Centro de Investigación y de Estudios
Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección Computación

Multiplicación Escalar en Curvas de Koblitz: Arquitectura en Hardware Reconfigurable

Tesis que presenta

Juan Manuel Cruz Alcaraz

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. Francisco Rodríguez Henríquez

México, D. F. **Noviembre 2005**

II

Resumen

En 1985, Victor Miller y N. Koblitz propusieron un criptosistema de llave pública análogo al esquema de El Gamal en el cual el campo GF(2^m) es substituido por el grupo de puntos en una curva elíptica definida en un campo finito. La operación básica es la multiplicación escalar, la cual está definida como el múltiplo entero de un punto dado sobre la curva.

Koblitz introdujo una familia de curvas que admiten multiplicaciones escalares especialmente veloces. En 1998 Jerome A. Solinas presenta versiones mejoradas de los algoritmos existentes para multiplicación escalar sobre curvas Koblitz donde es posible generar una expansión del factor escalar conocida como τNAF la cual reemplaza las operaciones de doblado de puntos por exponenciaciones cuadráticas.

En esta tesis se presenta una implementación en hardware reconfigurable de los algoritmos propuestos por Solinas en la cual, basándose en las ventajas presentadas proporcionadas por las curvas Koblitz, se logran desempeños competitivos en comparación con resultados previamente reportados con otras técnicas para el cálculo de multiplicación escalar en ECC.

Se propone una arquitectura de multiplicación compuesta por dos unidades aritméticas independientes (campos finitos y aritmética entera). Un bloque de memoria compartida es usado para comunicar resultados intermedios. Se presentan su implementación en VHDL y su respectiva síntesis sobre un dispositivo FPGA Xilinx XC2V4000 así como resultados en tiempos de ejecución de una multiplicación escalar y recursos utilizados del dispositivo.

Abstract

In 1985, Victor Miller and N. Koblitz proposed a public key cryptosystem analogous to ElGamal scheme, where the finite field GF(2^m) is replaced by the group of points in an elliptic curve defined over a finite field. The basic operation is *scalar multiplication*, defined as an integer multiple of a point in the curve.

Koblitz introduced a familiy of curves which admit especially fast elliptic scalar multiplication. In 1998 Jerome A. Solinas presented improved versions of the available algorithms for computing the scalar multiplication in Koblitz curves. These algorithms represent the scalar factor as an expansion known as τNAF. This expansion allows to replace doubling point operations with quadratic exponentiations.

This thesis presents a reconfigurable hardware implementation based in the algorithms proposed by Solinas. This arquitecture takes advantage of the Koblitz curves features obtaining a competitive performance against previous reported implementations using different techniques for the scalar multiplication on ECC.

The multiplication arquitecture proposed here is composed by two independent arithmetic units (finite fields and integer arithmetic). A shared memory block provides basic communication. The thesis presents a VHDL implementation and its respective synthesis using for a Xilinx XC2V4000 FPGA device. FPGA resources requirements and time needed to compute a scalar multiplication is also presented.

Agradecimientos

A mis padres Pedro y Magdalena. Gracias por nunca perder la fé en mi, ni en la realización de mis metas. Gracias por forjar en mi sus principios y su filosofía de la vida, por todos los sacrificios y desvelos.

A mis hermanos Jacobo y Alba. Gracias por toda la alegría que comparten y me contagian día con día, por recordarme con ella el verdadero valor y propósito de la vida. Gracias por acompañarme en mi niñez y compartir la suya conmigo, sé que nuestros caminos se dividen, pero de algún modo siempre están y estarán conmigo.

A mi familia, gracias por el apoyo incondicional, por el consejo que nunca me faltó.

A mis amigos Abigail, Claudia, Francisco, Grettel, Mónica, Nancy, Paty, Rocío y Ulises. Gracias por todos los recuerdos que ahora compartimos, por su compañia durante estos años, por esa felicidad que irradian y que me hizo posible sobrevivir tantas presiones y desvelos.

A mis profesores, por compartir sus consejos y conocimientos tanto académicos como de la vida diaria.

A Sofia Reza, por su apoyo y confianza, por hacer de la sección un mejor lugar de trabajo para todos sus integrantes.

Al CINVESTAV por proveer los recursos necesarios para el desarrollo de esta tesis.

Al Dr. Julio López Hernández de la Universidad de Campinas de Brasil por sus valiosas observaciones durante el desarrollo de esta tésis.

A Manuel Iván Tostado Ramírez de la Universidad Autónoma de Sinaloa por sus aportaciones en la realización de las estadísticas presentadas en el Apéndice B.

Al proyecto de CONACyt No.45306 el cual financió parcialmente el desarrollo de esta tesis.

Introducción

El avance de la tecnología ha permitido la implementación de algoritmos dedicados a resolver problemas criptográficos básicos como el problema de factorización entera y el problema del logaritmo discreto de una manera bastante eficiente. Los mejores algoritmos conocidos hasta hoy para resolver los problemas de factorización entera y el problema del logaritmo discreto, como lo son el algoritmo de la Criba en Campos Numéricos (*Number Field Sieve*) y el algoritmo de la Rho de Pollard (*Pollard's rho*) respectivamente, tiene un tiempo de cómputo subexponencial.

Las criptografía de curvas elípticas, desde que fué desarrollada en 1985 por Neal Koblitz y Victor Miller, ha demostrado ser capaz de proveer la misma funcionalidad que esquemas tradicionales y ampliamente usados en aplicaciones comerciales como RSA. Más aún, ECC es capaz de proveer los mismos niveles de seguridad que RSA pero con llaves significativamente más pequeñas. Por ejemplo, se acepta por lo general que una llave de curva elíptica de 160 bits provee el mismo nivel de seguridad que una llave RSA de 1024 bits [1].

