto de la ecuación 3.8, puede ser escrita equivalentemente como:

$$C = z^m A^H B^H + (A^H B^H + A^L B^L + (A^H + A^L)(B^L + B^H)) z^{\frac{m}{2}} + A^L B^L = z^m C^H + C^L \tag{3.9}$$

A pesar de que la ecuación 3.9 en apariencia es más compleja que la ecuación 3.8, es fácil notar que el costo computacional de la ecuación 3.9 es de cuatro sumas y tres multiplicaciones polinomiales. Mientras que en el caso de la ecuación 3.8 se requieren calcular cuatro multiplicaciones y tres sumas polinomiales.

En general una multiplicación polinomial es una operación costosa en cuanto a espacio en área. Por ejemplo, los multiplicadores clásicos y paralelos [41, 42] presentan un complejidad en espacio del orden de $O(m^2)$, sin embargo el enfoque que emplea el multiplicador binario de Karatsuba-Ofman tiene una complejidad en espacio de $O(m^{\log_2 3})$ [4]. Por lo que es válido decir que le ecuación 3.9 tiene un menor costo en área en lugar de emplear un algoritmo clásico de multiplicación polinomial.

3.2.4. Elevar al cuadrado

Sea $A = a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0, A \in GF(2^m)$, entonces se puede definir la operación de elevar al cuadrado como:

$$A^2 = a_{m-1}z^{2m-2} + \cdots + a_2z^4 + a_1z^2 + a_0 \bmod p(z) \tag{3.10}$$

Dada la ecuación 4.3 se puede hallar equivalentemente una matriz M tal que:

$$C = \begin{bmatrix} c_0 \\ \vdots \\ c_i \\ \vdots \\ c_{m-1} \end{bmatrix} = A^2 = MA = \begin{bmatrix} m_{0,0} & \cdots & m_{0,i} & \cdots & m_{0,m-1} \\ \vdots & & & & \vdots \\ m_{j,0} & \cdots & m_{j,i} & \cdots & m_{j,m-1} \\ \vdots & & & & \vdots \\ m_{m-1,0} & \cdots & m_{m-1,i} & \cdots & m_{m-1,m-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{m-1} \end{bmatrix} \tag{3.11}$$

donde M es una matriz de orden $m \times m$. Si se puede encontrar la matriz M de la ecuación 3.11 de tal forma que al multiplicar la matriz M por un elemento aleatorio $A \in GF(2^m)$, se encuentra el elemento C tal que $C = A^2$.

Para obtener esta matriz M, primero se toma la representación en base polinomial de $A \in GF(2^m)$, luego se realiza la operación $C = A \cdot A \bmod p(x)$, y se obtienen un conjunto de ecuaciones para cada coeficiente del polinomio resultante.

Existen dos tipos de polinomios irreducibles que se emplean en campos finitos $GF(2^m)$: los trinomio y los pentanomios. En el caso de trinomios es posible obtener ecuaciones reducidas de forma general para calcular el cuadrado de un elemento arbitrario de $GF(2^m)$.

Se considera un trinomio irreducible de la forma $p(z) = z^m + z^n + 1$, dada esta forma del polinomio, se consideran tres casos de estudio:

- Caso I: Calcular $C = A^2$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m, n enteros pares y $n < m/2$.

$$c_i = \begin{cases} a_{i/2} & i \text{ impar}, i < n, \\ a_{i/2} + a_{i/2+(m-n)/2} + a_{m-n+i/2} & i \text{ impar}, n < i < 2n, \\ a_{i/2} + a_{i/2+(m-n)/2} & i \text{ impar}, i \geq 2n, \\ a_{(m+i)/2} + a_{m-(n-i)/2} & i \text{ par}, i < n, \\ a_{(m+i)/2} & i \text{ par}, i \geq n \end{cases} \tag{3.12}$$

- Caso II: Calcular $C = A^2$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m impar, n par y $n < m/2$.

$$c_i = \begin{cases} a_{i/2} + a_{(m+i)/2} & i \text{ impar}, i < n \text{ ó } i \geq 2n, \\ a_{i/2} + a_{(m+i)/2} + a_{m-n+i/2} & i \text{ impar}, n < i < 2n, \\ a_{m+1-(n+i)/2} & i \text{ par}, i < n, \\ a_{(m-n+i)/2} & i \text{ par}, i \geq n \end{cases} \tag{3.13}$$

- Caso III: Calcular $C = A^2$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m impar, n par y $n = m/2$.

$$c_i = \begin{cases} a_{i/2} + a_{(m+i)/2} & i \text{ impar}, i < n, \\ a_{i/2} & i \text{ even}, i > n, \\ a_{m+1-(n/2+i)/2} & i \text{ par}, i < n, \\ a_{(n/2+i)/2} & i \text{ par}, i \geq n, \end{cases} \tag{3.14}$$

3.2.5. Raíz cuadrada

Sea $A = a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0, A \in GF(2^m)$, para calcular $\sqrt{A}$ se emplea el Teorema Pequeño de Fermat: $A^{2^m} = A$, entonces:

$$\sqrt{A} = A^{2^{m-1}} \tag{3.15}$$

Otra aproximación para extraer la raíz cuadrada de un elemento de $GF(2^m)$ es emplear la ecuaciones mostradas en §3.2.4. Primero se considera un elemento arbitrario $C \in GF(2^m)$, en base polinomial. Como se observo en la sección previa se puede obtener el cuadrado de un elemento arbitrario de $GF(2^m)$ mediante la ecuación 3.11.

Por lo que de una manera análoga, dado A, se puede calcular $C = \sqrt{A}$ resolviendo la siguiente ecuación,

$$C = \begin{bmatrix} c_0 \\ \vdots \\ c_i \\ \vdots \\ c_{m-1} \end{bmatrix} = M^{-1}A = \begin{bmatrix} m_{0,0} & \cdots & m_{0,i} & \cdots & m_{0,m-1} \\ \vdots & & & & \vdots \\ m_{j,0} & \cdots & m_{j,i} & \cdots & m_{j,m-1} \\ \vdots & & & & \vdots \\ m_{m-1,0} & \cdots & m_{m-1,i} & \cdots & m_{m-1,m-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{m-1} \end{bmatrix} \tag{3.16}$$

donde la matriz inversa M^{-1} es una matriz de orden $m \times m$. Así, para calcular $C = \sqrt{A}$ de un elemento $A \in GF(2^m)$, primero se encuentra la matriz inversa M^{-1} (si es que existe) de la matriz M a partir de la ecuación (3.16). Una vez que se halla dicha matriz inversa, se procede a calcular $C = M^{-1}A$.

A partir de las ecuaciones 3.12, 3.13, 3.14, se observa que la operación de elevar al cuadrado es una operación lineal. Por lo que se pueden obtener expresiones cerradas como en el caso de las ecuaciones para elevar al cuadrado. Se consideran tres casos de estudio.

- Caso I: Calcular D tal que $D^2 = A$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m, n enteros impares y $n < m/2$:

$$d_i = \begin{cases} a_{2i} & i < \lceil n/2 \rceil, \\ a_{2i} + a_{2i-n} & \lceil n/2 \rceil \leq i < \lceil m/2 \rceil, \\ a_{2i-n} + a_{2i-m} & \lceil m/2 \rceil \leq i < (m+n)/2, \\ a_{2i-m} & (m+n)/2 \leq i \end{cases} \tag{3.17}$$

- Caso II: Calcular D tal que $D^2 = A$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m impar, n par, y $n < m/2$:

$$d_i = \begin{cases} a_{2i} + a_{2i+n} & i < \lfloor n/2 \rfloor, \\ a_{2i} + a_{(2i+n) \text{ mód } m} + a_{2i-n} & \lfloor n/2 \rfloor < i < n, \\ a_{2i} + a_{(2i+n) \text{ mód } m} & n \leq i < m/2, \\ a_{(2i+n) \text{ mód } m} & i \geq m/2 \end{cases} \tag{3.18}$$

- Caso III: Calcular D tal que $D^2 = A$ mód $p(z)$, con $p(z) = z^m + z^n + 1$, m impar, n par y $n = m/2$:

$$d_i = \begin{cases} a_{2i} + a_{2i+m/2} & i < (m+2)/4, \\ a_{2i} & i = (m+2)/4 \end{cases} \tag{3.19}$$

3.2.6. Traza

La función traza se define como:

$$Tr(c) = c + c^2 + c^{2^2} + \ldots + c^{2^{m-1}} \tag{3.20}$$

Y tiene las siguientes propiedades [3]:

Propiedades. Sea $c, d \in GF(2^m)$.

1. $Tr(c) = Tr(c^2) = Tr(c)^2$, y $Tr(c) \in 0, 1$.
2. La función traza es lineal y por tanto, $Tr(c + d) = Tr(c) + Tr(d)$.
3. Si $(u, v) \in GF(2^m)$, entonces $Tr(u) = Tr(a)$, donde a es el coeficiente de la ecuación $y^2 + xy = x^3 + ax^2 + b$

3.2.7. Media Traza

Sea m un entero par, la función media traza ("half-trace") se define como:

$$H(c) = \sum_{i=0}^{(m-1)/2} c^{2^{2i}} \tag{3.21}$$

Y tiene las siguientes propiedades [3]:

Propiedades. Sea m un entero par.

1. $H(c + d) = H(c) + H(d)$, para todo $c, d \in GF(2^m)$.

2. $H(c)$ es una solución a la ecuación $x^2 + x = c + Tr(c)$.

3. $H(c) = H(c^2) + c + Tr(c)$, para todo $c \in GF(2^m)$

3.3. Curvas elípticas sobre los reales

Durante los últimos 30 años, se ha estudiado ampliamente la teoría de curvas elípticas con fines criptográficos. Los criptosistemas basados en curvas elípticas fueron propuestos por N. Koblitz [7] y V. Miller [8]. A partir de estas propuestas se han realizado un sin número de estudios y análisis. En la actualidad, los ECC son ampliamente empleados para aplicaciones de generación de llaves, firma y verificación de documentos digitales.

Las curvas elípticas pueden ser definidas sobre los números reales, números complejos u otro tipo de campos. Primeramente se explicarán las curvas elípticas definidas sobre los reales, ya que gracias a su representación geométrica se pueden comprender mejor sus propiedades. Sin embargo, desde el punto de vista criptográfico, sólo se emplean curvas elípticas sobre campos finitos, en particular en este trabajo se enfoca el estudio de curvas elípticas definidas sobre campos finitos binarios.

Definición 4 *Sea E una curva elíptica sobre el campo de los reales* $\mathbb{R}$ *de la forma:* $y^2 + xy = x^3 + ax + b$ *(a), donde* $a, b \in \mathbb{R}$, $b \neq 0$. *Un punto P de coordenadas* (x, y), *pertenece a E si satisface la ecuación (a), además del punto en el infinito, que se simboliza como* ϑ.

A continuación se presenta un ejemplo de curva elíptica de la forma $y^2 = x^3 + ax + b$ en $\mathbb{R}$ (ver figura 3.1):

El negativo de un punto P con coordenadas (x, y), es el reflejo de este punto sobre el eje de las x's, es decir el punto $-P$ con coordenadas $(x, -y)$. Nótese que para cada punto P sobre la curva E, su punto inverso $-P$ también está sobre la curva.

El grupo de curvas elípticas sobre los reales forma un grupo aditivo, donde su operación básica es la suma. La suma de dos puntos de una curva elíptica se puede definir geométricamente (ver figura 3.2).

La figura 3.2 muestra la representación geométrica de la suma de puntos elípticos distintos. Supóngase que P_1, P_2 son dos puntos distintos sobre la curva E, y además $P_1 \neq -P_2$ se observa que la suma de puntos consiste en tomar la secante entre dos puntos P_1 y P_2 con coordenadas (x_1, y_1) y (x_2, y_2) respectivamente, esta secante interseca con la curva en un punto $-Q$ de coordenadas $(x_3, -y_3)$. Enseguida para obtener la suma $P_1 + P_2$, el punto

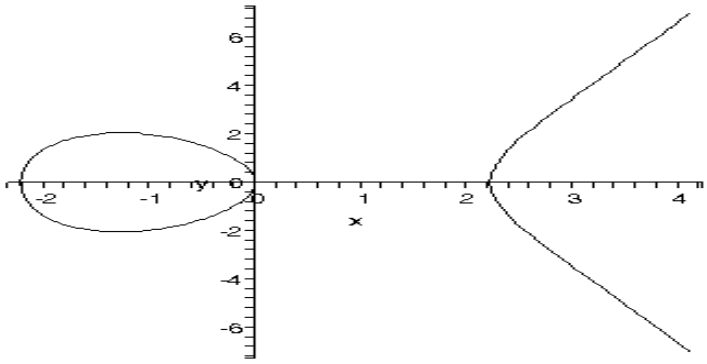

Figura 3.1: Curva elíptica de la forma $y^2 = x^3 + ax + b$.

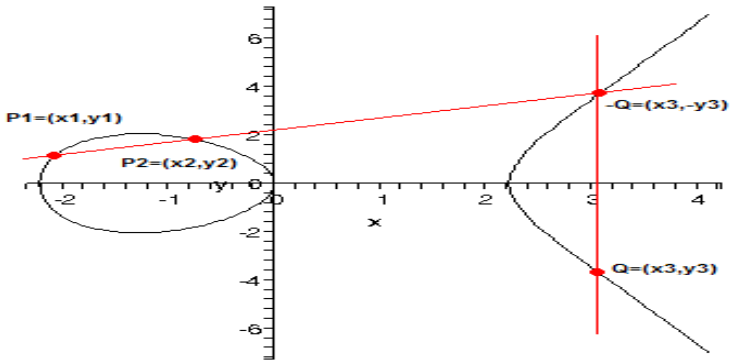

Figura 3.2: Representación gráfica de la suma de puntos elípticos.

$-Q$ se refleja sobre el eje de las x's para calcular así el punto $Q = P_1 + P_2$ de coordenadas (x_3, y_3).

En el caso de que sumen los puntos $P, -P$, geométricamente se tiene que al trazar la secante entre los puntos $P, -P$ se obtiene una línea vertical, la cual no interseca en un tercer punto con la curva elíptica E (ver figura 3.3). Por lo que la suma $P + (-P)$, no se puede obtener como en el caso anterior. Por esta razón es que el grupo de curvas elípticas incluye el punto al infinito es cual se denota como ϑ. Por definición $P + (-P) = \vartheta$, por lo que $P + \vartheta = P$, dentro del grupo aditivo de curvas elípticas. El elemento ϑ es conocido como la identidad del grupo de curvas elípticas y además toda curva elíptica posee un elemento identidad.

Se tiene un tercer caso de suma de puntos elípticos, en el cual se suma el punto P a sí mismo, este tipo de suma también es conocido como doblado de un punto elíptico. En su

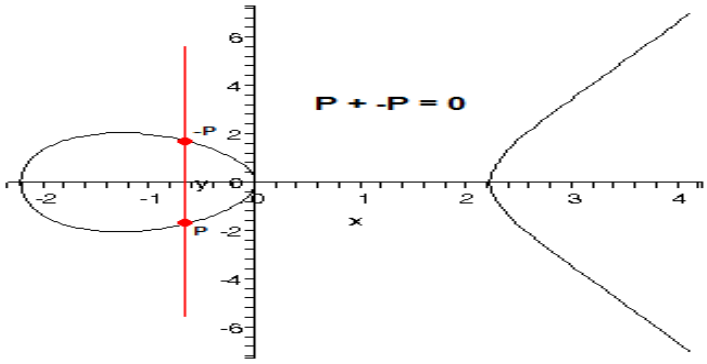

Figura 3.3: Representación gráfica de la suma de puntos elípticos.

representación geométrica (ver figura 3.4), se observa que el doblado de punto es tomar la tangente de un punto P de coordenadas (x_1, y_1) de la curva elíptica, esta tangente interseca con la curva en un punto $-P$ de coordenadas $(x_2, -y_2)$, luego para obtener el doble del punto, el punto $-P$ se refleja sobre el eje de las x's para poder obtener el punto $2P$ de coordenadas (x_2, y_2).

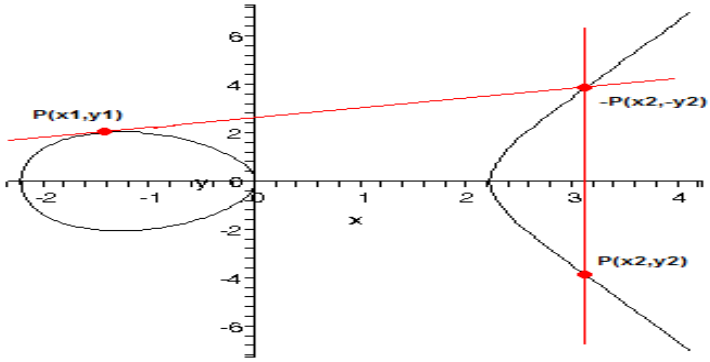

Figura 3.4: Representación gráfica del doblado de puntos elípticos.

3.4. Curvas elípticas sobre campos finitos $GF(2^m)$

Como se observó en la sección §3.1.4, los elementos que pertenecen al campo finito $GF(2^m)$ se pueden representar como polinomios (base polinomial), por lo tanto pueden ser

representados por cadenas de bits tomados de los coeficientes del polinomio.

La ecuación que representa una curva elíptica sobre un campo finito $GF(2^m)$, se forma escogiendo los coeficientes a, b de tal manera que éstos pertenezcan a $GF(2^m)$ (con la única restricción de que el coeficiente b sea distinto de 0). Por tanto la ecuación que define una curva elíptica sobre un campo finito $GF(2^m)$ de característica 2 queda de la siguiente forma:

$$y^2 + xy = x^3 + ax^2 + b \tag{3.22}$$

Formalmente se puede definir:

Definición 5 *Sea E una curva elíptica sobre un campo finito binario $GF(2^m)$ de la forma: $y^2 + xy = x^3 + ax^2 + b$ (a), donde $a, b \in GF(2^m)$, $b \neq 0$. Un punto P de coordenadas (x, y), pertenece a E si satisface la ecuación (a), además del punto en el infinito, que se simboliza como ϑ.*

Al igual que en el caso de las curvas elípticas definida sobre el campo de los reales, el grupo de curvas elípticas definidas sobre un campo finito binario describen un grupo abeliano, el cual define una operación principal: la suma. En lo que resta del capítulo se describen cómo están definidas estas operaciones sobre dicho grupo de curvas elípticas.

3.4.1. Leyes de grupo de una curva elíptica

1. *Identidad.* $P + \vartheta = \vartheta + P = P$, para todo $P \in GF(2^m)$.

2. *Negativo.* Si $P = (x, y) \in GF(2^m)$, entonces $(x, y)+(x, x+y) = \vartheta$. El punto es denotado por $-P = (x, x + y)$ y se llama el negativo de P.