El contraste entre los desempeños relativos entre ambos esquemas criptográficos ha motivado a diversos grupos de trabajo para desarrollar algoritmos eficientes. De la misma manera, se han publicado implementaciones de dichos algoritmos en una gran diversidad de plataformas y diseños especializados. Pueden encontrarse implementaciones en software para diversos procesadores comerciales y diseños de microprocesadores dedicados al cálculo de las operaciones básicas para la ley de grupo en curvas elípticas optimizados para diversos algoritmos.

De aquí el interés de presentar una arquitectura en hardware reconfigurable que presente buen desempeño en tiempos de cómputo y/o recursos de hardware utilizados.

En la presente tesis, se presenta un diseño en hardware reconfigurable de

los algoritmos propuestos por Solinas en [2] y [3]. Los algoritmos propuestos por Solinas son algoritmos análogos a métodos tradicionales de expansión NAF con la característica del reemplazo de doblados de puntos elípticos por exponenciaciones cuadráticas sobre campos finitos.

Las plataformas de hardware presentan excelentes desempeños debido a que la aritmética GF(2^m) es posible implementarla eficientemente. Debido a la dependencia que la eficiencia de la multiplicación escalar tiene con los algoritmos de aritmética modular $GF(2^m)$, es parte del problema a resolver presentar diseños en hardware reconfigurable de algoritmos para aritmética de campos finitos $GF(2^m)$ eficientes en recursos y tiempo de cómputo. Los algoritmos presentados originalmente son analizados y especializados para la plataforma de hardware propuesta para optimizar el rendimiento del sistema.

El contenido de la tesis ha sido organizado en 6 Capítulos. El Capítulo 1 presenta una introducción a criptografía de llave pública y diversos esquemas de firma digital para enfocarse finalmente a criptografía de curvas elípticas. El Capítulo 2 provee las bases matemáticas necesarias para la definición formal de criptografía de curvas elípticas. Define formalmente la operación de multiplicación escalar elíptica y diversos métodos existentes para el cálculo eficiente de dicha operación. Finalmente define y expone las características principales de las curvas Koblitz y su aplicación en el desarrollo del método de multiplicación propuesto por Solinas. El Capítulo 3 presenta un análisis de los algoritmos propuestos por Solinas para la multiplicación escalar y su adaptación y optimización para una plataforma de hardware. El Capítulo 4 describe en detalle la arquitectura propuesta y sintetizada sobre el dispositivo FPGA Xilinx XC2V4000 y el diseño de módulos de aritmética de campo finito eficientes. El Capítulo 5 presenta la evaluación de los resultados obtenidos en la etapa de síntesis del diseño propuesto en cuanto a unidades básicas del dispositivo FPGA utilizadas y tiempos de cómputo necesarios para el cómputo de una multiplicación escalar. Este último se presenta en microsegundos y cantidad de ciclos de reloj. Finalmente se presenta un estudio comparativo con diversos trabajos considerados similares a nuestra implementación. El Capítulo 6 presenta las conclusiones obtenidas durante el proceso de diseño y durante las comparaciones de los resultados finales. Diversos puntos considerados como posibles extenciones futuras del trabajo son comentados.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Criptografía de Curvas Elípticas

Desde que los sistemas criptográficos de curvas elípticas fueron propuestos por primera vez por N. Koblitz [4] y V. Miller [5], se ha publicado una gran cantidad de trabajos con el propósito de mejorar el desempeño de sus algoritmos en sistemas de seguridad de llave privada. Este capítulo presenta una base conceptual para el desarrollo de esta tesis. Introduce conceptos básicos de criptografía y de curvas elípticas. Presenta la aplicación del problema del logaritmo discreto generalizado en los servicios básicos de la criptografía y sus diferentes aplicaciones.

1.1. Servicios Básicos en Criptografía

La criptografía engloba un amplio conjunto de diseños y análisis de técnicas matemáticas para lograr una comunicación segura en presencia de adversarios maliciosos. Ofrece un conjunto de servicios sobre un modelo básico de comunicaciones tal como se ilustra en la Figura 1.1.

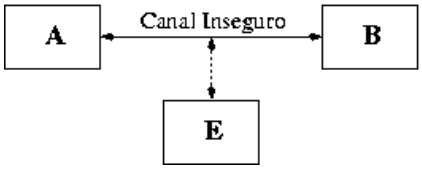


Figura 1.1: Modelo Básico de Comunicación

En la Figura 1.1 las entidades A y B se comunican por un canal inseguro. Se supone que dicha comunicación se realiza en la presencia de un adversario I cuyo objetivo es violar los servicios de seguridad en dicha comunicación.

Los servicios que provee la criptografía en dicho escenario son confidencialidad, integridad de información, autenticación de origen de datos y de las entidades participantes y no repudio. Dichos servicios pueden ser definidos como sigue [1]:

1. *Confidencialidad:* Mantener el contenido de la comunicación secreta para cualquier entidad no autorizada para verla.

2. *Integridad de Información:* Asegurar que la información en la comunicación no ha sido alterada por medios no autorizados.

3. *Autenticación del origen de los datos:* Poder corroborar la fuente de origen de los datos.

4. *Autenticación de las entidades:* Poder corroborar la identidad de las entidades participantes en la comunicación de forma autorizada.

5. *No Repudio:* Impedir la negación de cualquier acto implicado en la comunicación por cualquiera de las entidades participantes.

En terminología criptográfica el mensaje intercambiado entre las entidades A y B es llamado *texto en claro*. Codificar el contenido del mensaje de tal forma que el contenido esté oculto de entidades ajenas al circuito de comunicación autorizado (adversarios como la entidad I) es llamado *cifrado*. El mensaje cifrado es llamado *cifra*. El proceso de recuperar el texto plano a partir de la cifra es llamado *descifrado*. Los procesos de cifrado y descifrado hacen uso de un elemento llamado *llave*. La única forma de recuperar el texto plano a partir de la cifra es a través del conocimiento de la llave correcta.

1.2. Criptografía de Llave Pública

Los métodos criptográficos pueden ser clasificados en dos grupos principales, criptografía de llave secreta y criptografía de llave pública. La criptografía de llave secreta se basa en principios matemáticos y criptográficos ampliamente estudiados y en funciones primitivas relativamente simples como son: permutaciones, rotaciones, substituciones, corrimientos, operaciones

XOR y generación de secuencias pseudo-aleatorias. En cambio la criptografía de llave pública se basa en la dificultad de resolver computacionalmente ciertos problemas matemáticos [6].

La principal característica de este esquema criptográfico es que la llave utilizada para el cifrado es diferente de la utilizada para el descifrado o la llave para verificar la procedencia de un documento es diferente que la llave usada para autentificar su procedencia (*firmado*). Las llaves son llamadas pública y privada respectivamente. Dichas llaves son seleccionadas de tal forma que el problema de obtener la llave privada a partir del conocimiento de su correspondiente llave pública sea equivalente a un problema computacionalmente intratable. Los principales problemas matemáticos utilizados para implementar servicios criptográficos con llave pública son:

1. El problema de la factorización de enteros, base para el esquema RSA de cifrado y firma.

2. El problema del logaritmo discreto utilizado en el esquema criptográfico de ElGamal y sus variantes como el Algoritmo de Firma Digital (*Digital Signature Algorithm, DSA*).

3. El problema del logaritmo discreto en curva elípticas utilizado en el esquema criptográfico de curvas elípticas.

Los algoritmos más usados bajo este esquema son RSA, DSA, El Gamal y Criptografía de Curvas Elípticas (*Elliptic Curve Cryptography, ECC*).

Para que A establezca una comunicación segura con B es necesario que conozca la llave pública de B, e_B, la cual es de dominio público.En cambio B es la única entidad que posee la llave privada correspondiente d_B. Si A quiere enviar un mensaje confidencial a B entonces usa la función de cifrado ENC en el mensaje m para obtener la cifra $c = ENC_{e_B}(m)$. La única forma de recuperar el mensaje original es conociendo la correspondiente llave privada y usando la función de descifrado $m = DEC_{d_B}(c)$.

Para que A pueda autenticarse como el autor del mensaje usa su llave privada, d_A, de la cual sólo esta entidad tiene conocimiento, en la función de firmado $s = SIGN_{d_A}(m)$ para obtener la firma del documento. La entidad B utiliza el mensaje m, su firma s y la llave pública de A, e_A, para verificar si A fue realmente el autor del mensaje. Dado que sólamente A es capaz de firmar con su llave privada, esta entidad no puede negar ser el autor del mensaje, obteniendo también el servicio de no repudio.

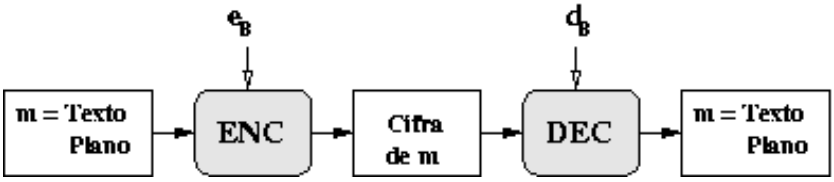


Figura 1.2: Modelo Básico de Llave Pública

1.3. Firma Digital RSA

Una firma digital es un medio para anexar un identificador a un documento y evitar falsificaciones. La firma identifica al autor original del documento y asegura que dicho documento no se ha falsificado. Un ejemplo es su aplicación en la identificación de origen de correos o la autorización para cargar en memoria un programa de origen externo al sistema.

El algoritmo más utilizado en aplicaciones comerciales es RSA. RSA fué nombrado así por las siglas de sus creadores Rivest, Shamir y Adleman, los cuales propusieron su algoritmo en 1977. RSA trabaja bajo el esquema de llave pública descrito en la Figura 1.2 .

El sistema descrito en la Figura 1.2 no permite que nadie, excepto el poseedor legítimo de la llave privada de B, sea capaz de leer el contenido del mensaje, pero no garantiza que A sea el autor de dicho mensaje. El algoritmo RSA es simétrico, así que tanto la llave pública como privada pueden cifrar un texto. Si se cifra un texto con la llave privada, cualquier poseedor de la correspondiente llave pública es capaz de obtener el texto plano de su cifra, aunque solamente el poseedor de la llave privada es capaz de realizar dicha cifra.

Si A necesita enviar un mensaje a B, primero será necesario calcular un *valor Hash* o una suma de comprobación (*checksum*). Existen varios algoritmos estándares para obtener dichos valores Hash. A debe de cifrar el valor Hash con su llave privada. El valor Hash es anexado al mensaje y enviado a B. Cuando B recibe el mensaje, la entidad descifra el valor Hash con la llave pública de A y calcula el valor Hash del mensaje original. Si éstos concuerdan, entonces el mensaje fué realmente enviado por A. Además, dado que los valores Hash son idénticos, se concluye que el mensaje no ha sido modificado desde su creación.