3. *Suma de puntos.* Sea $P = (x_1, y_1) \in E(GF(2^m))$ y $Q = (x_2, y_2) \in E(GF(2^m))$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{3.23}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{3.24}$$

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \tag{3.25}$$

4. *Doblado de punto.* Sea $P = (x_1, y_1) \in GF(2^m)$, donde $P \neq P$. Entonces $2P = (x_3, y_3)$, donde:

$$x_3 = \lambda^2 + \lambda + a \tag{3.26}$$

$$y_3 = x_1^2 + \lambda x_3 + x_3 \tag{3.27}$$

$$\lambda = x_1 + \frac{y_1}{x_1} \tag{3.28}$$

Como ejemplo, considere el campo finito binario $GF(2^4)$ representado por el polinomio irreducible $p(z) = z^4+z+1$. Un elemento arbitrario $a_3z^3+a_2z^2+a_1z+a_0 \in GF(2^4)$ puede ser representado por una cadena de bits (a_3, a_2, a_1, a_0) de longitud cuatro, por ejemplo la cadena binaria (0101) representa el polinomio z^2+1. Ahora sean los coeficientes $a = z^3, b = z^3+1$, y sea la curva elíptica definida por la ecuación

$$y^2 + xy = x^3 + z^3x^2 + (z^3 + 1) \tag{3.29}$$

la cual se encuentra definida en el campo $GF(2^4)$. Los puntos pertenecientes a dicha curva son:

$(0000, 1011)$	$(0101, 0000)$	$(1001, 1111)$
$(0001, 0000)$	$(0101, 0101)$	$(1011, 0010)$
$(0001, 0001)$	$(0111, 1011)$	$(1011, 1001)$
$(0010, 1101)$	$(0111, 1011)$	$(1100, 0000)$
$(0010, 1111)$	$(1000, 0001)$	$(1100, 1100)$
$(0011, 1100)$	$(1000, 1001)$	$(1111, 0100)$
$(0011, 1111)$	$(1001, 0110)$	$(1111, 1011)$

así como el punto en el infinito ϑ.

La gráfica que representa la curva 3.29 se muestra en la figura 3.5, que como se puede observar, al ser una curva elíptica de un campo finito, no tiene una forma definida como en el caso de de las curvas elípticas en los reales.

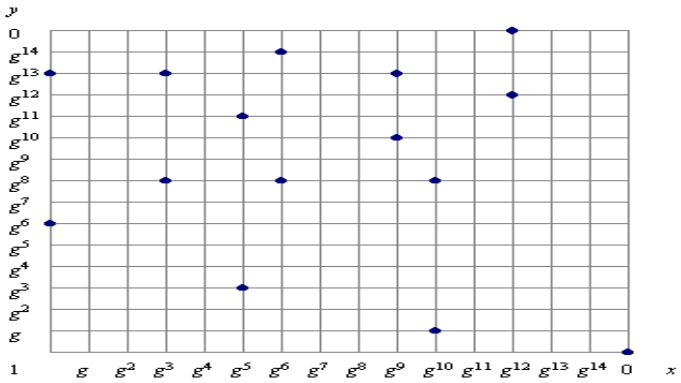

Figura 3.5: Curva $y^2 + xy = x^3 + z^3x^2 + (z^3 + 1)$.

Empleando las ecuaciones 3.25 para suma de puntos elípticos se puede comprobar que $(0010, 1111) + (1100, 1100) = (0001, 0001)$, de igual forma se pueden comprobar para el doblado de punto $2(0010, 1111) = (1011, 0010)$.

3.4.2. Representación de un punto elíptico

La multiplicación polinomial tiene un menor costo computacional a comparación de la inversión en campos finitos binarios, esto es la inversión es la operación aritmética en campos finitos binarios que tiene un mayor tiempo de ejecución. Al ser la inversión una operación costosa en tiempo, repercute en el tiempo de ejecución para realizar operaciones aritméticas sobre curva elípticas. Existe un método que permite suprimir la inversión para realizar operaciones sobre curvas elípticas, que es el emplear coordenadas proyectivas en lugar de coordenadas afines.

Un punto P en coordenadas proyectivas es representado usando una tripleta de coordenadas $(X : Y : Z)$. Este tipo de representación permite reducir el tiempo de ejecución de las operaciones aritméticas de suma y doblado de puntos en curvas elípticas. Normalmente se pasa el punto P en coordenadas proyectivas de vuelta a coordenadas afines al final de toda la ejecución. Esto debido a que el empleo de coordenadas afines sólo requiere de dos coordenadas (en lugar de tres que son las que se emplean en coordenadas proyectivas), lo que permite reducir las comunicaciones con dispositivos externos así reduciendo el ancho de bando de comunicación.

En coordenadas proyectivas estándar el punto $(X : Y : Z)$ donde $Z \neq 0$ corresponde a las coordenadas afines $x = X/Z$ y $y = Y/Z$. La ecuación de curva elíptica en coordenadas proyectivas está dada por:

$$Y^2Z + XYZ = X^3 + aX^2Z + bZ^3 \tag{3.30}$$

El punto en el infinito ϑ, corresponde a $(0 : 1 : 0)$, mientras que el negativo de un punto con coordenadas $(X : Y : Z)$ corresponde a $(X : X + Y, Z)$.

Sin embargo López y Dahab [13], proponen una variedad diferente de coordenadas proyectivas donde se realizan menos operaciones aritméticas. En este nuevo sistema de coordenadas el punto P con coordenadas proyectivas $(X : Y : Z)$, con $Z \not/ 0$ corresponde en coordenadas afines al punto $(X/Z, Y/Z^2)$, el cual corresponde a la curva elíptica con coordenadas proyectivas a la ecuación:

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{3.31}$$

El punto en el infinito ϑ, corresponde a $(1 : 0 : 0)$, mientras que el negativo de un punto con coordenadas $(X : Y : Z)$ corresponde a $(X : X + Y, Z)$.

Y las formulas para calcular las coordenadas $(X_3 : Y_3 : Z_3)$ que son el doble de un punto $(X_1 : Y_1 : Z_1)$ son:

$$\begin{array}{rcl} Z_3 & = & {X_1}^2 \cdot {Z_1}^2; X_3 = {X_1}^4 + b \cdot {Z_1}^4; \\ Y_3 & = & b{Z_1}^4 Z_3 + X_3 \cdot (aZ_3 + {Y_1}^2 + b{Z_1}^4) \end{array} \tag{3.32}$$

Y para calcular las coordenadas $(X_3 : Y_3 : Z_3)$ que es la suma de dos puntos con coordenadas $(X_1 : Y_1 : Z_1)$ y $(X_2 : y_2 : Z_2)$, se obtienen las siguientes ecuaciones:

$$\begin{array}{rclcrcl} A & = & Y_2 \cdot Z_1^2 + Y_1; & & B & = & X_2 \cdot Z_1 + X_1; \\ C & = & Z_1 \cdot B; & & D & = & B^2 \cdot (C + aZ_1^2); \\ Z_3 & = & C^2; & & E & = & A \cdot C; \\ X_3 & = & A^2 + D + E; & & F & = & X_3 + X_2 \cdot Z_3; \\ G & = & (X_2 + Y_2) \cdot Z_3^2; & & Y_3 & = & (E + Z_3) \cdot F + G \end{array} \tag{3.33}$$

3.4.3. Representación de un escalar

Si $P = (x, y) \in GF(2^m)$ entonces $-P = (x, x + y)$, y por tanto $-P = (x, -y)$ si $GF(2^m)$ tiene característica > 3. Se puede considerar que la substracción de puntos de una curva elíptica es tan eficiente como la suma de puntos elípticos. Esto motiva a emplear una representación numérica con signo del escalar $k = \sum_{i=0}^{l-1} k_i 2^i$, donde $k_i \in [0, \pm 1]$. Esta representación numérica con signo permite definir la forma no adyacente (NAF) de un escalar.

Definición 6 *La forma no adyacente (NAF) de un entero positivo k es una expresión de la forma* $k = \sum_{i=0}^{l-1} k_i 2^i$*, donde* $k_i \in [0, \pm 1]$*,* $k_{l-1} \neq 0$*, donde dos dígitos consecutivos son no cero. El tamaño de NAF es l.*

Si existe la posibilidad de almacenar elementos, se puede reducir el tiempo de ejecución del algoritmo empleando un método de ventana el cual calcula w dígitos del escalar k a la vez. Por lo que se define una ventana NAF (w-NAF) como:

Definición 7 *Sea* $w > 2$ *un entero positivo. La longitud w del NAF de un entero positivo k puede ser expresado como* $k = \sum_{i=0}^{l-1} k_i 2^i$*, donde cada coeficiente no cero* k_i *es impar,* $|k_i| < 2^{w-1}$*,* $k_{l-1} \neq 0$*, y existe a lo más alguno de los w dígitos consecutivos es no cero. El tamaño de w-NAF es l.*

El algoritmo que se muestra en la tabla 3.1, es el algoritmo que se emplea para calcular la ventana w-NAF de un entero k.

3.4.4. Aritmética de curvas elípticas

Una vez definidas los operaciones aritméticas que se pueden realizar sobre campos finitos binarios (ver sección §3.2), es posible definir tres operaciones esenciales que se pueden realizar sobre curvas elípticas: doblado de puntos de elípticos, suma de puntos elípticos y bisección de puntos elípticos.

Algoritmo w-NAF.
Entradas: longitud w de la ventana, un entero positivo k
Salida: $NAF_w(k)$

1. $i \leftarrow 0$.
2. Mientras $k \geq 1$ hacer
   a) Si k es impar entonces: $k_i \leftarrow k \bmod 2^w, k \leftarrow k - k_i$.
   b) Si no: $k_i = 0$.
   c) $k \leftarrow k/2, i \leftarrow i + 1$.
3. Regresa $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$.

Tabla 3.1: Algoritmo para calcular la expansión w-NAF.

3.4.4.1. Doblado de puntos elípticos

El doblado de un punto se define como: dado $P = (x, y)$ hay que calcular $2P = (u, v)$, donde:

$$u = \lambda^2 + \lambda + a \tag{3.34}$$
$$v = x^2 + u(\lambda + 1) \tag{3.35}$$

Y λ se calcular a partir de:

$$\lambda = x + y/x. \tag{3.36}$$

Sin embargo de la ecuación (3.36) la operación más costosa es la inversión x/y. Knudsen [1] y Schroeppel [2] proponen sustituir las operaciones de doblado por una nueva operación denominada bisección de punto (half point), donde se elimina la operación de inversión.

3.4.4.2. Bisección de punto

Se puede considerar la bisección de punto como una operación inversa del doblado de puntos.

La bisección de un punto se define de la siguiente manera: dado $2P = (u, v)$ hay que encontrar P tal que $P = (x, y)$, donde:

$$x^2 = v + u(\lambda + 1) \tag{3.37}$$

$$y = \lambda x + x^2 \tag{3.38}$$

La ecuación (3.37) se obtiene a partir de la ecuación (3.35), mientras que la ecuación (3.38) se obtiene a partir de la ecuación (3.36)

Y despejando λ de la ecuación (3.34) tenemos que:

$$\lambda^2 + \lambda = u + a \tag{3.39}$$

La tabla 3.2 muestra el algoritmo que se emplea para calcular la bisección de un punto P, donde se emplean cuatro operaciones aritméticas para poder realizar la bisección de puntos elípticos: revolver una ecuación cuadrática, calcular la traza de un elemento del campo $GF(2^m)$, efectuar una multiplicación en el campo $GF(2^m)$ y finalmente calcular la raíz cuadrada de un elemento arbitrario del campo $GF(2^m)$.

Bisección de punto.

Entradas: representación-λ (u, λ_Q) o coordenadas afines $(u, v) \in GF(2^m)$
Salida: representación-λ (x, λ_P) de $P = (x, y) \in GF(2^m)$, donde $Q = 2P$

1. Encontrar una solución $\hat{\lambda}$ de $\hat{\lambda}^2 + \hat{\lambda} = u + a$.
2. Si la entrada está en representación-λ, entonces calcular $t = u(u + \lambda_Q + \hat{\lambda})$; si no, calcular $t = v + u\hat{\lambda}$.
3. Si $Tr(t) = 0$ entonces $\lambda_P \leftarrow \hat{\lambda}$, $x \leftarrow \sqrt{t + u}$; si no $\lambda_P \leftarrow \hat{\lambda} + 1, x \leftarrow \sqrt{t}$
4. Regresa (x, λ_P)

Tabla 3.2: Algoritmo para calcular la bisección de un punto elíptico

Es conveniente definir la representación-λ [3] de un punto. Dado $Q = (x, y) \in GF(2^m)$ se puede definir (x, λ_Q), donde

$$\lambda_Q = x + \frac{y}{x} \tag{3.40}$$

Dada la representación-λ de Q en la entrada del algoritmo de bisección de punto, se puede calcular la bisección de punto sin tener que transformar a coordenadas afines. En

la multiplicación escalar en curvas elípticas, varias bisecciones de punto consecutivas se pueden calcular directamente en representación-λ, y sólo se transforma de vuelta en coordenadas afines cuando se necesita realizar una suma de puntos elípticos.

3.4.4.3. Suma de puntos elípticos

La suma de puntos elípticos se puede definir como: sea $P = (x_1, y_1) \in GF(2^m)$ y $Q = (x_2, y_2) \in GF(2^m)$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$, donde:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \tag{3.41}$$

$$y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \tag{3.42}$$

3.5. Multiplicación Escalar en Curvas Elípticas

Una vez definidas las operaciones de suma, doblado y bisección de puntos elípticos se puede definir la multiplicación escalar en curvas elípticas.

Y el algoritmo más comúnmente empleado para realizar la multiplicación escalar de curvas elípticas es el algoritmo de suma y doblado (double-and-add) de puntos elípticos el cual es descrito en la tabla 3.3. Cabe señalar que este algoritmo se basa en la idea de que para multiplicar k veces el punto P, se toma la representación binaria del escalar k y se realiza un barrido de izquierda a derecha de tal manera que a cada paso se dobla el punto; y si el bit actual k_i es 1, entonces se suma con el punto actual, por ejemplo:

$$15P = 2P + P \rightarrow 6P + P \rightarrow 14P + P \rightarrow 15P$$

Formalmente se define la operación de multiplicación escalar en curvas elípticas como:

Definición 8 *Sea E una curva elíptica sobre un campo finito binario $GF(2^m)$, sea $Q, P \in GF(2^m)$ y k un entero positivo; se define la multiplicación escalar en curvas en curva elípticas como: $Q = kP = \underbrace{P + P + \cdots + P}_{k \text{ veces}}$.*

Sin embargo se propuso un nuevo algoritmo (ver tabla 3.4) para multiplicación escalar de curvas elípticas, que es el algoritmo de suma y bisección, donde se emplea una ventana de longitud w, y que realiza un menor número de operaciones que el algoritmo de suma y doblado.

Suma y doblado.
Entrada: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(GF(2^m))$
Salida: kP

1. $Q \leftarrow 0$
2. Para i desde $t-1$ hacia 0 hacer
   a) $Q \leftarrow 2Q$
   b) Si $k_i = 1$ entonces $Q \leftarrow Q + P$
3. Regresa Q

Tabla 3.3: Algoritmo de suma y doblado para la multiplicación escalar en curvas elípticas.

3.6. Sistemas basados en Curvas Elípticas

Sea p un número primo, y sea $GF(p)$ el campo de los enteros módulo p. Una curva elíptica E sobre $GF(p)$ está definida por la ecuación de la forma:

$$y^2 = x^3 + ax + b \tag{3.43}$$

donde $a, b \in GF(p)$. El par de coordenadas (x, y), donde $x, y \in GF(p)$, es un punto sobre la curva si satisface la ecuación 3.43. El punto en el infinito, denotado por ϑ, también pertenece a la curva. El conjunto de todos los puntos sobre E se denota como $E(GF(p))$.

Dentro de este grupo hay métodos que permiten sumar dos puntos de la curva elíptica con coordenadas (x_1, y_1) y (x_2, y_2), que dan como resultado un punto de la curva con coordenadas (x_3, y_3). Para poder realizar la suma de puntos elípticos es necesario realizar otras operaciones aritméticas (sumas, restas, multiplicaciones y divisiones) en $GF(p)$ con las coordenadas x_1, y_1, x_2, y_2. Con esta ley de suma, el conjunto de puntos $E(GF(p))$ forman un grupo aditivo (abeliano) con el elemento ϑ haciendo las veces del elemento identidad. Los subgrupos cíclicos formados a partir de dichos grupos de curvas elípticas pueden ser empleados para implementar sistemas basados en DLP.

Este tipo de problema también es conocido como el problema del logaritmo discreto en curvas elípticas (ECDLP).

Problema ECDLP: *Sea E una curva elíptica definida sobre un campo finito $GF(p)$ un punto sobre E de orden n (número primo y grande). El ECDLP es, dados E, P, y un múltiplo escalar Q de P, determinar un entero k tal que $Q = kP$.*

Método de suma y bisección para multiplicación escalar en curvas elípticas.
Entradas: Longitud w de la ventana, $NAF_w = (2^{t-1}kmodn) = \sum_{i=0}^{t} k'_i 2^i$,
$P \in GF(2^m)$
Salida: kP. (Note: $k = k'_0/2^{t-1} + \cdots + k'_{t-1} + 2k'_t modn$)

1. Iniciar $Q_i \leftarrow 0$ para $i \in I = 1, 3, \cdots, 2^{w-1} - 1$
2. Si $k'_t = 1$ entonces $Q_1 = 2P$
3. Para i desde $t - 1$ hacia 0 hacer:
   a) Si $k'_i > 0$ entonces $Q_{k'_i} \leftarrow Q_{k'_i} + P$
   b) Si $k'_i < 0$ entonces $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$
   c) $P \leftarrow P/2$
4. $Q \leftarrow \sum_{i \in I} iQ_i$ 5. Regresar (Q)

Tabla 3.4: Algoritmo de suma y bisección para la multiplicación escalar en curvas elípticas.

3.6.1. Generación de llaves

Los parámetros de dominio de un esquema de curva elíptica, definen una curva elíptica E, que esta definida sobre un campo finito $GF(p)$, el cual tiene un punto base $P \in E(GF(p))$, y un orden n. Estos parámetros deben de ser escogidos de tal forma que el ECDLP sea resistente a todo tipo de ataques. Típicamente los parámetros de dominio son conocidos por ambas entidades que desean comunicarse, sin embargo en algunos casos es necesario especificarlos para cada usuario.

Los parámetros que son empleados para un criptosistema basado en curvas elípticas son $D = (q, FR, S, a, b, P, n, h)$:

- El orden del campo q.
- La representación empleada para los elementos del campo finito $GF(p)$, y que se denota por FR.
- Una semilla S, si la curva elíptica fue generada aleatoriamente.
- Dos coeficientes denotados como $a, b \in GF(p)$, los cuales definen la ecuación de la curva elíptica E sobre $GF(p)$, es decir $y^2 = x^3 + ax + b$ (en el caso de un campo finito

primo); y $y^2 + xy = x^3 + ax^2 + b$ (en el caso de un campo finito binario).

- Dos elementos del campo denotados $x_p, y_p \in GF(p)$, los cuales definen el punto $P = (x_p, y_p) \in E(GF(p))$ en coordenadas afines. P es de orden primo y se denomina como punto base de la curva elíptica.
- El orden n de P.
- El cofactor $h = \#E(GF(p))/n$.

Una vez establecidos estos parámetros de dominio, la llave pública es escogida al azar, es decir se escoge un punto Q aleatoriamente a partir del punto base P. Su correspondiente llave privada es $d = log_p Q$. La tabla 3.5 muestra como son generados el par de llaves.