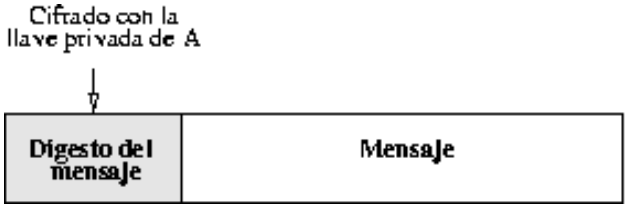


Figura 1.3: Proceso Básico de la Firma Digital

1.3.1. Generación de Llaves RSA

Un par de llaves RSA se genera a partir del Algoritmo 1. La llave pública consiste del par de enteros (n, e), donde el *módulo* RSA n, es el producto de dos números primos p y q de la misma longitud en bits, los cuales han sido seleccionados al azar y deben mantenerse secretos. El número e es llamado *exponente de cifrado* y cumple con $1 < e < \phi$ y $\gcd(e, \phi) = 1$ donde $\phi = (p-1)(q-1)$. La llave privada d es llamada *exponente de descifrado*, y es el entero que cumple $1 < d < \phi$ y $ed \equiv 1 (\text{mod } \phi)$. El problema de determinar la llave d a partir de la llave pública (n, e) es computacionalmente equivalente al problema de determinar los factores p y q de n el cual es mejor conocido como el *problema de factorización de enteros*.

Procedimiento 1 Generación de llaves RSA
Entrada: Parámetro de Seguridad l.
Salida: Llaves RSA pública (n, e) y privada d.
1: Seleccionar al azar dos números primos p y q de la misma longitud: $\frac{l}{2}$ bits.
2: Calcular $n = pq$ y $\phi = (p-1)(q-1)$.
3: Seleccionar un número arbitrario e tal que $1 < e < \phi$ y $\gcd(e, \phi)$=1.
4: Calcular el entero d que satisface $1 < d < \phi$ y $ed \equiv 1 \pmod{\phi}$.
5: Regresar (n, e, d).

1.3.2. Esquema de Firma Digital

Tanto los procesos de cifrado como de firma digital se basan en la identidad:

$$m^{ed} = m \pmod{n} \tag{1.1}$$

para todos los enteros m. El procedimiento de firma y verificación se muestran en los Algoritmos 2 y 3. El autor A del mensaje m calcula el valor Hash $h = H(m)$ usando una función criptográfica H, donde h será la huella digital de m. Entonces el autor calcula la firma s elevando a la d-ésima potencia. El autor transmite el mensaje y la firma como se muestra en la Figura 1.3. La entidad receptora B recupera el valor Hash a partir de la firma elevando, con su llave pública, la firma a la e-ésima potencia módulo n. Note que de la expresión 1.1 se concluye que $s^e \equiv (h^d)^e \equiv h \pmod{n}$. B vuelve a calcular el valor Hash a partir del algoritmo hash $h = H(m)$. Si ambos valores concuerdan, la firma es aceptada.

Procedimiento 2 Generación de Firma Digital RSA

Entrada: Llave pública RSA (n, e), llave privada RSA d, mensaje m.
Salida: Firma digital s.

1: Calcular $h = H(m)$ donde H es una función *hash*.
2: Calcular $s = h^d \bmod n$.
3: Regresa s.

Procedimiento 3 Verificación de Firma Digital RSA

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

1: Calcular $h = H(m)$.
2: Calcular $h' = s^e \bmod n$.
3: **if** $h = h'$ **then**
4: Acepta firma digital.
5: **else**
6: Rechaza firma digital.

1.4. Problema del Logaritmo Discreto

El problema del logaritmo discreto puede ser descrito en términos de un grupo cíclico finito. Es necesario introducir conceptos elementales de teoría de grupos para obtener una generalización del concepto de logaritmo discreto.

1.4.1. Grupos

Un *grupo abeliano* $(G, *)$ consiste de un conjunto G con una operación binaria $* : G \times G \leftarrow G$ que satisface las propiedades de conmutación, asociatividad, elemento identidad, e inverso [1].

La operación de grupo es usualmente llamada adición $(+)$ o multiplicación $(\cdot)$. En los grupos aditivos el elemento identidad se denota con 0 y el inverso de a se denota $-a$. En los grupos multiplicativos en cambio, el elemento identidad se denota con 1 y el inverso multiplicativo del elemento a se denota a^{-1}. El grupo será llamado *grupo finito* si el conjunto G es finito. La cantidad de elementos en G es llamado el *orden* de G.

Se define el conjunto $\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$ de los enteros módulo p y la operación $+$ como la adición de enteros módulo p. El grupo $(\mathbb{F}_p, +)$ es un grupo aditivo de orden p con 0 como su elemento identidad. De la misma manera se puede definir el conjunto $\mathbb{F}_p^*$ como el conjunto de todos los elementos de $\mathbb{F}_p$ distintos de cero y el operador $\cdot$ como la multiplicación módulo p entre los elementos de $\mathbb{F}_p^*$. El grupo multiplicativo $(\mathbb{F}_p^*, \cdot)$ tiene como elemento identidad a 1 y es de orden p-1. La terceta de elementos $(\mathbb{F}_p, +, \cdot)$ es llamada *campo finito* el cual es denotado de manera abreviada como $\mathbb{F}_p$.

Sea G un grupo multiplicativo de orden n y $g \in G$. Si t es el entero positivo más pequeño que cumple $g^t = 1$, t es llamado el *orden* del elemento g; este t existe siempre y, aún más, es un divisor de n. El conjunto generado por las potencias de t, $\langle g \rangle = \{g^i : 0 \leq i \geq t-1\}$ es un grupo bajo la misma operación que G y es llamado *subgrupo cíclico de G generado por g*. Las mismas definiciones pueden ser aplicadas a un grupo aditivo de manera análoga. Si definimos tg como la adición entre t copias de g, entonces decimos que el orden t de un elemento $g \in G$ es el elemento más pequeño que cumple que $tg = 0$ y $\langle g \rangle = \{ig : 0 \leq i \geq t-1\}$ es el subgrupo cíclico de G generado por g. Finalmente si existe un elemento $g \in G$ de orden n, entonces G es llamado un *grupo cíclico* y g es llamado *generador de G*.

1.4.2. Logaritmo Discreto Generalizado

Una vez definido una grupo cíclico $(G, \cdot)$ sobre un operador $\cdot$ de orden n con un generador g, es posible establecer un esquema de llave pública. Los parámetros g y n describen el grupo y la generación de sus elementos y son llamados *parámetros de dominio*. La llave pública es escogida al azar entre los n elementos pertenecientes al conjunto G potencialmente generados por

g, $x \in [1, n-1]$. La llave privada y es generada como el x-ésimo elemento generado por g, $y = g^x$. El problema de determinar la llave privada y a partir de la llave pública x es conocido como el *problema del logaritmo discreto*.

Es necesario notar que cualquier grupo cíclico del mismo orden n son esencialmente los mismos, diferenciándose por la representación de sus elementos. La elección de la representación de estos elementos puede llevar a mejoras en recursos de velocidad y/o memoria en los algoritmos aplicados en el cálculo de su aritmética y por lo tanto en la resolución del problema del logaritmo discreto.

Los grupo más comunmente utilizados para la definición de esquemas de firma digital son los subgrupos cíclicos multiplicativos sobre campos finitos, y los subgrupos cíclicos aditivos sobre curvas elípticas.

1.5. Firma Digital DSA

El esquema de firma digital DSA (por las siglas en inglés *Digital Signature Algorithm*) se basa en el esquema propuesto por ElGamal en 1984 fundamentado en el problema del logaritmo discreto con un esquema de llave pública. Muchas variantes han sido propuestas. El esquema presentado por ElGamal fué propuesto en 1991 por el Instituto Nacional de Estándares y Tecnologías de los Estado Unidos (NIST *U.S. National Institute of Standards and Technology*) y registrado como la firma digital estándar (DSS por las siglas en inglés *Digital Signature Standard*).

1.5.1. Generación de Llaves

Las llaves privada y pública se basan en el problema del logaritmo discreto. Los parámetros p, q y g son de dominio público y especifican un grupo multiplicativo módulo p. El parámetro $g \in [1, p-1]$ especifica un generador de un subgrupo cíclico multiplicativo $\langle g \rangle$ de orden q, esto es q es primo divisor de $p-1$ y es el entero positivo más pequeño que cumple $g^q \equiv 1$. Las llaves son generadas dentro del subgrupo cíclico especificado por los parámetros anteriores como se muestra en el Algoritmo 4.

La llave privada es seleccionada como un número aleatorio x entre los elementos pertenecientes al subrupo $x \in \langle q \rangle \in [1, q-1]$ y su correspondiente llave pública es el x-ésimo elemento generado por g $y = g^x \bmod p$ como se muestra en el Algoritmo 5. El problema de determinar x a partir

de los parámetros de dominio (p, q, g) y la llave pública x es el problema del logaritmo discreto.

Procedimiento 4 Generación de paramétros de dominio DSA

Entrada: Parámetros de seguridad l y t.

Salida: Parámetros de dominio (p, q, g).

1: Seleccionar un número primo q de t bits y otro número primo p de l bits tal que q divida a $p - 1$.
2: Seleccionar un elemento g de orden q:
3: Seleccionar un número de manera aleatoria $h \in [1, p - 1]$ y calcular $g = h^{\frac{p-1}{q}} \bmod p$.
4: Si $g = 1$ entonces repetir el paso anterior.
5: **Regresa** (p, q, g).

Procedimiento 5 Generación de llaves DSA

Entrada: Parámetros de dominio p, q, g.

Salida: Llave privada x y llave pública y.

1: Seleccionar aleatoriamente $x \in [1, q - 1]$.
2: Calcular $y \equiv g^x \bmod p$.
3: **Regresa** (y, x).

1.5.2. Algoritmo de Firma Digital DSS

Una vez que se han generado la pareja de llaves pública y privada (y, x), una entidad A genera la firma $S = (r, s)$ de un mensaje m primero obteniendo un número aleatorio $k \in [1, q-1]$ el cuál debe ser secreto e incluso puede ser destruido al finalizar el proceso de firma. En seguida, se obtiene el k-ésimo elemento generado por g, $T = g^k \bmod p$, y $r = T \bmod q$. El mensaje m es procesado con un algoritmo hash H para obtener su valor Hash $h = H(m)$. El siguiente elemento de la firma es generado:

$$s \equiv k^{-1}(h + xr) \bmod q \tag{1.2}$$

La firma digital está compuesta por el vector (s, r). Donde solamente el poseedor de la llave privada x y el número aleatorio k es capaz de generarla.

La entidad receptora B verificará la validez de la firma a través de verificar la igualdad 1.2. Sin embargo, al no poseer la llave privada ni el número aleatorio k, es imposible verificarla directamente. La verificación se hace a partir de la siguiente observación:

$$k \equiv s^{-1}(h + xr) \bmod q. \tag{1.3}$$

Ahora utilizando el elemento generador g en ambos lados de la expresión 1.3:

$$g^k \equiv g^{s^{-1}h} g^{xs^{-1}r} \bmod p \tag{1.4}$$

Finalmente recordando $T = g^k \bmod p$ y $g^x \bmod p$ es la llave pública:

$$T \equiv g^{hs^{-1}} y^{rs^{-1}} \bmod p \tag{1.5}$$

La firma puede ser verificada mediante el cálculo de T a partir de los parámetros de dominio (p, q, g), la llave pública y y la forma r, s para comprobar la igualdad $r = T \bmod q$.