ECC, generación de llaves.
Entradas: Parámetros públicos (q, FR, Sa, b, P, n, h). Salida: Llave pública Q y llave privada d.

1. Seleccionar $d \in R_{[1,n-1]}$.
2. Calcular $Q = dP$.
3. Devolver (Q, d).

Tabla 3.5: Generación de llaves en ECC.

Cabe señalar que la dificultad de encontrar la llave privada d a partir de la llave pública Q, consiste precisamente en resolver el ECDLP. Por lo que es crucial que los parámetros de dominio D sean escogidos de tal forma que el ECDLP sea intratable. Además, es importante que el número d, sea escogido aleatoriamente, ya que así se reduce la probabilidad de que un atacante escoja un valor en particular a partir del cual pueda realizar un búsqueda de la llave privada correcta.

3.6.2. ECDSA

El algoritmo de firma digital con curvas elípticas (ECDSA) es análogo al algoritmo de firma digital (DSA). ECDSA es en la actualidad un estándar para un criptosistema basado en curvas elípticas, el cual forma parte de los siguientes estándares: ANSI X9.62 [10], FIPS 186-2[9], IEEE 1363 [12] y ISO/IEC 15946-2 [11].

Los algoritmos mostrados en 3.6 y 3.7 muestran como se realiza la firma digital y la verificación.

ECDSA, generación de firma digital.

Entradas: Parámetros públicos $D = (q, FR, S, a, b, P, n, h)$, llave privada x, mensaje m.
Salida: Firma (r, s).

1. Escoger k tal que $k \in R_{[1,n-1]}$.
2. Calcular $kP = (x_1, y_1)$, y convertir x_1 en un entero $\overline{x}_1$.
3. Calcular $r = \overline{x}_1 \bmod n$. Si $r = 0$, entonces regresar al paso **1**.
4. Calcular $e = H(m)$.
5. Calcular $s = k^{-1}(e + dr) \bmod n$. Si $s = 0$, entonces regresar al paso **1**.
6. Devolver (r, s).

Tabla 3.6: Firma digital empleando DSA.

Para probar que la verificación funciona, suponga que una firma (r, s) de un mensaje m es en verdad generada por quien envía el mensaje, entonces $s \equiv k^{-1}(e + dr) \bmod n$. Reordenando se tiene que:

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d (\bmod n). \tag{3.44}$$

por lo que $X = u_1P + u_2Q = (u_1 + u_2d)P = kP$, y por lo tanto $v = r$ es válido.

ECDSA, verificación de firma digital.

Entradas: Parámetros públicos $D = (q, FR, S, a, b, P, n, h)$, llave pública Q, mensaje m, firma (r, s).
Salida: Aceptación o denegación de la firma.

1. Verificar que r, s sean dos enteros dentro del intervalo $[1, n-1]$. Si la verificación falla, entonces la firma no es válida.
2. Calcular $e = H(m)$.
3. Calcular $w = s^{-1} \bmod n$.
4. Calcular $u_1 = ew \bmod n$ y $u_2 = rw \bmod n$.
5. Calcular $X = u_1 P + u_2 Q$.
6. Si $X = \vartheta$, entonces firma inválida.
7. Convertir la coordenada x_1 de X en un entero $\overline{x}_1$; y calcular $v = \overline{x}_1 \bmod n$.
8. Si $v = r$, entonces la firma es válida, en caso contrario la firma es inválida.

Tabla 3.7: Verificación de firma empleando ECDSA.

Capítulo 4

Unidad Aritmético Lógica

En capítulo §3, se hizo mención sobre los aspectos matemáticos sobre los cuales esta basado la firma digital en curvas elípticas. Además, se mostraron tanto las operaciones aritméticas sobre campos finitos binarios, como las operaciones aritméticas sobre curvas elípticas que se deben de efectuar para poder realizar dicha acción.

Una vez conocido cómo realizar estas operaciones aritméticas, se da a continuación la explicación de cómo se implementaron cada unas de estas operaciones aritméticas bajo una plataforma FPGA. Dada la naturaleza de este tipo de dispositivo se trataron de paralelizar tantas operaciones como fue posible para poder obtener los mejores rendimientos y tiempos de ejecución posibles. Por otro lado también se dan los tiempos obtenidos por simulación mediante la herramienta ModelSim 5.8c de Xilinx.

4.1. Aritmética de campos finitos

Como se hizo mención en el capítulo §1, para realizar las operaciones aritméticas de curvas elípticas, es necesario primero crear una subcapa inferior que consiste de operaciones aritméticas sobre campos finitos (más en especifico sobre campos finitos binarios, los cuales son empleados en el presente trabajo).

Para definir las operaciones aritméticas de curvas elípticas se deben primero definir las operaciones aritméticas sobre campos finitos como: suma, multiplicación, elevar al cuadrado, raíz cuadrada, función traza y función de media traza. Por lo que en primera instancia, se van a describir los módulos creados que calculan estas operaciones aritméticas y que fueron implementados bajo una plataforma FPGA. Cabe mencionar que la aritmética que se utilizó, es la aritmética definida por el campo finito binario $GF(2^{163})$.

4.1.1. Suma

La operación aritmética suma es la operación de menor complejidad en un campo finito binario. La implementación de esta operación en hardware reconfigurable es simple, como se describió en la sección §3.2.1, esta operación consiste en realizar una operación XOR bit a bit entre los operandos a sumar. Al tratarse de un campo $GF(2^{163})$, la suma de dos elementos que pertenezcan a este campo consiste en una operación XOR entre los 163 bits de ambos operandos.

Esta es una de las operaciones más fáciles de implementar, además de que su costo tanto en tiempo de ejecución, como su costo en área es despreciable a comparación de la operación de multiplicación en campos finitos binarios.

4.1.2. Multiplicación

Al tratarse de una de las operaciones más complejas de aritmética de campos finitos binarios (la otra operación es la inversión modular), se adoptó por utilizar una estrategia en paralelo (Karatsuba-Ofman) para calcular esta operación aritmética en el menor tiempo posible.

Se usó el campo finito binario $GF(2^{163})$, que está definido por el polinomio irreducible $p(z) = z^{163} + z^7 + z^6 + z^3 + 1$. Como se explicó en la sección §3.2.2, esta operación se realiza en dos pasos: primero se efectúa una multiplicación polinomial, para en un segundo paso realizar la reducción polinomial.

4.1.2.1. Multiplicación polinomial

Se emplea un multiplicador binario de Karatsuba - Ofman, empleando la estrategia propuesta en [38], se divide el campo finito en dos bloques uno de 128 bits y un bloque de 35 bits, como se muestra en las figuras 4.1 y 4.2

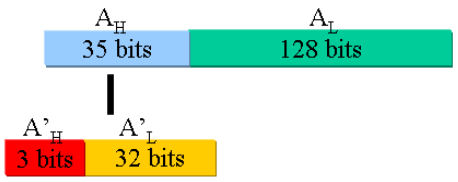

Figura 4.1: Disposición de bits para el polinomio $a(x)$.

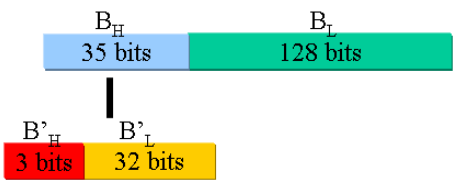

Figura 4.2: Disposición de bits para el polinomio $b(x)$.

Gracias a esta división de los elementos polinomiales, se puede emplear un multiplicador binario Karatsuba-Ofman, el cual está constituido de tres multiplicadores: dos multiplicadores de 128 bits y uno de 35 bits (como se describió en la sección §3.2.3), utilizando la ecuación 3.9 es necesario emplear tres multiplicadores binarios en paralelo, así como es indispensable calcular cuatro sumas binarias.

El multiplicador de 128 bits a su vez esta constituido de tres multiplicadores de 64 bits con el enfoque de Karatsuba-Ofman, y cada uno de ellos su vez están constituidos de tres multiplicadores de 32; y así sucesivamente hasta obtener multiplicadores de 4 bits (ver figura 4.3). Estos últimos multiplicadores son multiplicadores binarios implementados con lógica combinacional.

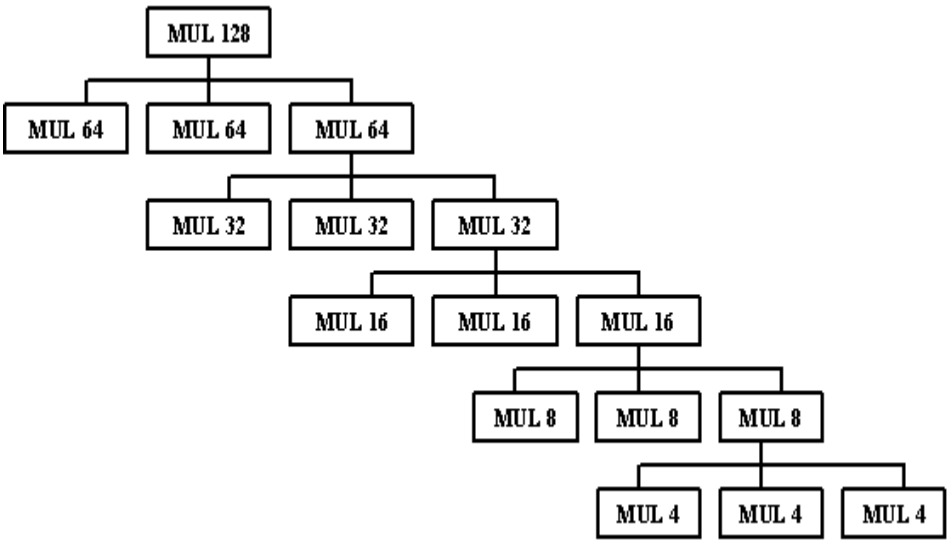

Figura 4.3: Multiplicador binario de Karatsuba de 128 bits.

Por otra parte el multiplicador de 35 bits en lugar de emplear el algoritmo de Karatsuba-Ofman, emplea la ecuación 3.8, en donde se emplean cuatro multiplicadores binarios en paralelo y tres sumas binarias. Al utilizar este enfoque, a pesar de emplear cuatro multiplicadores binarios en paralelo; emplea menos recursos del FPGA ya que estos multiplicadores no son multiplicadores "cuadrados", es decir los operandos no tienen la misma longitud de bits. Es más sencillo en estos casos implementar multiplicadores chicos con lógica combinacional. Así el multiplicador de 35 bits está constituido por un multiplicador de 3 bits, un multiplicador de 32 bits y por dos multiplicadores de 32×3 bits (ver figura 4.4).

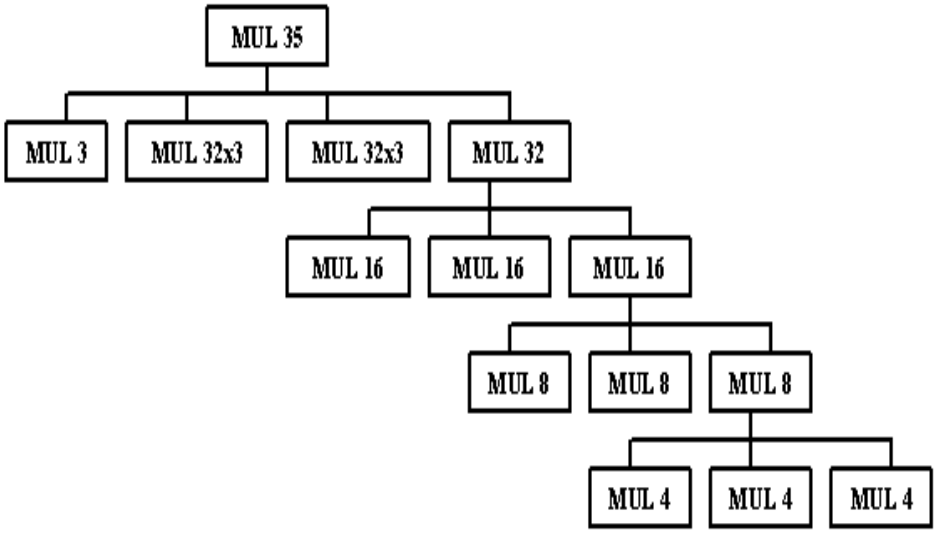

Figura 4.4: Multiplicador binario de Karatsuba de 35 bits.

4.1.2.2. Esquema de reducción

El resultado de una multiplicación entre dos polinomios de grado $m-1$, resulta en un polinomio de grado $2m-2$, sin embargo en aritmética de campos finitos es necesario reducir este polinomio modulo un polinomio irreducible para obtener un polinomio de grado $m-1$. El presente trabajo emplea un campo $GF(2^{163})$, que tiene como polinomio irreducible $p(z) = z^{163} + z^7 + z^6 + z^3 + 1$.

Para realizar la reducción se emplea el esquema mostrado en la figura 4.5, el cual puede ser implementado empleando únicamente compuertas XOR.

Finalmente el multiplicador de 163 bits es mostrado en la figura 4.6, donde se puede observar que el primer multiplicador de 128 bits se encarga de procesar la parte baja del polinomio, es decir calcula $A^L B^L$ (i.e. los primeros 128 bits); el producto $(A^H + A^L)(B^L + B^H)$ lo realiza el segundo multiplicador de 128 bits, y por último el producto $A^H B^H$. Después se pasa a la etapa de traslape donde se realizan las cuatro sumas de la ecuación 3.9.

Usando los pasos anteriormente mencionados se obtiene el circuito que se muestra en la figura 4.6.

La tabla 4.1 muestra el espacio en área como tiempos de ejecución para diversos multiplicadores binarios de Karatsuba-Ofman para una plataforma en hardware, estos resultados fueron presentados por F. Rodríguez-Henríquez y C. Koc [4].

Este módulo de multiplicación polinomial, es la base del circuito final. Debido a la

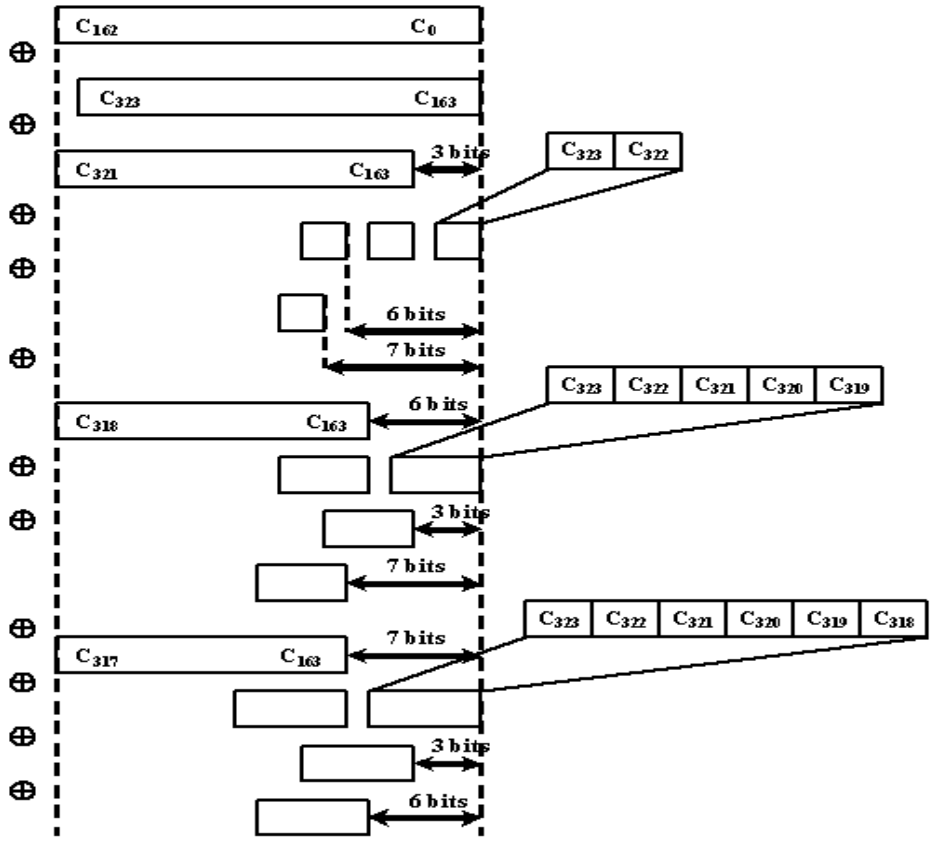

Figura 4.5: Reducción pentanomial.

complejidad para realizar esta multiplicación polinomial es el módulo que emplea más recursos (tanto en tiempo como en área). Se va a demostrar que las operaciones restantes consumen menos recursos del FPGA a comparación de la multiplicación polinomial.

4.1.3. Elevar al cuadrado

Para calcular A^2, donde A es un elemento arbitrario de $GF(2^{163})$, se considera la representación polinomial de $A = a(z)$ como

$$a(z) = \sum_{i=0}^{162} a_i z^i, a_i \in 0, 1. \tag{4.1}$$

La forma más directa de implementar esta operación aritmética, es empleando la defini-

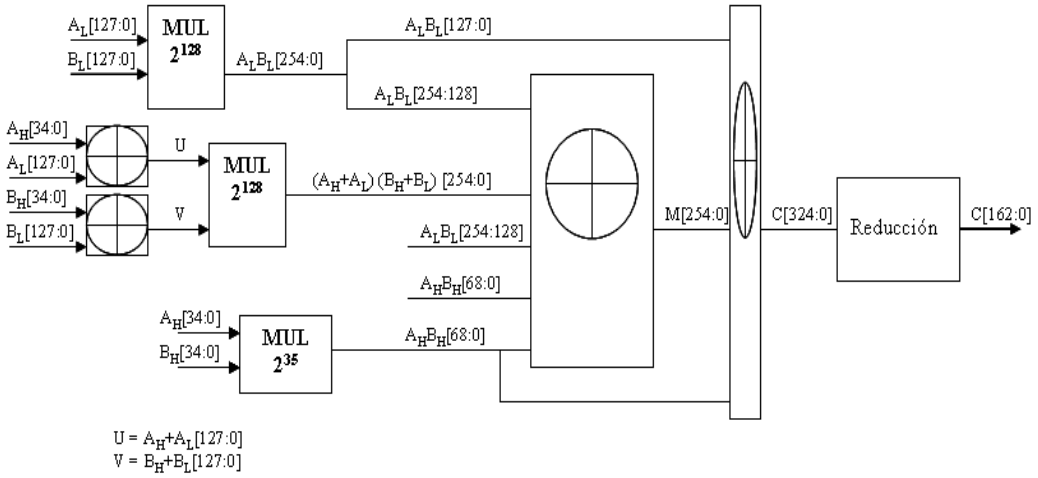

Figura 4.6: Circuito multiplicador binario de 163 bits.

ción de multiplicación polinomial que consiste primeramente en realizar el producto $c'(z) = a(z) \cdot a(z)$, para en un segundo paso aplicar la reducción modular ($c(z) = c'(z)$ mód $p(z)$) ocupando un polinomio irreducible. Usando esta técnica se emplearían los módulos de multiplicación polinomial y reducción polinomial descritos en §4.1.4.