Los algoritmos de generación y verificación de firma DSA se presentan en los Procedimientos 6 y 7.

Procedimiento 6 Generación de Firma DSA

Entrada: Parámetros de dominio (p, q, g), llave privada x, mensaje m.
Salida: Firma (r, s).

1: Seleccionar aleatoriamente $k \in [1, q-1]$.
2: Calcular $T = g^k \bmod p$.
3: Calcular $r = T \bmod q$. Si $r = 0$ Repetir desde el paso 1.
4: Calcular $h = H(m)$.
5: Calcular $s = k^{-1}(h + xr) \bmod q$. Si $s = 0$ repetir desde el paso 1.
6: **Regresa** (r, s).

1.6. Firma Digital de Curvas Elípticas

Se define una *curva elíptica* E sobre los números reales como el conjunto de todos los puntos $P = (x_p, y_p)$ que cumplen con la equación:

$$y^2 = x^3 + ax + b, \tag{1.6}$$

Procedimiento 7 Verificación de Firma DSA

Entrada: Parámetros de dominio (p, q, g), llave pública y, mensaje m y firma (r, s).

Salida: Aceptar o Rechazar la firma digital.

1: Verificar que r, s son enteros en el intervalo $[1, q-1]$. Si la verificación falla, regresar "Rechazo de Firma".
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 $r = r'$ regresar "Firma Aceptada", sino regresar "Rechazo de Firma".

donde a y b satisfacen $4a^3 + 27b^2 \neq 0$. Se dice que el *punto al infinito* ∞ está sobre la curva. Para definir una curva elíptica sobre $GF(2^m)$ la ecuación es ajustada para una representación binaria:

$$y^2 + xy = x^3 + ax^2 + b \tag{1.7}$$

con $a, b \in GF(2^m)$ y $b \neq 0$. La curva elíptica incluye todos los puntos (x, y) que satisfacen la ecuación sobre $GF(2^m)$ y el punto al infinito. El conjunto de los puntos pertenecientes a la curva E se denota por $E(\mathbb{F}_{2^m})$.

Es posible definir la adición entre puntos sobre la curva (x_1, y_1) y (x_2, y_2) para obtener un tercer punto sobre la curva (x_3, y_3). Con esta adición se tiene un grupo abeliano aditivo con ∞ como su elemento identidad.

1.6.1. Parámetros de Dominio en Curvas Elípticas

Una curva elíptica E queda descrita por un conjunto de parámetros, de la misma manera, el desarrollo de un esquema de firma digital basado en el problema del logaritmo discreto necesita la descripción de un subgrupo cíclico sobre el cual generar elementos. La elección de los parámetros de dominio es fundamental para ofrecer un nivel de seguridad óptimo.

El campo finito sobre el cual se basa la definición de la ecuación de la curva y la adición entre puntos es un campo primo, un campo binario o un campo de extensión óptima.

Los *parámetros de dominio* necesarios para establecer un esquema de llave pública basado en el problema del logaritmo discreto son los siguientes[1]:

1. El *orden* del *campo finito base* $\mathbb{F}_q$, q.
2. Los coeficientes $a, b \in \mathbb{F}_q$ que definen la ecuación de la curva elíptica E sobre $\mathbb{F}_q$.
3. Una pareja de elementos $(x_p, y_p) \in \mathbb{F}_q$ representando un punto $P = (x_p, y_p) \in E(\mathbb{F}_q)$. P tiene un orden primo y, al servir de generador para un subgrupo cíclico aditivo, es llamado *punto base*.
4. El *orden* n de P.
5. El *cofactor* $h = \#E(\mathbb{F}_q)/n$.

1.6.2. Generación de Llaves

Sea $P \in E(\mathbb{F}_q)$ con orden n, donde E es una curva elíptica. Entonces es posible generar un subgrupo cíclico $\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (n-1)P\}$.

El campo q, la ecuación de la curva E , el punto P y el orden de n son parámetros de dominio público. La llave privada es un entero d seleccionado al azar entre el intervalo cerrado $[1, n-1]$ y la llave pública correspondiente es el punto $Q = dP$. El problema de determinar d a partir de los puntos P y Q es el *problema del logaritmo discreto en curvas elípticas*.

Procedimiento 8 Generación de llaves con curvas elípticas

Entrada: Parámetros de dominio público de la curva (q, E, P, n).
Salida: Llave pública Q y llave privada d.

1: Selecciona d al azar del intervalo cerrado $1, n-1$
2: Calcula $Q = dP$.
3: Regresa (Q, d).

1.6.3. Algoritmo de Firma Digital de Curvas Elípticas

El Algoritmo de Firma Digital de Curvas Elípticas (ECDSA por las siglas en inglés *Elliptic Curve Digital Signature Algorithm*), es el algoritmo análogo de DSA con curvas elípticas. Es el esquema de firma basado en curvas elípticas más ampliamente estandarizado [7].

En los Algoritmos 9 y 10, H es una función hash cuyas salidas deben tener una longitud en bits no mayor que n.

Procedimiento 9 Generación de Firma Digital ECDSA

Entrada: Parámetros de Dominio: (q,a,b,P,n,h).

Salida: Firma r, s,

1: Selecciona k al azar en el intervalo cerrado $[1, n-1]$.
2: Calcula $kP = (x_1, y_1)$ y convertir x_1 en el entero $\overline{x}_1$.
3: Calcular $r = \overline{x}_1 \bmod n$. Si $r = 0$ regresar al paso 1.
4: Calcular $e = H(m)$.
5: Calcular $s = k^{-1}(e + dr) \bmod n$.Si $s = 0$ regresar al paso 1.
6: Regresar (r, s).

Procedimiento 10 Verificación de Firma ECDSA

Entrada: Parámetros de Dominio: (q,a,b,P,n,h).

Salida: Aceptación o rechazo de la firma digital.

1: Verificar que r y s sean enteros en el intervalo $[1, n-1]$. Si cualquier verificación falla regresar "Rechazo de la Firma".
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 $X = \infty$ regresar Rechazo de la Firma".
7: Convertir la coordenada en x x_1 de X a un entero $\overline{x}_1$.
8: Calcular $v = \overline{x}_1 \bmod n$.
9: Si $v = r$ regresar "Firma Aceptada"si no regresa "Rechazo de la Firma".

Si un mensaje m tiene una firma (r, s) legítima, entonces $s \equiv k^{-1}(e + dr) \mod n$. Esto implica $k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}d \equiv we + wdr \equiv u_1 + u_2 \pmod{n}$. Por lo tanto $X = u_1P + u_2Q = (u_1 + u_2d)P = kP$, y entonces $v = r$ como se requería.

1.7. Aplicaciones Criptográficas

El gran auge del cual han gozado las comunicaciones electrónicas en los últimos años ha generado una gran demanda por los servicios que proporciona la criptografía. Las redes de computadoras se han popularizado y en nuestros días encontramos una gran cantidad de aplicaciones sobre ellas, todas basadas en el intercambio de información. Estas aplicaciones se extienden desde intercambio de archivos, hasta comercio e incluso banca electrónica.

Entre más complejo es el servicio ofrecido, la información intercambiada es más valiosa. Las aplicaciones criptográficas ya no están marginadas al campo del espionaje industrial o internacional.

La siguiente lista [8] pretende enumerar algunas de las aplicaciones más comunes en nuestros días que requieren de servicios criptográficos para ofrecer seguridad a sus clientes. La lista no pretende abarcar las aplicaciones existentes en su totalidad, sino dar una visión general de la amplia gama de las aplicaciones posibles para los servicios criptográficos.

- *Correo Electrónico*: El mantener la privacidad de un mensaje y corroborar el origen de ellos ha sido una necesidad en redes grandes como Internet, sin embargo se empieza a popularizar su uso en redes internas.

- *World Wide Web*: La WWW es la herramienta más popularizada para accesar a bases de datos, administrar sistemas de forma remota e incluso para aplicaciones comerciales, entre otros. Se presenta como una necesidad el ofrecer la seguridad de la gran diversidad de información como una característica intrínseca de la WWW.

- *Conexiones cliente-servidor*: Las redes de computadoras incrementan el tráfico en las redes al manejar esquemas de centralización del proceso de la información. Es necesario que, los servicios ofrecidos por el servidor, como elemento centralizador de cómputo, garantize la seguridad de sus procedimientos.

- *Redes privadas virtuales*: Compañias con sucursales nacionales e internacionales acostumbran acoplar las redes internas de cada sucursal, a través de redes globales como la Internet. La información debe garantizarse libre de acceso no autorizado mientras viaja por redes ajenas a la compañía y regresar a sus permisos de acceso originales una vez que se encuentre en redes internas autorizadas.

- *Dinero Electrónico*: La seguridad proporcionada por las técnicas actuales permite el intercambio de valores a través de la red, a partir de documentos electrónicos para realizar pagos de manera remota.

- *Banca Electrónica*: Las instituciones financieras requieren del manejo de valores a través de redes de computadoras en escalar enormes. La criptografía permite garantizar el manejo de los recursos financieros de una institución bancaria de manera segura.

- *Administración Remota*: La administración de equipos y redes de cómputo de manera remota requiere de servicios criptográficos para garantizar el acceso autorizado a dichos recursos de cómputo.

1.8. Conclusiones

La gran cantidad y diversidad de aplicaciones que requieren servicios criptográficos se encuentra en una etapa de expansión. Los algoritmos de firma digital proveen servicios de integridad de información, autenticación del origen de los datos y no repudio. Dado que la firma digital provee una amplia variedad de servicios, es necesario presentar métodos optimizados para realizarla. El avance de la tecnología permite implementaciones eficaces de los diversos algoritmos de firma digital, pero la misma tecnología permite que los ataques de entidades externas hacia los algoritmos criptográficos se vuelvan también más eficaces. Esto impulsa a la comunidad criptográfica a buscar métodos no solamente eficaces, sino también eficientes para ofrecer servicios criptográficos robustos. La criptografía de curvas elípticas ofrece métodos que presentan una vasta gama de características explotables para ofrecer implementaciones altamente eficientes. En la presente tesis se ha explorado una familia de curvas y sus características particulares para ofrecer una arquitectura de cómputo optimizada sobre un algoritmo para el cálculo

de la multiplicación por un escalar en curvas elípticas para una plataforma de hardware reconfigurable.