No obstante si se desarrolla dicho producto polinomial como en la figura 4.7, se observa que sólo los elementos cuyos índices son pares del elemento resultante del producto polinomial, es decir $c_8 = a_4, c_6 = a_3, c_4 = a_2, c_2 = a_1, c_0 = a_0$, tienen un valor entero, los elementos cuyos índices son impares por el contrario tienen un valor nulo.

						$a_4a_3a_2a_1a_0$		
						$a_4a_3a_2a_1a_0$		
				a_4a_0	a_3a_0	a_2a_0	a_1a_0	a_0a_0
			a_4a_1	a_3a_1	a_2a_1	a_1a_1	a_0a_1	
		a_4a_2	a_3a_2	a_2a_2	a_1a_2	a_0a_2		
	a_4a_3	a_3a_3	a_2a_3	a_1a_3	a_0a_3			
a_4a_4	a_3a_4	a_2a_4	a_1a_4	a_0a_4				
a_4	0	a_3	0	a_2	0	a_1	0	a_0

Figura 4.7: Ejemplo de producto polinomial de un elemento aleatorio que pertenece al campo $GF(2^5)$

m	AND's	XOR's	Tiempo (ns)
3	9	4	7.824
4	16	9	7.846
8	48	55	9.990
16	144	225	12.179
32	432	799	13.434
35	633	827	14.448
64	1,296	2,649	15.554
128	3888	8,455	16.856
163	8,409	17,837	20.272
256	11,664	26,385	n/a

Tabla 4.1: Tabla de resultados (área y tiempo) para diferentes multiplicadores binarios de Karatsuaba-Ofman (tomado de [4]).

Por lo que un elemento arbitrario que pertenece al campo $GF(2^m)$ elevado al cuadrado se obtiene mediante la operación:

$$c(z) = a(z)a(z) = a^2(z) = \left(\sum_{i=0}^{m-1} a_i z^i\right)\left(\sum_{i=0}^{m-1} a_i z^i\right) = \sum_{i=0}^{m-1} a_i z^{2i} \bmod p(z) \tag{4.2}$$

Para implementar en hardware esta ecuación, basta con solo insertar ceros entre cada bit del elemento de entrada, el elemento resultante sería un polinomio de 326 bits, por lo que hay que aplicar reducción modular.

Sin embargo, en la sección §3.2.4 se describe otro forma para poder obtener el cuadrado de un elemento arbitrario de $GF(2^{163})$. Para calcular A tal que $C = A^2 \in GF(2^{163})$ se tiene que:

$$C = \begin{bmatrix} c_0 \\ \vdots \\ c_i \\ \vdots \\ c_{162} \end{bmatrix} = A^2 = MA = \begin{bmatrix} \\ 163 \times 163 \\ \\ \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{162} \end{bmatrix} \tag{4.3}$$

Si se puede encontrar la matriz M de la ecuación 4.3 de tal forma que al multiplicar la matriz M por un elemento aleatorio $A \in GF(2^{163})$, se encuentra el elemento C tal que $C = A^2$.

Para obtener esta matriz M, primero se toma la representación en base polinomial de $A \in GF(2^{163})$, luego se realiza la operación $C = A \cdot A \bmod p(x)$, y se obtienen un conjunto de ecuaciones para cada coeficiente del polinomio resultante, para el caso el campo $GF(2^{163})$ se obtienen las siguientes ecuaciones:

$$
\begin{aligned}
a_0 &= c_0 \oplus c_{160} \\
a_1 &= c_{82} \oplus c_{160} \oplus c_{162} \\
a_2 &= c_1 \oplus c_{161} \\
a_3 &= c_{83} \oplus c_{160} \oplus c_{161} \\
a_4 &= c_2 \oplus c_{82} \oplus c_{160} \\
a_5 &= c_{84} \oplus c_{161} \oplus c_{162} \\
a_6 &= c_3 \oplus c_{83} \oplus c_{160} \oplus c_{161} \\
a_7 &= c_{82} \oplus c_{85} \\
a_8 &= c_4 \oplus c_{82} \oplus c_{84} \oplus c_{160} \oplus c_{161} \\
a_9 &= c_{83} \oplus c_{86} \\
a_{10} &= c_5 \oplus c_{83} \oplus c_{85} \oplus c_{161} \oplus c_{162} \\
a_{11} &= c_{84} \oplus c_{87} \\
a_{12} &= c_6 \oplus c_{84} \oplus c_{86} \oplus c_{162} \\
a_{13} &= c_{85} \oplus c_{88} \\
a_{14} &= c_7 \oplus c_{85} \oplus c_{87} \\
a_{15+i} &= c_{86+i} \oplus c_{89+i}, \text{ para } i \text{ impar tal que } i \in [0, 147] \\
a_{16+i} &= c_{8+i} \oplus c_{86+i} \oplus c_{88+i}, \text{ para } i \text{ par tal que } i \in [0, 147]
\end{aligned}
$$

Tabla 4.2: Ecuaciones para calcular el cuadrado de un elemento arbitrario de $GF(2^{163})$, con polinomio irreducible $p(z) = z^{163} + z^7 + z^6 + z^3 + 1$.

De esta forma se obtiene una forma que es fácil de implementar en hardware, ya que solamente se emplean compuertas XOR para calcular los elementos c_i. Se puede apreciar que para cada coeficiente a_i se tiene a lo más cinco operandos, y se tienen cuando menos dos operandos por ecuación. Tomando en cuenta las ecuaciones mostradas en la tabla 4.2 se tienen un total de 256 compuertas XOR para calcular la operación de elevar al cuadrado.

4.1.4. Raíz cuadrada

Para realizar la operación de raíz cuadrada, se emplea el método descrito en §3.2.5. Para extraer la raíz cuadrada de un elemento de $GF(2^{163})$, se considera un elemento arbitrario $C \in GF(2^{163})$, en base polinomial como en le ecuación 4.1. Como se observó en la sección previa se puede obtener el cuadrado de un elemento arbitrario de $GF(2^{163})$ mediante la ecuación 4.3.

Por lo que de una manera análoga, dado A, se puede calcular $C = \sqrt{A}$ resolviendo la siguiente ecuación,

$$C = \begin{bmatrix} c_0 \\ \vdots \\ c_i \\ \vdots \\ c_{162} \end{bmatrix} = M^{-1}A = \begin{bmatrix} \\ \quad 163 \times 163 \quad \\ \\ \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{162} \end{bmatrix} \tag{4.4}$$

Así, para calcular $C = \sqrt{A}$ de un elemento $A \in GF(2^m)$, primero se debe de encontrar la matriz inversa M^{-1} (si es que existe) de la matriz M a partir de la ecuación (4.4). Una vez que se halla dicha matriz inversa, se procede a calcular $C = M^{-1}A$. Para el caso de la curva $B - 163$ la matriz M se obtiene a partir de las ecuaciones presentadas en la sección previa §4.1.1, por lo que sólo es necesario encontrar la matriz inversa M^{-1} de M, las ecuaciones obtenidas se pueden ver en el apéndice A. El número de operandos por ecuación es a lo más de 54, y cuando menos las ecuaciones tienen dos operandos.

Por lo que al igual que la operación de elevar al cuadrado, solamente es necesario emplear compuertas XOR para poder calcular la raíz cuadrada de un elemento arbitrario de $GF(2^{163})$. Empleando las ecuaciones del apéndice A se obtiene un total de 8214 compuertas XOR.

4.1.5. Traza

En la sección §3.2.6, se definió la función traza, así como también se dieron las propiedades de dicha función.

Y debido a su propiedad de linealidad, la función traza se puede implementar de tal forma que su tiempo de ejecución sea $O(1)$. Si se toma un elemento aleatorio $C = c(z) \in GF(2^m)$ en base polinomial, se puede redefinir la función traza como:

$$Tr(C) = Tr(\sum_{i=0}^{m-1} c_i z^i) = \sum_{i=0}^{m-1} c_i Tr(z^i) \tag{4.5}$$

Como ejemplo, si se tiene el campo finito $GF(2^{163})$ definido por el polinomio de reducción irreducible $p(z) = z^{163} + z^7 + z^6 + z^3 + 1$, se tiene que $Tr(z^i) = 1$ si y solo si $i \in \{0, 157\}$. Por lo que la implementación de la función traza en hardware reconfigurable sólo necesita una compuerta XOR para suma los bits *0 y 157* del polinomio de entrada.

4.1.6. Media Traza

Para resolver la ecuación cuadrática descrita por la ecuación 4.6,

$$\hat{\lambda}^2 + \hat{\lambda} = u + a = c \tag{4.6}$$

se usa la función de media traza. En la sección §3.2.7 se definió la función de media traza, así como también se dieron a conocer sus propiedades.

Si se define un elemento $C = c(z)$ como $c(z) = \sum_{i=0}^{m-1} c_i z^i \in GF(2^m)$ con $Tr(C) = 0$, y si se emplea la propiedad de linealidad de la función de media traza, se tiene que se puede redefinir la función de media traza ($HT(C)$) como:

$$H(C) = H(\sum_{i=0}^{m-1} c_i z^i) = \sum_{i=0}^{m-1} c_i H(z^i) \tag{4.7}$$

Sin embargo si se emplea el pentanomio irreducible $p(z) = z^{163} + z^7 + z^6 + z^3 + 1$ así como la definición de la media traza dada por la ecuación 4.7. Se pueden precalcular las 163 medias trazas de los elementos z^i pertenecientes al campo finito binario, desde $i = 0, 1, \ldots, 162$; para así poder obtener un conjunto de 163 ecuaciones, una ecuación por cada coeficiente del polinomio resultante. Cada una de estas ecuaciones tiene 82 coeficientes no nulos, por lo que resultan en ecuaciones con muchos coeficientes.

Reagrupando el conjunto de ecuaciones obtenidas dentro de una matriz M cuadrada de 163×163, se puede calcular la media traza de un elemento arbitrario $C \in GF(2^m)$ realizando la multiplicación de matrices $H(C) = CB$, dada por:

$$H(C) = \begin{bmatrix} h_0 & \ldots & h_{162} \end{bmatrix} = MC = \begin{bmatrix} \\ \quad 163 \times 163 \quad \\ \\ \end{bmatrix} \begin{bmatrix} c_0 & \ldots & c_{162} \end{bmatrix} \tag{4.8}$$

Al realizar el producto descrito por la ecuación 4.8, se obtienen el conjunto de ecuaciones, cada una de estas ecuaciones representa un coeficiente h_i del polinomio resultante de la operación de media traza. Las ecuaciones resultantes al ser sumas de los coeficientes c_i del elemento de entrada, se pueden traducir a compuertas XOR en una implementación en hardware reconfigurable. Dentro de este conjunto de ecuaciones, cada ecuación tiene 82 coeficientes no nulos. Por lo que en este caso en necesario empelar $82 \times 163 = 13,366$ compuertas XOR.

4.2. Aritmética de curvas elípticas

Una vez que se tienen definidas las operaciones aritméticas sobre campos finitos binarios, se prosigue a crear las primitivas para poder realizar las tres operaciones de curvas elípticas: suma, doblado y bisección de puntos elípticos. Ya que a partir de estas operaciones que se puede calcular la multiplicación escalar en curvas elípticas kP.

En esta sección se explica como se conjuntaron las operaciones aritméticas en campos finitos para poder crear una arquitectura que permita, realizar operaciones de curvas elípticas de una manera rápida y eficiente.

4.2.1. Doblado de un punto en curvas elípticas

El doblado de un punto en curvas elípticas es una operación que es requerida por la operación de suma de puntos elípticos, en el caso de que se sume un punto elíptico con sigo mismo. Para poder realizar esta operación se emplean coordenadas mixtas (coordenadas afines y coordenadas proyectivas LD -propuestas por López y Dahab [13] -). El algoritmo que es presentado se trata de una implementación en software.

Se puede apreciar que el algoritmo mostrado en la tabla 4.2.1, consta de quince pasos para poder realizar una operación de doblado de un punto, requiere de cinco operaciones de elevar el cuadrado, de cuatro sumas polinomiales y de cuatro multiplicaciones polinomiales.

Sin embargo en una implementación en hardware se pueden reducir el número de pasos para poder realizar dicha operación. El algoritmo mostrado en la tabla 4.4, es un algoritmo que consta de cuatro pasos para poder ejecutar el doblado de un punto. Estos cuatro pasos son logrados gracias a que una implementación en hardware las operaciones de suma y elevar al cuadrado no emplean muchos recursos (compuertas XOR); por lo que la única operación costosa en recursos es la multiplicación (puesto que se emplea un multiplicador binario de Karatsuba-Ofman). Así se pueden agrupar hasta cuatro operaciones por ciclo.

Se observa que gracias a estos cuatros ciclos siempre se emplea el multiplicador binario de Karatsuba-Ofman, por lo que no se desperdician ni recursos ni ciclos de reloj. Cabe añadir que case las operaciones que se realizar en esos cuatros ciclos son de la forma:

$$oper1 \cdot oper2 \tag{4.9}$$

siendo la operación del ciclo 1 ($Z_3 = X_1^2 \cdot Z_1^2$) la operación más compleja. O también se tiene operación de la forma:

$$oper1 \cdot oper2 + oper3 \tag{4.10}$$

doblado de un punto en curvas elípticas ($y^2 + xy = x^3 + ax^2 + b, a \in \{0, 1\}$), coordenadas LD-afines.
Entradas: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD sobre $E/K : y^2 + xy = x^3 + ax^2 + b$.
Salida: $2P = (X_3 : Y_3 : Z_3)$ en coordenadas LD.

1. Si $P = 0$ entonces regresa ϑ
2. $T_1 \leftarrow Z_1^2$ $\{T_1 \leftarrow Z_1^2\}$
3. $T_2 \leftarrow X_1^2$ $\{T_2 \leftarrow X_1^2\}$
4. $Z_3 \leftarrow T_1 \cdot T_2$ $\{z_3 \leftarrow X_1^2 Z_1^2)\}$
5. $X_3 \leftarrow T_2^2$ $\{X_3 \leftarrow X_1^2\}$
6. $T_1 \leftarrow T_1^2$ $\{T_1 \leftarrow X_2^4\}$
7. $T_2 \leftarrow T_1 \cdot b$ $\{T_2 \leftarrow bZ_1^4\}$
8. $X_3 \leftarrow X_3 + T_2$ $\{X_3 \leftarrow X_1^4 + bZ_1^4\}$
9. $T_1 \leftarrow Y_1^2$ $\{T_1 \leftarrow Y_1^2\}$
10. Si $a = 1$ entonces $T_1 \leftarrow T_1 + Z_3$ $\{T_1 \leftarrow aZ_3 + Y_1^2\}$
11. $T_1 \leftarrow T_1 + T_2$ $\{T_1 \leftarrow aZ_3 + Y_1^2 + bZ_1^4\}$
12. $Y_3 \leftarrow X_3 \cdot T_1$ $\{Y_3 \leftarrow X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
13. $T_1 \leftarrow T_2 \cdot Z_3$ $\{T_1 \leftarrow bZ_1^4 Z_3\}$
14. $Y_3 \leftarrow Y_3 + T_1$ $\{Y_3 \leftarrow bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
15. Regresa $(X_3 : Y_3 : Z_3)$

Tabla 4.3: Algoritmo de López-Dahab para doblado de puntos elípticos.

donde la operación del ciclo 3 ($T_1 = X_3 \cdot (Z_3 + Y_1^2 + T_2)$), es la operación más compleja.

Además gracias al paralelismo empleado, se pueden obtener hasta dos cálculos por ciclo.

4.2.2. Suma de puntos

La operación de aritmética en curvas elípticas que emplea mayor número de operaciones sobre campos finitos binarios es la suma de puntos elípticos. Sobre todo si se emplean coordenadas mixtas (afines-proyectivas), este algoritmo fue propuesto por López y Dahab [**?**, LOPEZ99] De los algoritmos que emplean coordenadas proyectivas el de López y Dahab, es el que realiza menos operaciones de multiplicación, por que es un algoritmo más efectivo tanto en software como en hardware. La tabla 4.2.1 muestra dicho algoritmo.

Se observa que el algoritmo mostrado por la tabla 4.2.1 consta de veinte y seis pa-

Algoritmo paralelo para doblado de un punto, coordenadas LD-afines.

Entradas: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD sobre $E/K : y^2 + xy = x^3 + ax^2 + b$.

Salida: $2P = (X_3 : Y_3 : Z_3)$ en coordenadas LD

# ciclo	C_0	C_1
1. ciclo:	$Z_3 = X_1^2 \cdot Z_1^2$	$T_1 = Z_1^4$
2. ciclo:	$X_3 = b \cdot T_1 + X_1^4$	$T_2 = b \cdot T_1$
3. ciclo:	$T_1 = X_3 \cdot (Z_3 + Y_1^2 + T_2)$	
4. ciclo:	$Y_3 = T_2 \cdot Z_3 + T_1$	

Tabla 4.4: Versión paralela del algoritmo de López-Dahab para el doblado de puntos.

sos (para su implementación en software). Los cuales requieren de cinco operaciones de elevar al cuadrado, de nueve sumas de polinomios y de ocho multiplicaciones polinomiales. A partir de estas ocho multiplicaciones se agrupan las demás operaciones, ya que la multiplicación polinomial emplea mayor tiempo de ejecución. Siguiendo el patrón que se empleó en el doblado de puntos elípticos, se consigue una aproximación paralela del algoritmo de suma de puntos elípticos de López-Dahab (ver tabla 4.6). Esta nueva versión consta de ocho pasos para poder ejecutar la suma de puntos en coordenadas proyectivas, ya que aprovecha al máximo los recursos que proporciona un FPGA, para así realizar una implementación eficiente.

Además al igual que en el algoritmo paralelo de doblado de puntos elípticos 4.4, se obtienen operaciones de la forma

$$oper1 \cdot oper2 \tag{4.11}$$

donde $oper1, oper2$ pueden ser la suma de dos o más operandos. Como ejemplo, en el ciclo 4, la operación más compleja de este tipo ($X_3 = X_3^2 \cdot (Z_1^2 + T_1)$).

También se obtienen operaciones de la forma:

$$oper1 \cdot oper2 + oper3 \tag{4.12}$$

donde $oper1, oper2$ puede ser una composición de dos o más operaciones, siendo en el ciclo 5 la operación más compleja ($X_3 = Y_3 \cdot T_1 + X_3 + Y_3^2$).

De la misma forma que el doblado de puntos, esta versión permite el calculo de una o dos operaciones a la vez.

Suma de puntos elípticos ($y^2 + xy = x^3 + ax^2 + b, a \in \{0, 1\}$)*, coordenadas LD-afines*.

Entradas: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD, $Q = (x_2, y_2)$ en coordenadas afines sobre $E/K : y^2 + xy = x^3 + ax^2 + b$.
Salida: $P + Q = (X_3 : Y_3 : Z_3)$ en coordenadas LD

1. Si $Q = 0$ entonces regresa (P)
2. Si $P = 0$ entonces regresa $x_2 : y_2 : 1$
3. $T_1 \leftarrow Z_1 \cdot x_2$ $\{T_1 \leftarrow X_2 Z_1\}$
4. $T_2 \leftarrow Z_1$ $\{T_2 \leftarrow Z_1^2\}$
5. $X_3 \leftarrow X_1 + T_1$ $\{X_3 \leftarrow B = X_2 Z_1 + X_1\}$
6. $T_1 \leftarrow Z_1 \cdot X_3$ $\{T_1 \leftarrow C = Z_1 B\}$
7. $T_3 \leftarrow T_2 \cdot y_2$ $\{T_3 \leftarrow Y_2 Z_1^2\}$
8. $Y_3 \leftarrow Y_1 + T_3$ $\{Y_3 \leftarrow A = Y_2 Z_1^2 + Y_1\}$
9. Si $X_3 = 0$ entonces
   a) Si $Y_3 = 0$ entonces emplear algoritmo 4.2.1 para calcular $(X_3 : Y_3 : Z_3) = 2(x_2 : y_2 : 1)$ y regresa $(X_3 : Y_3 : Z_3)$
   b) Si no regresa ϑ
10. $Z_3 \leftarrow T_1^2$ $\{Z_3 \leftarrow C^2\}$
11. $T_3 \leftarrow T_1 \cdot Y_3$ $\{T_3 \leftarrow E = AC\}$
12. Si $a = 1$ entonces $T_1 \leftarrow T_1 + T_2$ $\{T_1 \leftarrow C + aZ_1^2\}$
13. $T_2 \leftarrow X_3^2$ $\{T_2 \leftarrow B^2\}$
14. $X_3 \leftarrow T_2 \cdot T_1$ $\{X_3 \leftarrow D = B^2(C + aZ_1^2)\}$
15. $T_2 \leftarrow Y_3^2$ $\{T_2 \leftarrow A^2\}$
16. $X_3 \leftarrow X_3 + T_2$ $\{X_3 \leftarrow A^2 + D\}$
17. $X_3 \leftarrow X_3 + T_3$ $\{X_3 \leftarrow A^2 + D + E\}$
18. $T_2 \leftarrow x_2 \cdot Z_3$ $\{T_2 \leftarrow Z_2 Z_3\}$
19. $T_2 \leftarrow T_2 + X_3$ $\{T_2 \leftarrow F = X_3 + X_2 Z_3\}$
20. $T_1 \leftarrow Z_3^2$ $\{T_1 \leftarrow Z_3^2\}$
21. $T_3 \leftarrow T_3 + Z_3$ $\{Y_3 \leftarrow E + Z_3\}$
22. $Y_3 \leftarrow T_3 \cdot T_2$ $\{Y_3 \leftarrow (E + Z_3)F\}$
23. $T_2 \leftarrow x_2 + y_2$ $\{T_2 \leftarrow X_2 + Y_2\}$
24. $T_3 \leftarrow T_1 \cdot T_2$ $\{T_3 \leftarrow G = (X_2 + Y_2)Z_3^2\}$
25. $Y_3 \leftarrow Y_3 + T_3$ $\{Y_3 \leftarrow (E + Z_3)F + G\}$
26. Regresa $(X_3 : Y_3 : Z_3)$

Tabla 4.5: Algoritmo de suma de puntos de López-Dahab.

Algoritmo paralelo para suma de puntos elípticos, coordenadas LD-afines.
Entradas: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD, $Q = (x_2, y_2)$ en coordenadas afines sobre $E/K : y^2 + xy = x^3 + ax^2 + b$.
Salida: $P + Q = (X_3 : Y_3 : Z_3)$ en coordenadas LD

# ciclo	C_0	C_1
1. ciclo:	$Y_3 = y_2 \cdot Z_1^2 + Y_1$	
2. ciclo:	$X_3 = x_2 \cdot Z_1 + X_1$	
3. ciclo:	$T_1 = X_3 \cdot Z_1$	
4. ciclo:	$X_3 = X_3^2 \cdot (Z_1^2 + T_1)$	$Z_3 = T_1^2$
5. ciclo:	$X_3 = Y_3 \cdot T_1 + X_3 + Y_3^2$	$T_1 = Y_3 \cdot T1$
6. ciclo:	$T_1 = x_2 \cdot Z_3 + X_3$	
7. ciclo:	$Y_3 = (x_2 + y_2) \cdot Z_3^2$	$T_2 = T_3$
8. ciclo:	$Y_3 = (T_2 + Z_3) \cdot T_1 + Y_3$	

Tabla 4.6: Versión paralela del algoritmo de López-Dahab para suma de puntos.

4.2.3. Bisección de punto

El algoritmo de bisección de punto (mostrado en la sección §3.4.4.2), emplea cuatro operaciones aritméticas sobre campos finitos binarios:

- resolver una ecuación cuadrática (calculando la media traza de un elemento de $GF(2^n)$),
- una multiplicación polinomial,
- calcular la traza de un elemento de $GF(2^n)$,
- y finalmente extraer la raíz cuadrada de un elemento de $GF(2^n)$.

Más unas suma polinomiales que son despreciables (en cuanto a tiempo de ejecución se refiere) a comparación de las otras cuatro operaciones.

Las implementaciones de estas operaciones aritméticas fueron descritas en esta sección). Así como para la suma y doblado de puntos elípticos se emplea un solo multiplicador binario de Karatsuba-Ofman; a partir de esta operación se agrupan las operaciones restantes para poder realizar la bisección de punto.

Capítulo 5

Arquitectura en paralelo

En el capítulo previo (§4), se mostró cómo se implementaron las operaciones aritméticas de campos finitos binarios. Estos módulos se agruparon para poder formar una arquitectura paralela que permita realizar hasta dos operaciones por ciclo. Además se ha hecho mención de las modificaciones que se realizaron a los algoritmos de suma y doblado de puntos elípticos para obtener la versión paralela de dichos algoritmos, optimizando así los tiempos de ejecución de dichas operaciones.

En el presente capitulo se muestra la arquitectura completa del circuito que calcula la operación de multiplicación escalar en curvas elípticas. Este circuito emplea la unidad aritmético lógica presentada en la sección previa, y gracias a esta unidad es posible realizar las operaciones aritméticas de curvas elípticas (bisección, doblado y suma) en uno, cuatro y ochos ciclos de reloj respectivamente. Se presentarán los flujos de ejecución de estas operaciones aritméticas, así como la unidad de control que gestiona dichos flujos, y por último se darán a conocer los resultados obtenidos en espacio y tiempos de ejecución sobre la plataforma Virtex-II.

5.1. Expansión del escalar k

Para poder calcular la multiplicación escalar en curvas elípticas de una forma rápida y eficiente, se utilizan métodos de ventaneo. Estos métodos permiten reducir el número de operaciones aritméticas de curvas elípticas a realizar. El método binario para multiplicación escalar tiene un tiempo ejecución aproximado de

$$\frac{m}{2}A + mD \tag{5.1}$$

Donde A, D representan el tiempo de ejecución de las operaciones de suma y doblado de puntos elípticos respectivamente, y m es el tamaño del campo finito binario.

Sin embargo si se emplean otro tipo de representación par el escalar k, es posible reducir el número de sumas de puntos elípticos a calcular. Los métodos de ventaneo permiten reducir el tiempo de ejecución a:

$$\frac{m}{w+1}A + mD \tag{5.2}$$

Donde w denota el tamaño de la ventana a emplear.

Para utilizar los métodos de ventaneo, es necesario emplear la forma no adyacente (NAF) para representar el escalar k. En la sección §3.4.3, se explica a detalle cómo transformar el escalar a una representación digital con signo. En la implementación se ocupó una ventana de tamaño 2, por lo que los posibles valores de la expansión del escalar k son : $0, 1, -1$.

Se implementó el algoritmo 3.1 en hardware reconfigurable. El circuito resultante de dicha implementación se muestra en la figura 5.1

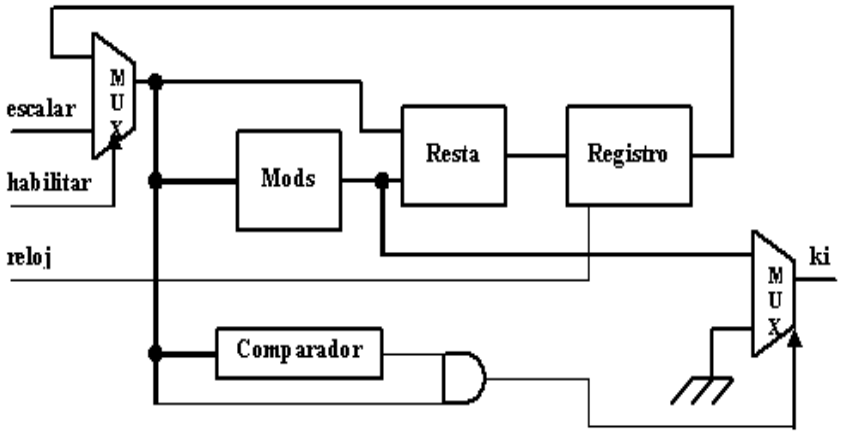

Figura 5.1: Circuito para calcular la expansión w-NAF.

El circuito consta de cuatro módulos principales:

1. Un módulo para obtener el modulo de un número con signo (mods). Hay que resaltar que el resultado obtenido por la operación $u = k \bmod 2^w$, es un entero tal que $u \equiv k \bmod 2^w$, y que además $-2^{w-1} \leq u < 2^{w-1}$.

2. Un módulo para realizar un resta en aritmética de números enteros.

3. Un comparador que habilita el funcionamiento del ciclo principal del algoritmo mostrado en la tabla 3.1.

4. Y finalmente un registro de 163 bits, para almacenar los resultados parciales del escalar k.

Para poder realizar la operación "mods", primero se realiza la operación "mod", al tratarse de un modulo que es una potencia de dos (2^n), para su implementación en hardware es sencillo: hay que tomar solamente los n bits menos significativos. Luego para que se cumpla la condición $-2^{w-1} \leq valor < 2^{w-1}$, hay que restarle al modulo obtenido el valor de 2^w, para así obtener el resultado de la operación "mods".

La figura 5.2, representa el circuito que se implementó para realizar la operación modular "mods". El circuito está constituido a su vez de un substractor (aritmética de números enteros) de tres bits. Este substractor resta un valor constante de 2^w ($2^2 = 4$ para nuestro caso donde se emplea una ventana de tamaño 2). El comparador que se emplea habilita que salida del multiplexor: si el resultado del comparador es 0, la salida será el valor de entrada (los n bits menos significativos), lo cual quiere decir que el módulo se encuentra de los valores limites de la operación "mods". Sin embargo si el resultado del comparador es 1, el módulo no se encuentra dentro de los valores límites y hay que sustraer 2^w para obtener el resultado correcto.

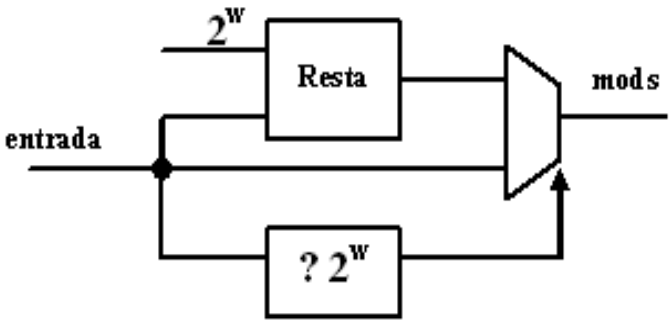

Figura 5.2: Circuito para calcular la operación modular mods.

El substractor de la figura 5.1, es un substractor de 163 bits. Una de las entradas del substractor es el valor del escalar k (163 bits), pero la segunda entrada es de sólo dos bits (ya que el valor que se obtiene del operación "mods" puede ser 0 ó 1 ó -1).

El multiplexor de la salida puede dar como resultado 0 ó k_i (que es resultado de la operación "mods"). Da 0 mientras el bits menos significativo de k sea 0 (es decir si k es par). Da k_i, mientras la salida del comparador sea 1 (es decir mientras $k \geq 1$) y el bits menos significativo de k sea 1 (es decir si k es impar).

Para realizar la división por 2 que requiere el algoritmo mostrado en la tabla 3.1 se efectúa un corrimiento de bits de izquierda a derecha (es decir del bit más significativo al menos significativo).

Al sintetizar circuito que calcula expansión w-NAF del escalar k, se obtuvieron como resultados:

- Espacio en área: 262 slices, 327 slice flip-flops.
- Tiempo de ejecución: un periodo mínimo de 12,87ns

Para obtener un coeficiente k_i de la expansión w-NAF, se requiere un solo ciclo de reloj. Para obtener todos los coeficientes k_i de la expansión w-NAF se requieren de m ciclos de reloj (siendo m el tamaño en bits del escalar k), por lo que el tiempo estimado para obtener la expansión w-NAF es de:

$$mT = 163(12{,}52ns) = 2{,}097\mu s \tag{5.3}$$

Donde T es el periodo del circuito.

5.2. ALU

Una vez que se tiene la expansión w-NAF del escalar k, se prosigue con la implementación de una unidad que permita calcular operaciones aritméticas tanto de campos finitos binarios como operaciones aritméticas de curvas elípticas. Con dicho fin se implementó una unidad aritmético-lógica (ALU), esta unidad puede realizar las operaciones aritméticas de campos finitos binarios como: sumas, multiplicaciones, elevar al cuadrado, obtener raíces cuadradas, y calcular trazas y medias trazas de un elemento del campo finito definido.

La unidad aritmético-lógica (ALU) es la unidad principal de la arquitectura propuesta, está consta de seis circuitos que realizan la operación de elevar al cuadrado, siete sumadores binarios, una unidad que calcula la raíz cuadrada de un elemento de $GF(2^m)$, una unidad de traza, una unidad de media-traza, un multiplicador binario de Karatsuba-Ofman de 163 bits.

La constitución de cada uno de estos bloques se puede observar en la sección §4.1. La figura 5.3 muestra como se acomodaron dichos componentes.

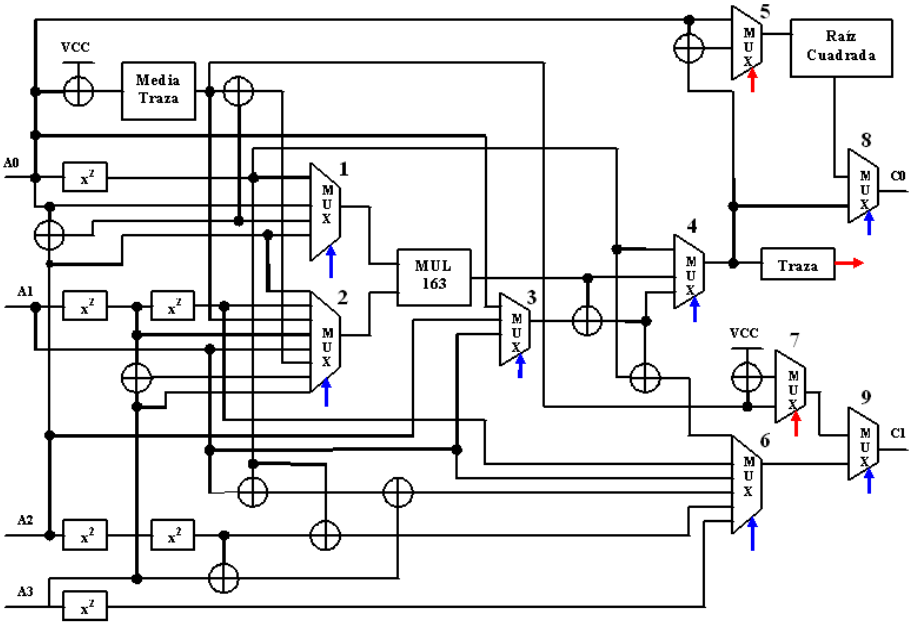

Figura 5.3: Circuito ALU.

Además este circuito consta de nueve multiplexores, los cuales permiten escoger que operación se va a calcular:

- Primer multiplexor (primer operando del multiplicador): A_0^2, A_0, A_2, $A_0 + A_2$.
- Segundo multiplexor (segundo operando del multiplicador): A_1, A_2, A_3, A_1^2, A_1^4, $HT(A_0)$, $A_1^2 + A_3$, $A_0 + A_2 + HT(A_0)$.
- El tercer multiplexor selecciona que operando que se va a sumar al resultado dado por el multiplicador binario, el operando a escoger es: A_0, A_1 y A_2.
- El cuarto multiplexor al igual que el tercero tiene tres entradas a seleccionar (A_0^2, MUL[*], $MUL + MUX3_{out}$ [**]).
- El quinto multiplexor selecciona el operando de entrada (A_0, $MUX4_{out} + A_0$ [***]) para el modulo que calcula la raíz cuadrada.

[*] MUL denota el resultado obtenido del multiplicador binario de Karatsuaba
[**] $MUX3_{out}$ denota la salida del multiplexor 3
[***] $MUX4_{out}$ denota la salida del multiplexor cuatro.

- El sexto multiplexor alimenta una de las entradas del multiplexor nueve, la salida corresponde a alguna de las siguientes opciones: A_1, A_0^2, $MUL + MUX3_{out}$, A_1^4, $A_0^2 + A_1 + A_3 + A_2^4$, $A_2^4 + A_0^2$, A_3^2.
- El séptimo multiplexor sirve de entrada al multiplexor nueve, su salida corresponde a una de dos opciones: $HT(A_0)$ ò $HT(A_0) + 1$.
- El multiplexor ocho selecciona una de dos entradas: R_{out} [****]
- El multiplexor nueve selecciona una de dos entradas: $MUX7_{out}$ [*****], $MUX6_{out}$ [******]

La ALU tiene cuatro entradas (A_0, A_1, A_2, A_3) que es por donde se cargan los datos a procesar (operandos), así como dos salidas (C_0, C_1) que permiten hasta dos escrituras simultáneas dentro del bloque de registros.

Para poder escribir los resultados dados por las operaciones aritméticas de campos finitos y las operaciones aritméticas de curvas elípticas se emplearon memorias RAM de doble puerto que permiten la escritura de dos registros a la vez en un solo ciclo de reloj.

5.3. Flujo de Datos

Una vez que se tiene los bloques que calculan las operaciones aritméticas de campos finitos binarios dentro de la ALU, está se puede ocupar para construir primitivas (operaciones) que permitan realizar las operaciones aritméticas de curvas elípticas (sea suma, doblado y bisección de puntos elípticos).

Como se mostró en la sección §4.2 se puede realizar el doblado de puntos elípticos en cuatro ciclos de reloj; y la suma de puntos elípticos en ocho ciclos de reloj, empleando el enfoque de una arquitectura en paralelo. Las figuras 5.4 y 5.5 ilustran como se realizan dichas operaciones en tres fases:

1. carga de datos,
2. procesamiento de datos,
3. y escritura de datos.

[****] R_{out} denota la salida del modulo de raíz cuadrada.
[*****] $MUX7_{out}$ denota la salida del multiplexor siete.
[******] $MUX6_{out}$ denota la salida del multiplexor seis.

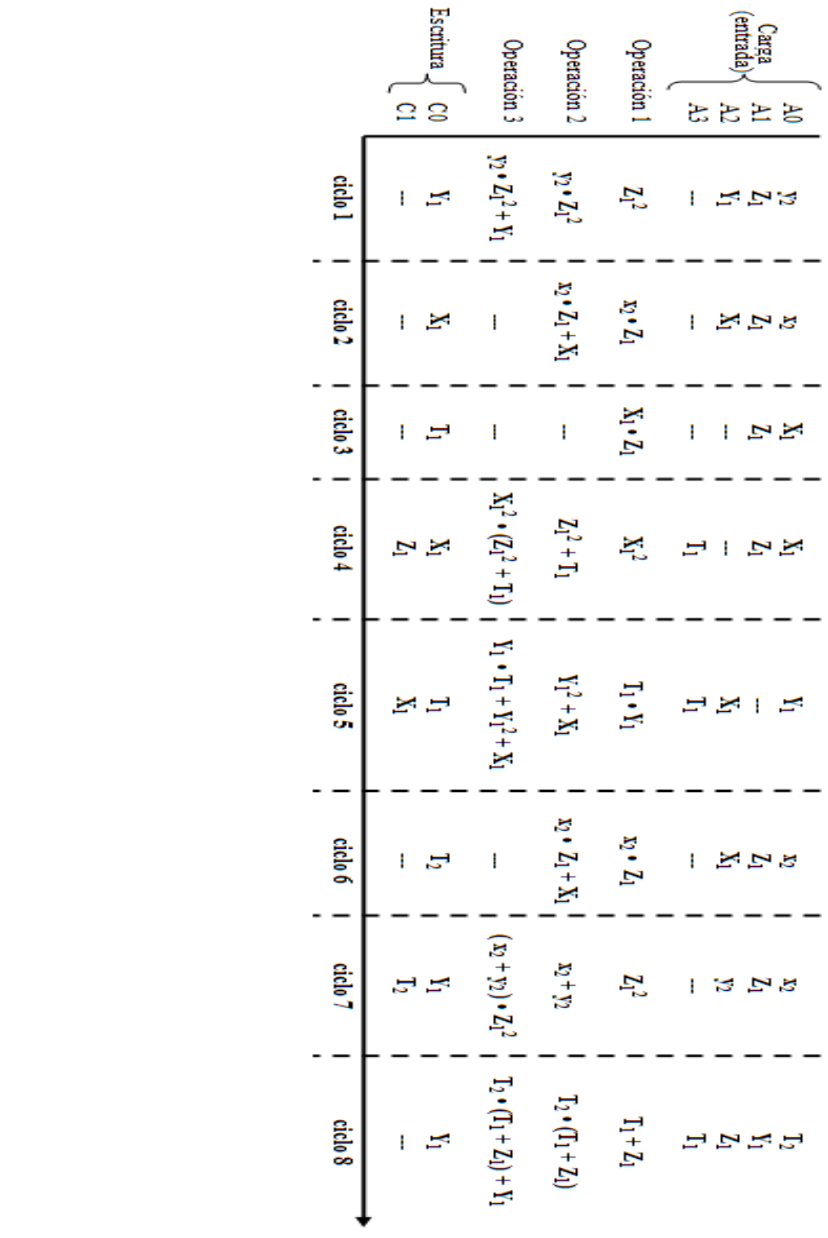

Figura 5.4: Cronograma de ejecución de la suma de puntos elípticos.

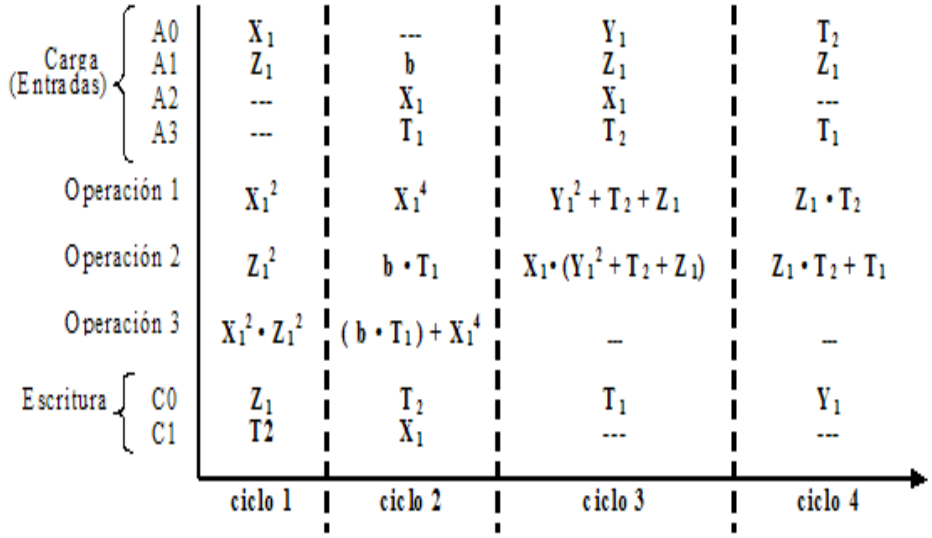

Figura 5.5: Cronograma de ejecución del doblado de puntos elípticos.

Dentro de la fase de carga de datos se realiza la lectura de los registros de almacenamiento, se leen hasta cuatro operandos los cuales van a cargar dentro de las cuatro entradas de la ALU. Como se hace empleo de BRAM, éstas permiten también leer dos registros por ciclo de reloj.

Luego en la fase de procesamiento de datos se realizan las operaciones de campos finitos correspondientes a la palabra de control que recibe. Esta fase se divide en tres etapas:

1. Pre-procesamiento: donde se realizan operaciones de elevar al cuadrado, o sumas, o cálculos de la media traza del elemento en $GF(2^m)$. Los resultados que se obtienen en esta etapa sirven como operandos para el multiplicador binario de Karatsuba-Ofman.

2. Multiplicación: consiste en realizar una multiplicación sobre un campo binario. Este bloque es el que emplea mayores recursos del FPGA, así como también es el que mayor tiempo de ejecución emplea.

3. Post-procesamiento: es esta etapa se realizan los cálculos finales para poder dar resultados a los registros de escritura. De las operaciones que se pueden ejecutar en esta etapa son calculadas la raíz cuadrada, suma o la traza de en elemento de $GF(2^m)$.

Finalmente en la fase de escritura de resultados, se envían los resultados obtenidos de la ALU al bloque de registros, donde de acuerdo al paso en que se encuentre del algoritmo sea de suma de puntos, sea de doblado de puntos, se envían a las respectivas memorias.

Como ejemplo, para poder ilustrar el flujo de datos dentro de la ALU, suponga que se desea realizar una bisección de punto en coordenadas afines. El flujo de datos correspondientes a esta operación se puede observar en la figura 5.6.

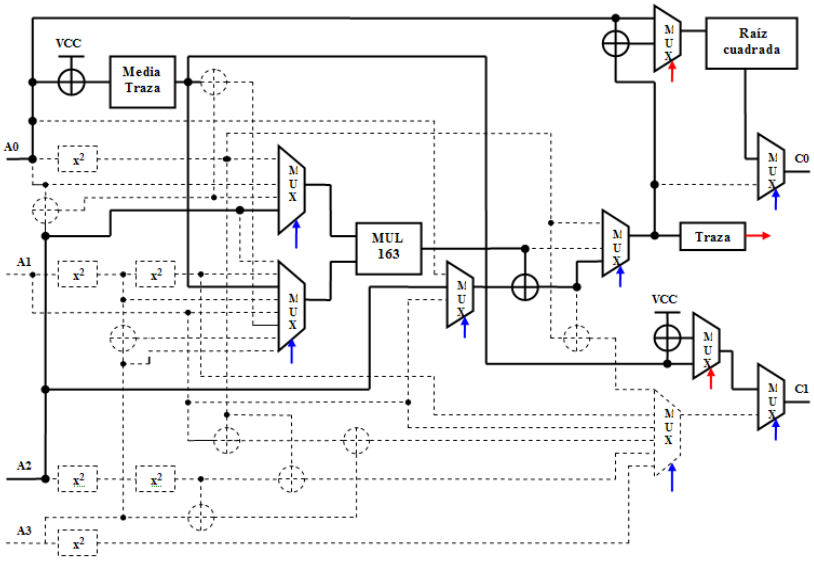

Figura 5.6: Ejecución de la bisección de punto en coordenadas afines.

Primero se cargan los datos x_2, y_2 en las entradas A_0, A_2 de la unidad ALU. Luego se pasan las operaciones $HT(A_0 + 1)$ y A_2 como operandos al multiplicador binario de Karatsuba-Ofman; gracias a las ordenes dadas por la unidad de control. Después se ejecuta la operación $A_2 \cdot HT(A_0 + 1)$. Y el resultado que se obtiene de esta multiplicación se pasa a la unidad de traza, esto para poder enviar el operando adecuado al circuito que calcula la raíz cuadrada. Finalmente se seleccionan las salidas correspondientes de la operación de bisección de punto y se envían a las salidas C_0, C_1 de la ALU.

5.4. Unidad de Control

Una vez establecido el circuito que se encarga de las operaciones aritméticas, es necesario crear una unidad que se encarga de gestionar la secuencia de operaciones necesarias

para poder realizar la multiplicación escalar. La unidad de control (ver figura 5.7) es el circuito que se encarga de dicha función.

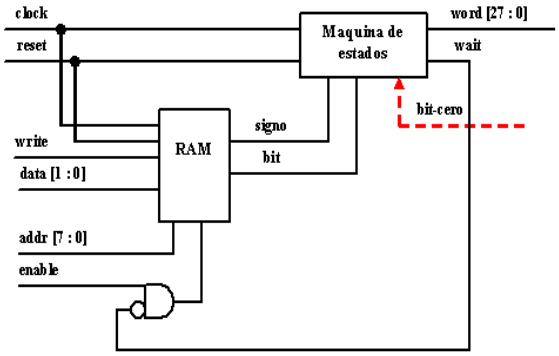

Figura 5.7: Arquitectura de la unidad de control.

La unidad de control está constituida principalmente por una máquina de estados, y en una memoria RAM donde se almacena la expansión w-NAF del escalar k. Luego estos valores sirven a la unidad de control para que pueda gestionar las operaciones que se deben de realizar dentro de la ALU; es decir que genera la palabra de control correspondiente a una operación, ya sea suma, doblado o bisección de un punto elíptico.

La máquina de estados dependiendo del valor k_i que se lee de la expansión w-NAF (obtenido de la memoria RAM), genera la palabra de control para que la ALU realice alguna de las tres operaciones siguientes:

- Suma de puntos elípticos: que consta de ocho ciclos para su ejecución (ver figura 5.4). Esta operación se realiza si el valor de la expansión es 1 ó -1.
- Doblado de un punto elíptico: que consta de cuatro ciclos para su ejecución (ver figura 5.5). Esta operación se realiza si y solo si el bit "bit-cero" está activo, y que corresponde al caso en que se requiera hacer un doblado de punto en vez de una suma.
- Bisección de un punto elíptico: un solo ciclo de ejecución. Si el valor de la expansión es 0.

Si se requiere calcular una suma o un doblado, la máquina de estados además genera un señal de espera (wait) la cual deshabilita la lectura de la memoria, en caso de que se requiera calcular una suma o un doblado de puntos elípticos.

La unidad de control consta de siete señales de entrada y una señal de salida:

- clock: que se encarga de sincronizar el funcionamiento del circuito completo.
- reset: la señal de reset permite borrar la memoria, además inicializar el funcionamiento del circuito.
- write: al activar esta señal se permite la escritura de la memoria RAM, en la dirección dada por la señal de addr.
- enable: esta señal al estar en activo permite sea la escritura, sea la lectura de la memoria RAM.
- data [1:0]: los valores de esta señal son los valores correspondientes de la expansión del escalar k, a escribir en la memoria, y que provienen de la unida w-NAF.
- addr [7:0]: es la señal que corresponde a la dirección en memoria a leer o escribir.
- bit-cero: es la señal proveniente de la ALU, y que corresponde al caso en que se requiera ejecutar una operación de doblado de punto, en lugar de una suma de puntos elípticos.
- palabra ([27:0]) : es la señal de salida. Esta es la palabra de control que gestiona el funcionamiento global de toda la arquitectura. Esta palabra de control consiste de veinte y ocho bits los cuales están ordenados de la siguiente forma:

$$\underbrace{10XX001010}_{\text{dirección}}\ \underbrace{1100}_{MUX}\ \underbrace{100110010XXX1X}_{ALU}$$

  los primeros diez bits indican que dirección del bloque de memoria se va a leer, los cuatro siguientes bits permiten seleccionar los operandos que se cargarán en la ALU, y finalmente los últimos catorce bits indican que operaciones se ejecutaran dentro de la ALU, conforme lo indica la tabla 5.1.

La tabla 5.1, muestra el tipo de operaciones que puede calcular la ALU por ciclo de reloj, así como su respectiva palabra de control necesaria para ejecutar dichas acciones. Las primeras ocho filas muestran la secuencia a seguir para poder obtener un suma de puntos elípticos. Las siguientes cuatro filas muestran las operaciones para obtener un doblado de punto. Y finalmente las últimas tres filas son las operaciones para obtener una bisección de punto; ya sea en coordenadas afines (primera fila), ya sea en representación lambda (segunda fila). La última fila es para realizar el cambio de representación-λ a coordenadas

operación	entradas	palabra de control	salidas
	$a_0a_1a_2a_3$	$s_{25}\cdots s_0$	c_0c_1
$Y_1 = y_2 \cdot Z_1^2 + Y_1$	$y_2Z_1Y_1-$	1xx01000xx11010000110xxx1x	Y_1x
$X_1 = x_2 \cdot Z_1 + X_1$	$x_2Z_1X_1-$	110xxxx0xx00010010110xxx1x	X_1x
$T_1 = X_1 \cdot Z_1$	X_1Z_1--	10xxxxx0x0xx01001xx00xxx1x	T_1x
$X_1 = X_1^2 \cdot (Z_1^2 + T_1)$	$X_1Z_1 - T_1$	00xxxxx010xx00100xx0000111	X_1Z_1
$X_1 = Y_1 \cdot T_1 + X_1 + Y_1^2$	$y_2Z_1Y_1-$	0xx01000xx11010000110xxx1x	T_1X_1
$T_2 = x_2 \cdot Z_1 + X_1$	$x_2Z_1X_1-$	110xxxx0xx00010010110xxx1x	T_2x
$Y_1 = (x_2 + y_2) \cdot Z_1^2$	$x_2Z_1y_2-$	01xxx010xx0111000xx00xxx1x	Y_1x
$Y_1 = (T_1 + Z_1) \cdot T_2 + Y_1$	$Y_1T_1T_2Z_1$	0xx0010x1011100110010xxx1x	Y_1x
$Z_1 = X_1^2 \cdot Z_1^2$	X_1Z_1--	00xxxxx0x0xx00000xx0000011	Z_1T_2
$X_1 = (X_1^4 + T_1) \cdot (Y_1^2 + Z_1 + T1)$	$Y_1Z_1X_1T_1$	0x010xxxx10xxxxxxxx0101011	T_2X_1
$Y_1 = Z_1 \cdot T_1 + T_2$	$T_2Z_1 - T_1$	00xxx101xx01010010110xxx1x	Y_1x
Bisección de punto (afines)	$x_2 - y_2-$	101xxx01xx01011010110xxx00	x_2y_2
Bisección de punto (representación-λ)	$x_2 - y_2-$	101xxx01xx0101110xx00xxx00	x_2y_2
$y_2 = \lambda x_2 + x_2^2$	$x_2 - y_2-$	101xxx01xx01010011010xxx1x	$-y_2$

Tabla 5.1: Tabla de operaciones que se pueden realizar dentro de la ALU.

afines; cabe señalar que esta operación sólo es necesaria cuando se requiere realizar una suma de puntos elípticos.

La primera columna representa la operación que se realiza, la segunda columna señala las entradas de la ALU que se emplean, la tercera es la palabra de control que se genera; la cual es la que controla las operaciones de la ALU, y por último la cuarta columna muestra las salidas que se escribirán dentro del bloque de registros.

5.5. Arquitectura Final

La arquitectura propuesta que permite calcular la multiplicación escalar sobre curvas elípticas se muestra en la figura 5.8; esta arquitectura esta compuesta por tres componente principales, una unidad que calcula la expansión w-NAF del escalar k, una unidad aritmético-lógica (unidad ALU) que calculas las operaciones de campos finitos y de curvas elípticas, y finalmente por una unidad de control que gestiona el circuito completo.

Además de las tres unidades mencionadas anteriormente, se puede observar que hay un bloque de registros en medio de la ALU y de la unidad de control. Estos registros permiten primero cargar los datos iniciales, para después guardar los cálculos realizados por la ALU (actúan como registros acumuladores), así como almacenar valores temporales. Este bloque de registros consiste de tres pequeños bloques de dos o tres registros cada uno.

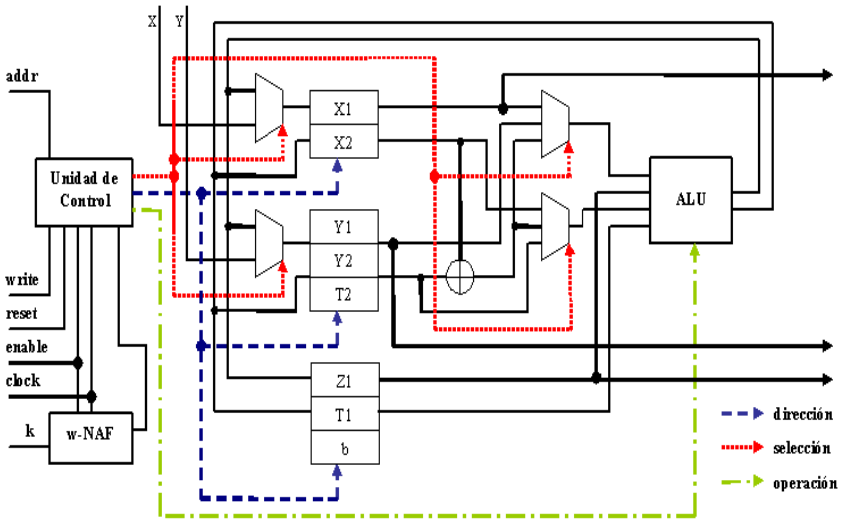

Figura 5.8: Arquitectura para calcular kP.

Para realizar la operación elíptica $Q - P$, se requiere convertir el punto P al punto $-P$, lo que involucra que las coordenadas (x, y) del punto P cambien a las coordenadas $(x, x + y)$, que es una propiedad de las curvas elípticas sobre campos finitos binarios. Para lograr esta conversión, como los registros X_1, Y_1, contienen los valores del punto P solo hay que sumar ambos valores para obtener la coordenada $x + y$. Con esta solución, se evita guardar las coordenadas del punto $-P$, pues éstas se obtienen directamente a partir de las coordenadas del punto P.

Los dos multiplexores que se localizan antes de la unidad de registros, permiten seleccionar que datos se van a guardar en dichos registros. Inicialmente se cargan los valores (coordenadas) del punto inicial de la curva (x, y), en lo posterior se guardaran los valores temporales dados por la unidad ALU, y al final del proceso en los registros X_1, Y_1, Z_1, se encontrarán las coordenadas del punto final dado por la operación kP

Los multiplexores que están ubicados antes de la unidad ALU, permiten seleccionar qué operandos serán introducidos en la ALU.

Todos los módulos mencionados fueron implementados en una plataforma de hardware reconfigurable, empleando la herramienta de Xilinx ISE Foundation v. 6.3i, y para verificar el correcto funcionamiento de la implementación se empleó la herramienta de ModelSim XE II 5.8c. Además se empleo la herramienta de Maple 9 para comparar resultados.

5.6. Resultados

Como se ha estado mencionando a lo largo de la tesis, la arquitectura propuesta nos permite realizar tres operaciones aritméticas básicas de curvas elípticas suma de puntos, doblado de puntos y bisección de puntos elípticos. La tabla 5.2 presenta los ciclos que requieren cada una de estas operaciones.

Operación Aritmética de Curva Elíptica	# ciclos
Bisección de Punto (coordenadas afines)	1
Bisección de Punto (representación-λ)	2
Doblado de Punto	4
Suma de Puntos	8

Tabla 5.2: Ciclos por operación

Como se puede apreciar, la suma de puntos elípticos es la operación que requiere mayor número de ciclos (ocho), por que su implementación requiere cuando menos de ocho multiplicaciones (empleando coordenadas mixtas), puesto que la arquitectura consta de un solo multiplicador binario de Karatsuba-Ofman (debido a su costo en área). De manera análoga el doblado de puntos requiere de cuatro multiplicaciones (empleando coordenadas proyectivas). La bisección de punto en representación-λ requiere de uno o dos ciclos; un ciclo si la siguiente operación es solamente otra bisección de punto, dos ciclos si la siguiente operación es una suma de puntos. Y la operación menos costosa es la bisección de punto en coordenadas afines que solo emplea un ciclo de reloj.

5.6.1. Tiempos y Costo en Área

Como se mencionó al principio del trabajo, se trabajó sobre una plataforma de hardware reconfigurable. La arquitectura de la ALU junto con la de la unidad de control se implementaron en un FPGA VirtexE XCV3200. Se empleó la herramienta Xilinx Foundation Tool F6.3i para la programación del FPGA, la implementación de la arquitectura y la síntesis de la misma. Para pruebas de simulación y verificación de resultados se empleo la herramienta de ModelSim XE II 5.8c. La tabla 5.3 muestra tanto los tiempos de ejecución de cada un de los componentes de la arquitectura, así como su costo en área (slices).

Nótese que el multiplicador binario de Karatsuba-Ofman, es el componente que emplea mayores recursos (multiplicador binario más reducción pentanomial) con 6730 slices. Los componentes restantes ocupan mucho menos recursos que el multiplicador binario, por lo que es posible ocupar más de un componente de elevar al cuadrado. En cuanto a

Circuito	CLB slices	Tiempos
Multiplicador binario de Karatsuba-Ofman $GF(2^{163})$	6730	21,620ns
Unidad de Media Traza $GF(2^{163})$	1,258	14,621ns
Unidad para Elevar al Cuadrado $GF(2^{163})$	95	7,938ns
Unidad de Raíz Cuadrada $GF(2^{163})$	328	21,243ns
ALU + Unidad de Control $GF(2^{163})$	11,063	41,765ns

Tabla 5.3: Costos en área y tiempos.

los sumadores binarios no se especifican en la tabla, pero al tratarse de sumas binarias sin acarreos, solamente se emplean compuertas XOR; y como se empleó la curva elíptica B-163 (recomendada por el NIST [27]) los sumadores binarios emplean 163 compuertas XOR.

En cuanto a tiempos de ejecución los componentes que ocupan mayor tiempo de ejecución son el multiplicador binario de Karatsuba-Ofman, la unidad que calcula la raíz cuadrada y la unidad que calcula la media traza de un elemento de $GF(2^m)$ con 21,620ns y 21,243ns 14,621ns respectivamente. La unidad para calcular la media traza de un elemento de $GF(2^m)$ y los sumadores binarios son las componentes que efectúan las operaciones más rápidas.

5.6.2. Comparaciones

La arquitectura propuesta tiene un periodo de ejecución de 41,765ns, y para poder obtener el tiempo de ejecución de una operación completa de multiplicación escalar en curvas elípticas se obtiene de la siguiente forma: se requieren de ocho ciclos de reloj para obtener una suma de puntos elípticos en coordenadas mixtas. Sin embargo es necesario pasar de representación-λ a coordenadas afines, por lo que se requieren 8 (suma de puntos elípticos) + 3 (bisección de punto + conversión) = 11 multiplicaciones. Sin embargo como se mencionó, se emplea un método de ventana (método w-NAF), por lo que hay que realizar la expansión w-NAF del escalar k y la cual se ejecuta en un tiempo T, que en este caso es una ventana de dos. Por lo que el tiempo aproximado de ejecución es

$$\left(\frac{t}{w+1} - 2^{w-2}\right)11M + T \tag{5.4}$$

donde M denota el tiempo de ejecución necesario para calcular una multiplicación sobre el campo $GF(2^m)$, y así el tiempo de ejecución de la implementación es: 0,027ms.

La tabla 5.4 es una tabla de comparación, donde se muestran diversas implementaciones de multiplicación escalar sobre una plataforma de hardware.

Ref.	Campo	Plataforma	kP	Frecuencia	Área	Speed/Área
[18]	$GF(2^{178})$	VLSI	4,4ms	227Hz	143,000 gates	n/a
[43]	$GF(2^{167})$	XCV400E	0,21ms	4.7MHz	3,002 LUTs	1.56
[19]	$GF(2^{163})$	XCV2000E	0,143ms	6.9MHz	-	-
[19]	$GF(2^{193})$	XCV2000E	0,187ms	5.3MHz	-	-
[44]	$GF(2^{163})$	XCV2000E	0,075ms	13.3MHz	10,017 LUTs	1.32
[23]	$GF(2^{191})$	XCV3200E	0,056ms	17.8MHz	19,626 slices	0.90
Este trabajo	$GF(2^{163})$	XCV3200E	0,027ms	41MHz	11,616 slices	3.52

Tabla 5.4: Tabla de comparación.

G. Orlando y C. Paar [43] presentaron un procesador en hardware reconfigurable que realiza las operaciones de curvas elípticas en un campo finito binario ($GF(2^m)$); este procesador emplea multiplicadores de 16 bits, con el cual obtienen un tiempo de ejecución de $210\mu s$, para realizar una multiplicación escalar.

N. Gura, S. Shantz, y H. Eberle [19] muestran un diseño el cual emplea un multiplicador de 64 bits, el cual emplea el enfoque de suma y corrimiento de bits, este algoritmo emplea una ventana de 4 para poder realizar la operación kP en 0,143ms.

J. Lutz [44] presenta un co-procesador para ejecutar operaciones aritméticas sobre curvas elípticas, dentro de su diseño emplean un seriales de 41 bits, además de emplear el método binario conjunto con coordenadas proyectivas para poder calcular kP, obteniendo así un tiempo de ejecución final de 0,075ms.

Finalmente la propuesta de N. A. Saqib, F. Rodríguez-Henríquez, A. Díaz-Pérez [23], presenta una arquitectura en paralelo para realizar la multiplicación escalar en campos finitos binarios, esta arquitectura emplea dos multiplicadores binarios de Karatsuba-Ofman, obteniendo como tiempo de ejecución 56μs para realizar la operación kP.

R. Schroeppel, C. L. Beaver, R. Gonzáles, R. Miller y T. Draelos [18] han propuesto crear un procesador dedicado que realice todas las operaciones básicas de aritmética de campos finitos (sumas, divisiones, elevadas al cuadrado), así como las operaciones de curvas elípticas (sumas, doblados, bisecciones, multiplicación escalar), ya que tal arquitectura permitiría ejecutar las operaciones de firma/verificación de documentos electrónicos de una eficiente, de tal manera que la autenticación de usuarios y documentos ya no tendría que ser operaciones que empleen tanto poder de cómputo. Además esta propuesta permitiría crear una capa independiente de las aplicaciones, de tal forma que se podrían crear aplicaciones sobre esta capa para realizar las operaciones aritméticas de una forma rápida y eficiente.

Los trabajos que se muestran son los más relevantes para realizar una comparación con esta nueva implementación, ya que se tratan en su mayoría de implementaciones en hardware reconfigurable de multiplicación escalar en curvas elípticas. El trabajo que se presenta en [18], es hasta el momento la única implementación de multiplicación escalar

empleando bisección de punto, sin embargo esta implementación es sobre una plataforma VLSI. Los trabajos restantes [43, 19, 44, 23] son implementaciones en FPGA's.

Como se puede apreciar en la tabla 5.4, el diseño que se propone muestra un mejor rendimiento (en tiempo), a comparación de los demás trabajos, sin embargo ocupa bastantes recursos, ocupa menos recursos que en el trabajo [23], ya que esta implementación ocupa dos multiplicadores binarios de Karatsuba en paralelo.

Capítulo 6

Conclusiones

6.1. Resumen

El punto importante que se trató en este trabajo de tesis, fue de realizar un estudio y un análisis del algoritmo de suma y bisección de punto ("half-and-add") que se emplea para calcular la multiplicación escalar (kP) de un punto elíptico. Así como su implementación sobre una plataforma de hardware reconfigurable. Se propuso desde el inicio como objetivo, el realizar una implementación óptima y eficiente en cuanto a espacio en área y tiempo de ejecución dentro de un FPGA.

A fin de lograr ese objetivo, primero se analizó la constitución de las aplicaciones que emplean criptografía de curvas elípticas. Al realizar este análisis se observó que un criptosistema de curvas elípticas está conformado por cinco capas: la capa de aplicación, los protocolos de curvas elípticas, las primitivas de curvas elípticas, la aritmética de curvas elípticas y finalmente la aritmética de campos finitos. Se optó por emplear curvas elípticas sobre campos finitos binarios, ya que gracias a su naturaleza y sus características tienen una mayor afinidad para implementarse sobre plataformas de hardware, para los propósitos de este estudio se tomo la curva elíptica aleatoria B-163 definida por el NIST [27]. Además se tomo como método para calcular la operación kP, el método de suma y bisección ya que se ha demostrado que la operación de bisección de un punto elíptico puede a llegar a ser más eficiente que una operación de doblado de un punto elíptico. También se optó por emplear técnicas de ventaneo para reducir el número de sumas elípticas a calcular, en el cual es indispensable realizar una expansión w-NAF del escalar k. Por lo que se el trabajo se dividió en dos partes: crear la expansión w-NAF e implementar el algoritmo de suma y bisección de puntos elípticos.

Primero se enfocó a crear la unidad que realiza la expansión w-NAF, se tomó como

longitud de ventana un tamaño de dos. Esta unidad emplea aritmética de números enteros y requiere un substractor y de un divisor de 163 bits. Al tratarse de divisiones por dos el divisor son corrimientos de bits: del bit más significativo al menos significativo. La expansión w-NAF resultante del circuito se almacena en una memoria RAM, de donde la arquitectura lee los valores para realizar el cómputo de la operación kP.

Después para lograr una implementación eficiente del algoritmo de suma y bisección de puntos elípticos, hay que remarcar que las operaciones de curvas elípticas al ser una composición de operaciones aritméticas de campos finitos, se concentró primero en crear una aritmética de campos finitos binarios lo más rápida y eficiente que fuera posible. El estudio se abocó a crear primitivas en hardware reconfigurable. Se observó que la operación que requiere mayor tiempo de ejecución es la multiplicación polinomial, por lo que se optó por emplear un multiplicar paralelo. En este caso se uso un multiplicador paralelo de Karatsuba-Offman, el cual emplea el menor número de recursos disponibles (en cuanto área) para un multiplicador paralelo. Por otro lado, se lograron obtener propuestas para implementar las operaciones de elevar el cuadrado, raíz cuadrada, y media traza sobre campos finitos binarios, de una manera rápida y eficiente empleando solamente compuertas XOR, sin tener que recurrir a técnicas iterativas.

Posteriormente, a partir de estas operaciones sobre campos finitos binarios, se creó una arquitectura que empleará dichas operaciones aritméticas de campos finitos dando como resultado una unidad aritmético-lógica (ALU) que permite realizar operaciones aritméticas sobre curvas elípticas [45]. Para obtener esta arquitectura se requirió paralelizar operaciones de los algoritmos de suma y doblado de puntos elípticos, obteniendo así algoritmos paralelos que calculan la suma de dos puntos elípticos en ocho ciclos, y para el doblado de un punto elíptico en cuatro ciclos.

De los resultados obtenidos por el presente trabajo, el prototipo propuesto es capaz de calcular una multiplicación escalar en curvas elípticas (kP) en $0{,}027ms$. Todas estas operaciones aritméticas se realizan en representación polinomial de los elementos del campo $GF(2^{163})$. Mientras que la representación que se emplean para las operaciones en curvas elípticas son una combinación de coordenadas afines con coordenadas mixtas. El uso de coordenadas afines permite calcular la bisección de un punto elíptico (ya que no se puede calcular la bisección de un punto en coordenadas proyectivas). Por otro lado el uso de coordenadas proyectivas permite realizar la suma de puntos elípticos sin tener que calcular una división a cada paso del algoritmo. Sin embargo para realizar la suma de un punto en coordenadas proyectivas con otro punto pero en coordenadas afines, se emplea un algoritmo de suma elíptica que permite usar coordenadas mixtas, lo que ahorra la conversión de coordenadas.

La eficiencia de la arquitectura propuesta se demostró por medio de su implementación en un FPGA de la familia Virtex-II de Xilinx, el prototipo se programó gracias a la her-

ramienta de Xilinx ISE Foundation 6.3i, y los tiempos de ejecución y costo en área se obtuvieron mediante el simulador ModelSim XE II 5.8c. El tiempo obtenido (0,025ms), es el más rápido reportado dentro de la literatura abierta [43, 19, 18, 44, 23].

6.2. Contribuciones

En la presente tesis se analizó y se implementó un circuito que realiza la operación kP en 0,027ms sobre una plataforma de hardware reconfigurable, de donde se pueden destacar las siguientes contribuciones:

- Una implementación eficiente en una plataforma de hardware reconfigurable para calcular una multiplicación escalar en curvas elípticas empleando el método de suma y bisección.
- La arquitectura propuesta trabaja en un campo finito binario $GF(2^{163})$, y calcula una multiplicación escalar en 0,027ms.
- Se obtuvieron ecuaciones generales para poder calcular el cuadrado de un elemento del campo finito $GF(2^n)$ (caso trinomial), el cual es un método más eficaz para realizar dicha operación en hardware.
- Se obtuvieron ecuaciones generales para poder calcular la raíz cuadrada de un elemento del campo finito $GF(2^n)$ (caso trinomial), el cual es un método más eficaz para realizar dicha operación en hardware.
- Se proponen dos algoritmos paralelos para calcular la suma de puntos elípticos y el doblado de puntos elípticos en ocho y cuatro ciclos respectivamente.
- Los módulos que efectúan operaciones aritméticas en campos finitos binarios, se pueden adaptar para emplear campos finitos binarios de varios tamaños.
- La publicación de un artículo en un congreso nacional [45].
- Una tabla donde se proponen campos finitos $GF(2^n)$ de la forma $z^n + z^m + 1$ (caso trinomial), donde se pueden calcular de una forma eficiente tanto el cuadrado como las raíces cuadradas de los elementos de dichos campos finitos (ver apéndice C); ya que son las opciones donde se obtienen el menor numero de coeficientes por ecuación (< 6).

6.3. Trabajo a Futuro

En el trabajo a futuro a realizar, se proponer realizar más optimizaciones sobre los módulos propuestos. Se propone además trabajar sobre un campo finito binario $GF(2^m)$ generado por un trinomio irreducible (por ejemplo $P(z) = z^{233} + z^{74} + 1$. Así como también emplear otras estrategias para el modulo de multiplicación polinomial (por ejemplo multiplicadores seriales).

Apéndice A

Ecuaciones para $\sqrt{A}$ en $GF(2^m)$

Ecuaciones para obtener los coeficientes de un elemento arbitrario de $C \in GF(2^{163})$ tal que $C = \sqrt{A}$

$$
\begin{array}{rcl}
c_0 & = & \sum_{j=0}^{24} a_{159-6j} \oplus a_9 \oplus a_3 \oplus a_0 \\
c_1 & = & \sum_{j=0}^{26} a_{6j+5} \oplus a_2 \\
c_2 & = & \sum_{j=0}^{25} a_{6j+7} \oplus a_4 \\
c_3 & = & a_6 \oplus a_3 \\
c_4 & = & a_8 \oplus a_5 \oplus a_1 \\
c_5 & = & a_{10} \oplus a_7 \oplus a_3 \oplus a_1 \\
c_6 & = & \sum_{j=0}^{24} a_{159-6j} \oplus a_{12} \oplus a_5 \\
c_{80} & = & \sum_{j=0}^{26} a_{6j+5} \oplus a_{160} \sum_{j=0}^{25} a_{6j+3} \\
c_{81} & = & \sum_{j=0}^{26} a_{6j+1} \oplus a_{162} \sum_{j=0}^{25} a_{6j+5} \\
c_{160} & = & \sum_{j=0}^{26} a_{6j+3} \\
c_{161} & = & \sum_{j=0}^{26} a_{6j+5} \\
c_{162} & = & \sum_{j=0}^{26} a_{6j+1} \\
c_{3i+8} & = & \sum_{j=0}^{26} a_{6j+5} \oplus \sum_{j=0}^{23-i} a_{6(i+j)+19} \oplus a_{6i+16} \oplus \sum_{j=0}^{i+1} a_{6j+3}, \text{ para } i \in [0, 23] \\
c_{3i+9} & = & \sum_{j=0}^{26} a_{6j+1} \oplus \sum_{j=0}^{23-i} a_{6(i+j)+21} \oplus a_{6i+18} \oplus \sum_{j=0}^{i+1} a_{6j+5}, \text{para } i \in [0, 23] \\
c_{3i+7} & = & \sum_{j=0}^{26} a_{6j+3} \oplus \sum_{j=0}^{24-i} a_{161-6j} \oplus a_{6i+14} \oplus \sum_{j=0}^{i+1} a_{6j+1}, \text{ para } i \in [0, 24] \\
c_{3i+82} & = & \sum_{j=0}^{26} a_{6j+3} \oplus \sum_{j=0}^{25-i} a_{157-6j}, \text{ para } i \in [0, 25] \\
c_{3i+83} & = & \sum_{j=0}^{26} a_{6j+5} \oplus \sum_{j=0}^{25-i} a_{159-6j}, \text{ para } i \in [0, 25] \\
c_{3i+84} & = & \sum_{j=0}^{26} a_{6j+1} \oplus \sum_{j=0}^{25-i} a_{161-6j}, \text{ para } i \in [0, 25]
\end{array}
$$

Tabla A.1: Ecuaciones de los coeficientes c_i de C.

Apéndice B

Tecnología FPGA

B.1. Introducción

Las aplicaciones criptográficas que se emplean en la actualidad requieren de gran poder de computo para poder realizar operaciones de cifrado/descifrado, firma/verificación. Sin embargo se ha demostrado que las implementaciones en software de dichas aplicaciones, no son la mejor opción, ya que implica una carga de trabajo extra para las ya existentes aplicaciones digitales, ya que se funcionalidad y desempeño dependen de la plataforma en que se implementen, y en como se emplean los recursos de dichas plataformas.

Sin embargo en los últimos años se ha estado trabajando en implementaciones en hardware que ayuden a disminuir los tiempos de ejecución de las aplicaciones criptográficas, sobre todo el uso de computo reconfigurable se ha vuelto una opción atractiva por sus bajos costos de adquisición. Además de presentar ventajas como la facilidad y flexibilidad de programación.

Se define el computo reconfigurable como el proceso de programar dispositivos lógicos (dispositivos reconfigurables) para que realicen una determinada función o tarea. A diferencia de la programación lógica con un microcontrolador, el cómputo reconfigurable permite modificar y alterar el flujo de datos, así como el flujo de control; lo cual no puede fácilmente ser realizado en un microcontrolador. Dicho de otra forma, mientras que la programación lógica (en microcontroladores) es la programación de una máquina de propósito general, el cómputo reconfigurable permite la programación de dispositivos de tal forma que se pueden obtener máquinas de propósito especifico.

El cómputo reconfigurable hace uso de dispositivos reconfigurables para su programación. Estos dispositivos son comúnmente conocidos como FPGA's (Field-Programable Gate Array).

Arquitectura General

Un FPGA es un dispositivo semiconductor que se emplea para procesar información digital, de una manera similar a un microcontrolador. Es una tecnología basada en la programación de un arreglo de compuertas (gate array). Esto permite realizar la programación del dispositivo después de su manufactura. En la actualidad existen FPGA que tienen la capacidad de ser reprogramados en tiempo de ejecución.

Existen diversos fabricantes de dispositivos FPGA : Altera, Xilinx, Actel, QuickLogic, Lucent, entre otros. Los cuales producen diferentes tipos de FPGA con características especificas. La arquitectura general de un dispositivo FPGA se puede observar en la figura B.1.

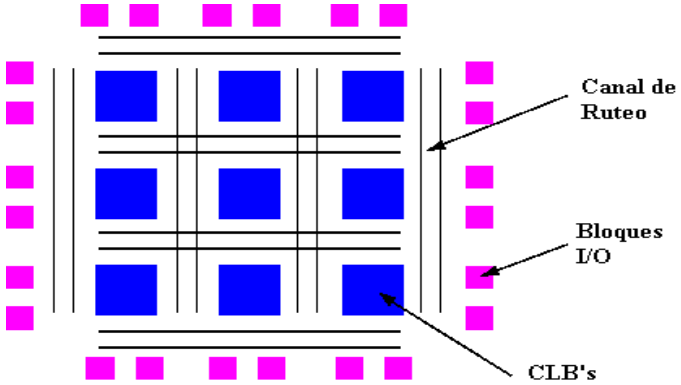

Figura B.1: Arquitectura general de un FPGA.

La arquitectura consiste de un arreglo de bloques lógicos (CLB's) y un arreglo de canales de ruteo, alrededor de dicho arreglo se encuentran los bloques de entrada/salida (IOB's), los cuales son bidireccionales (dependiendo de su configuración).

- CLB (Configurable Logic Block): Es un arreglo de celdas lógicas con múltiples entradas/salidas para programar. El arreglo se encuentra distribuido en filas y columnas dentro del circuito general. Estos CLB's se emplean para implementar macros o funciones lógicas especificas. La arquitectura general de un CLB se muestra en la figura B.2 Este circuito consta de un LUT de cuatro entradas [*], y de un flip-flop. La salida generada por el CLB pues ser registrada o no-registrada.

[*]Un LUT (Look-Up Table) es una arreglo o una matriz de valores donde se guarda información que se desea buscar.

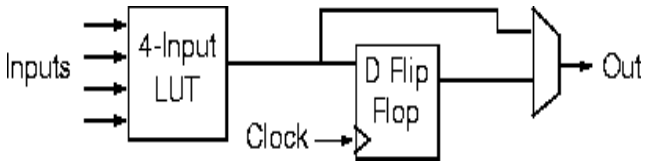

Figura B.2: Arquitectura general de un CLB.

- IOB (Input Output Buffer): Son los bloques que sirven como entrada/salida del dispositivo FPGA, estos puertos son duales, es decir pueden ser programados para que actúen como entradas o como salidas. Es decir interconectan el FPGA con dispositivos externos. En la figura B.3, muestra cómo están constituidos estos IOB's. Gracias a los flip-flops es posible que la salida pueda ser registrada o no-registrada.

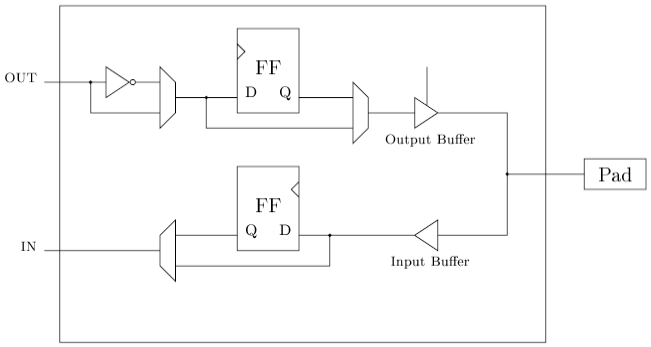

Figura B.3: Arquitectura general de un IOB.

- Canal de ruteo: Es una matriz de interconexión que comunica a los CLB's entre sí, y también comunican al CLB con los IOB's. Esta matriz de comunicación al igual que los los otros dos dispositivos es programable de acuerdo a la funcionalidad final que se requiera obtener.

Este trabajo utilizó el FPGA fabricado por la compañía Xilinx, en especial la familia Virtex-II, es cual fue empleado para crear el prototipo del trabajo presente.

Arquitectura de la familia Xilinx Virtex-II

La familia de dispositivos reconfigurables Virtex-II consiste de un arreglo de compuertas que son programables por el usuario, además de incluir una serie de elementos pro-

gramables extras. Como se puede observar en la figura B.4, la arquitectura del Virtex-II esta compuesta por bloques IOB's y por bloques internos reconfigurables.

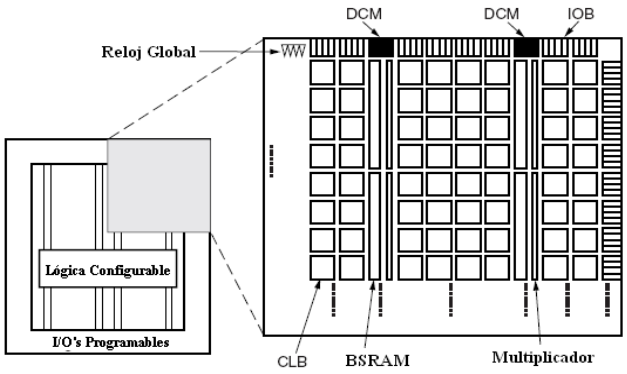

Figura B.4: Arquitectura general del dispositivo Virtex-II.

Dentro de los bloques internos reconfigurables se encuentran cuatro elementos ordenados en forma de un arreglo:

- CLB's (Configurable Logic Blocks): los cuales son los elementos funcionales, pueden ser programados para funcionar como circuitos combinacionales o como circuitos secuenciales. Además pueden permitir el almacenamiento de ciertos valores.
- BSRAM (Block Select RAM): son módulos de memoria, pueden almacenar hasta 18 Kbits, en bloques RAM de doble puerto.
- Bloques Multiplicadores (Multiplier Blocks): multiplicadores dedicados de 18 bits.
- DCM (Digital Clock Manager): es un bloque que permite distribuir la señal de reloj dentro de cada bloque, así como también permite crear multiplicadores y divisores de frecuencia.

Todos los bloques internos están interconectados por medio de una Matriz General de Ruteo (GRM), el cual es un arreglo de ruteadores. Cada elemento interno se comunica con los demás a través de esta matriz general de interconexión, lo que permite tener múltiples conexiones. La programación de esta matriz de realiza de un forma jerárquica por lo que permite crear diseños de gran velocidad.

Todos los elementos programables, incluyendo los recursos de ruteo están controlados por valores almacenados en celdas de memoria estática. Estos valores son cargados dentro de celdas de memoria durante la configuración, y pueden ser recargados para cambiar la funcionalidad de los elementos programables.

IOB's (Input/Output Blocks)

IOB's son programables y pueden ser catalogados como:

- Bloque de entrada, el cual puede ser single-data-rate, o double-data-rate (DDR).
- Bloque de salida,, el cual puede ser single-data-rate, double-data-rate, o como buffer de 3 estados.
- Bloque bidireccional, con cualquier configuración posible de entrada y de salida.

Los registros pueden ser de flip-flops de tipo D, o pueden ser también de tipo latch.

CLB's (Configurable Logic Blocks)

Los CLB's estan constituidos por cuatro slices y dos buffers de tres estados cada uno. Cada slice equivale a:

- Dos funciones generadoras (F, G).
- Dos elementos de almacenamiento.
- Compuertas con funciones de aritmética lógica.
- Amplios multiplexores.
- Amplia variedad de funciones

Las funciones generadoras F, G, son configuradas como LUT's de cuatro entradas, o como un registro de corrimiento de 16 bits, o como una memoria distribuida de 16 bits.

Además los elementos que se pueden almacenar, pueden ser guardados por medio de flip-flops de tipo D, o por medio de un latch.

Cada CLB esta conectado con la matriz de ruteo, lo cual permite un fácil acceso con los demás componentes del FPGA.

BSRAM (Block Select-RAM Memory)

Los bloques BSRAM pueden almacenar hasta 18 Kbits de memoria, en bloques RAM de doble puerto, pueden ser programados como memorias de 16 Kbits $\times$ 1 bit hasta memorias de 512×36 bits, dependiendo del tamaño y profundidad de la memoria que se requiera.

Cada puerto es síncrono e independiente, lo que permite los modos de operación de "escritura después de lectura" y "lectura después de escritura". Estos bloques de memoria pueden ser empleados en cascada para obtener memorias de mayor capacidad.

El bloque multiplicador está asociado con cada uno de los bloques BSRAM. El multiplicador es un multiplicador de 18 bits, y está optimizado para efectuar su operación con ayuda de los bloques BSRAM. Sin embargo también se puede emplear el multiplicador de 18 bits de manera independiente sin emplear los bloques BSRAM.

Reloj Global

El dispositivo Virtex-II cuenta con hasta 12 bloques DCM, para generar señales desfasadas de reloj ya sean internas o externas. Cada DCM puede ser empleado para evitar el retraso de propagación de la señal de reloj. Además cuenta con 16 relojes globales, lo que permite tener hasta 8 relojes por cuadrante de la red de CLB. Cada bloque DCM puede controlar hasta cuatro de los 16 relojes globales del dispositivo.

Recursos de Ruteo

Todos los bloques IOB, CLB, BSRAM's, DCM y el multiplicador están conectados a la misma matriz global de interconexión, por lo que tienen acceso entre sí. Hay un total de 16 líneas globales de conexión con los relojes globales (una por cada reloj global), con ocho disponibles por cada cuadrante del sistema interno. Existen además 24 líneas de conexión verticales y horizontales por fila y por columna, así como también se cuenta con una conexión secundaria y local para un ruteo más rápido de la señal.

Apéndice C

Tabla de trinomios irreducibles

La siguiente tabla, muestra polinomios irreducibles mínimos sobre un campo finito $GF(2^m)$. La tabla se enfoca en los polinomios de la forma $z^m + z^k + 1$, es decir en trinomios irreducibles, donde el valor del exponente k es el menor posible, de tal forma que se puedan obtener el menor número de coeficientes por ecuación para calcular el cuadrado o la raíz cuadrada de un elemento de $GF(2^m)$ (que se mostro en las secciones §3.2.4 y §3.2.5).

m	k	$p(z)$	m	k	$p(z)$	m	k	$p(z)$
167	35	$z^{167} + z^{35} + 1$	281	93	$z^{281} + z^{93} + 1$	439	49	$z^{439} + z^{49} + 1$
191	9	$z^{191} + z^{9} + 1$	313	79	$z^{313} + z^{79} + 1$	449	167	$z^{449} + z^{167} + 1$
193	15	$z^{193} + z^{15} + 1$	337	55	$z^{337} + z^{55} + 1$	457	61	$z^{457} + z^{61} + 1$
199	67	$z^{199} + z^{67} + 1$	353	69	$z^{353} + z^{69} + 1$	463	93	$z^{463} + z^{93} + 1$
223	33	$z^{223} + z^{33} + 1$	359	117	$z^{359} + z^{117} + 1$	479	105	$z^{479} + z^{105} + 1$
233	74	$z^{233} + z^{74} + 1$	367	21	$z^{367} + z^{21} + 1$	487	127	$z^{487} + z^{127} + 1$
239	81	$z^{239} + z^{81} + 1$	383	135	$z^{383} + z^{135} + 1$	503	3	$z^{503} + z^{3} + 1$
241	70	$z^{241} + z^{70} + 1$	401	152	$z^{401} + z^{152} + 1$	521	158	$z^{521} + z^{158} + 1$
257	41	$z^{257} + z^{41} + 1$	409	87	$z^{409} + z^{87} + 1$	569	77	$z^{569} + z^{77} + 1$
263	93	$z^{263} + z^{93} + 1$	431	231	$z^{431} + z^{231} + 1$			
271	58	$z^{271} + z^{58} + 1$	433	33	$z^{433} + z^{33} + 1$			

Tabla C.1: Tabla de trinomios irreducibles.

Bibliografía

[1] E. W. Knudsen. Elliptic scalar multiplication using point halving. In *ASIACRYPT '99: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (LNCS 1716)*, volume 274, pages 135–149, London, UK, 1999. Springer-Verlag.

[2] R. Schroeppel. Elliptic curve point ambiguity resolution apparatus and method. International Application Number PCT/US00/31014. filed 9 November 2000.

[3] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2003.

[4] F. Rodríguez-Henríquez and C. K. Koc. On fully parallel karatsuba multipliers for $gf(2^m)$. In *Proceedings of International Conference on Computer Science and Technology CST*, pages 405–410, May 19-21 2003. Acta Press, Cancún México.

[5] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[6] R. Rivest, A. Shamir, and L. Adleman. A method for obtainig digital signatures and public-key cryptosystems. *Communications of the ACM*, 1:120–126, 1978.

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

[8] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology—CRYPTO 85 (LNCS 218)[483]*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[9] FIPS 186-2. Digital Signature Standard DSS. Federal Information Processing Standards Publication 186-2. National Institute of Standards and Technology, 2000.

[10] ANSI X9.62. Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standards Institute, 1999.

[11] ISO/IEC 15946-2. Information technology - security techniques - cryptographic techniques based on elliptic curves - part 2:digital signatures, 2002.

[12] Standard specifications for public-key cryptography, Draft Version D18. IEEE standards documents, Disponible en: http://grouper.ieee.org/groups/1363/, 2004.

[13] J. López and R. Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (LNCS 1717)[262]*, pages 316–327, London, UK, 1999. Springer-Verlag.

[14] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, New York, 1996.

[15] LiDIA Group. Lidia. Disponible En: http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[16] Crypto++. URL: http://www.eskimocom/ weidai/cryplib.html.

[17] J. Goodman and A.P. Chandrakasan. An energy-efficient reconfigurability public-key cryptography processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808–1820, November 2001.

[18] R. Schroeppel, C. L. Beaver, R. Gonzales, R. Miller, and T. Draelos. A low-power design for an elliptic curve digital signature chip. In *Cryptographic Hardware and Embedded Systems-CHES 2002 (LNCS 2523)*, volume 238, pages 366–380, London, UK, 2003. Springer-Verlag.

[19] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila. An end-to-end systems approach to elliptic curve cryptography. In *Cryptographic Hardware an Embedded Systems - CHES 2002, 4th International Workshop*, pages 349–365, 2002. Revised Papers, 2523:349-365, August 2003.

[20] I. K. H. Leung and P. H. W. Leong. A microcoded elliptic curve processor using FPGA technology. *IEEE Transactions on VLSI Systems*, 10(5):550–559, 2002.

[21] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, and J. von zur Gathen. Reconfigurable implementation of elliptic curve crypto algorithms. In *In Proc. of The 9th Reconfigurable Architectures Workshop (RAW-02)*, Fort Lauderdale, Florida, U.S.A., April 2002.

[22] M. Ernst, M. Jung, and F. Madlener. A reconfigurable system on chip implementation for elliptic curve cryptography over $GF(2^n)$. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop (LNCS 2523)*, pages 381–399, Redwood Shores, CA, USA, August 2002.

[23] N. A. Saqib, F. Rodríguez-Henríquez, and A. Díaz-Pérez. A parallel achitecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$. *Elsevier Journal of Microprocessors and Microsystems*, 28:329–339, 2004.

[24] K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. *IEEE Trans. Computers*, 53(8):1047–1059, 2004.

[25] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptative chosen-message attacks. *SIAM. Journal on Computing*, (17):281–308, 1988.

[26] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, (31):469–472, 1985.

[27] National Institute of Standards and Technology. NIST special publication 800-56: Recommendation on key establishment schemes, draft 2.0. Disponible En: http://csrc.nist.gov/CryptoToolkit/tkkeymmt.html, January 2003.

[28] A. Lenstra and editors H. Lenstra. *The Development of the Number Field Sieve, Lecture Notes in Mathematics 1554*. Springer-Verlag, 1993.

[29] D. Gordon. Discret logaritms in $GF(p)$ using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6:124–138, 1993.

[30] J. Pollard. Montecarlo methods for index computacion (mod p). *Mathematics of Computation*, 13:918–924, 1978.

[31] RSA Challenge. avalible at http://www.rsasecurity.com/rsalabs/node.asp?id=2092, 2003.

[32] Certicom challenge. avalible at http://www.certicom.com/index.php, 2003.

[33] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, Boston, MA, 1987.

[34] N. Koblitz. *Introduction to elliptic curves and modular forms*. Springer-Verlag, Berlin Heidelberg, Germany, 1984.

[35] F. Rodríguez-Henríquez. *New Algorithms and Architectures for Arithmetic in $GF(2^m)$ Suitable for Elliptic Curve Cryptography*. PhD thesis, Oregon State Univ., 2000.

[36] A. J. Menezes, Paul C. van Oorschot, and Scott A.Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1996.

[37] W. Trappe and L.C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, Inc, Upper Saddle River, NJ 07458, 2002.

[38] Ç . K. Koç and B. Sunar. Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47(3):1998, March 1998.

[39] B. Sunar and Ç . K. Koç. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):552–527, May 1999.

[40] C. Paar. *Effcient VLSI Architectures for Bit Parallel Computation in Galois Fields*. PhD thesis, University at GH Essen, VDI Verlag, 1994.

[41] M. A. Hasan, M. Z. Wang, and V. K. Bhargava. A modified massey-omura parallel multiplier for a class of finite fields. *IEEE Transactions on Computers*, 42(10):1278–1280, November 1993.

[42] H. Wu and M. A. Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47(8):883–887, August 1998.

[43] G. Orlando and C. Paar. A high performance reconfigurable elliptic curve processor for $GF(2^m)$. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems (LNCS 1965)[263]*, pages 41–56, London, UK, 2000. Springer-Verlag.

[44] J. Lutz. High performance elliptic curve cryptographic co-processor. Master's thesis, University of Waterloo, 2004.

[45] S. M. Hernández-Rodríguez and F. Rodríguez-Henríquez. An FPGA arithmetic logic unit for computing scalar multiplication using the half-and-add method. In *Proceedings of International Conference on Reconfigurable Computing and FPGA's-ReConFig 2005*, Puebla, México, September 28-30 2005.