Capítulo 2

Conceptos Matemáticos

2.1. Campos Finitos

Un *grupo* $\langle G, * \rangle$ es un objeto matemático abstracto el cual consiste en un conjunto G y una operación $*$ definida en parejas de elementos en G.

$$* : G \times G \leftarrow G, (a, b) \mapsto a * b \tag{2.1}$$

Un grupo es llamado *abeliano* si cumple las siguientes condiciones:

1. Cerradura: $\forall a, b \in G : a * b \in G$.
2. Asociatividad: $\forall a, b, c \in G : (a * b) * c = a * (b * c)$.
3. Conmutatividad: $\forall a, b \in G : a * b = b * a$.
4. Elemento Neutro: $\exists 0 \in G, \forall a \in G : a + 0 = a$.
5. Elementos Inversos: $\forall a \in G, \exists b \in G : a + b = 0$

Un *anillo* $\langle R, +, * \rangle$ está conformado por un conjunto R con dos operaciones definidas en sus elementos, aqui denotador por $+$ y $*$. Este conjunto debe de cumplir con las siguientes condiciones [6]:

1. La estructura $\langle R, + \rangle$ es un grupo *abeliano*.
2. La operación $*$ es *cerrada* y asociativa sobre R. Existe un elemento neutro para $*$ en R.

3. Las dos operaciones + y $*$ están relacionadas por la *ley distributiva*. $\forall a, b, c \in R : (a + b) * c = (a * c) + (b * c)$.

4. Un anillo $\langle R, +, * \rangle$ es llamado *conmutativo* si la operación $*$ es conmutativa.

Por ejemplo, los números enteros, racionales, reales y complejos conforman anillos con las operaciones *(*multiplicación*) y + (*adición*). Además, un elemento x de un anillo se dice *invertible* si x tiene un inverso multiplicativo en R. Esto es, si $e \in R$ es el elemento neutro sobre el operador multiplicativo, entonces existe u tal que $xu = ux = e$.

Una estructura $\langle F, +, * \rangle$ es llamada un *campo* si F es un anillo en el cuál la multiplicación es conmutativa y cada elemento , excepto 0, tiene un inverso multiplicativo. Es posible definir un campo F con respecto a la adición y la multiplicación si:

- F es un grupo conmutativo con respecto a la adición.
- $F - \{0\}$ es un grupo conmutativo con respecto a la multiplicación.
- Se mantiene la ley distributiva de los anillos.

La *substracción* de un elemento está definida en términos de la adición: $\forall a, b \in F, a - b = a + (-b)$ donde $-b$ es el elemento inverso aditivo de b. De manera análoga se define la *división* con respecto a la multiplicación: $\forall a, b$ con $b \neq 0, a/b = a * b^{-1}$ donde b^{-1} es el elemento inverso multiplicativo de b. Un campo se dice *finito* si tiene un número finito de elementos. Los campos finitos más comunmente usados en criptografía son los campos primos, los campos binarios y los campos óptimos extendidos.

El *orden* de un campo finito es el número de elementos en el campo. Un campo finito F de orden q solamente puede existir si q es una potencia de un número primo $q = p^m$. El número primo p es llamado la *característica* de F. Si $m = 1$ el campo es llamado un *campo primo*. Si $m \geq 2$, F es llamado un *campo extendido*. Todos los campos con el mismo orden q representan la misma estructura matemática, pero con diferente representación de sus elementos, por lo que se dice que son *isomorfos* entre ellos.

2.1.1. Aritmética Modular

Sea un entero arbitrario n, es posible representar cualquier entero i como el producto de n por un cociente q más un residuo r en el intervalo $[0, n-1]$: $i = qn + r$. El conjunto de todos los posibles residuos para el entero n es $R_n = 0, 1, \ldots, n-1$.

La aritmética entre residuos módulo n es llamada *aritmética modular*. Observemos que si $r = i \bmod n$ y $s = j \bmod n$, entonces r y s pueden ser representados como residuos módulo n: $r = i - qn$ y $s = j - tn$; para algún cociente q y t. Entonces:

$$r + s = i + j - (q + t)n; \tag{2.2}$$

$$rs = ij - (qj + ti)n + qtn^2. \tag{2.3}$$

Por lo tanto $(r + s) \bmod n = (i + j) \bmod n$ y $rs \bmod n = ij \bmod n$, por lo tanto la suma y producto modulares pueden definirse como:

$$r \oplus s = (r + s) \bmod n \tag{2.4}$$

$$r * s = (rs) \bmod n \tag{2.5}$$

2.1.2. Campos Finitos Primos

El conjunto de todos los residuos $R_n = \{0, 1, \ldots, n-1\}$ módulo n, para cualquier entero positivo n, y la adición modular $\oplus$ conforman un grupo abeliano. Este grupo es nombrado *enteros módulo n* y se denota $\mathbb{Z}_n$.

Un grupo puede ser llamado *cíclico finito* si existe un elemento g de este grupo el cual puede expresar cualquier otro elemento en G a través de sumas consecutivas de g, $g \oplus \ldots \oplus g$. Todos los elementos de G son generados a través de sumas sucesivas de g y en la secuencia: $\{g, g \oplus g, g \oplus g \oplus g, \ldots\}$. Las sumas sucesivas entre i copias de g las podemos escribir como ig, donde i es un entero positivo y g un elemento del grupo: $ig = g \oplus \ldots \oplus g$

Dado que g genera a G es llamado elemento *generador* y existirá $ig = 0$ para alguna i. Sea n el número menor que cumple $ng = 0$, por lo tanto $ig \neq 0$ para $1 \leq i \leq n-1$. Todos los elementos generados en dicho intervalo serán diferentes porque si sumamos jg a cualquier elemento generado en el intervalo $ig \neq 0$ para $1 \leq i \leq n-1$, observamos que $(i + j)g \neq jg$. Además

observamos que $(j+n)g = jg$ para cualquier $j > 0$, Así cualquier elemento $i > n$, ig tiene un elemento idéntico en la secuencia $(i-n)g$. La secuencia $\{0g, 1g, \ldots, (n-1)g\}$ representa todos los elementos de G.

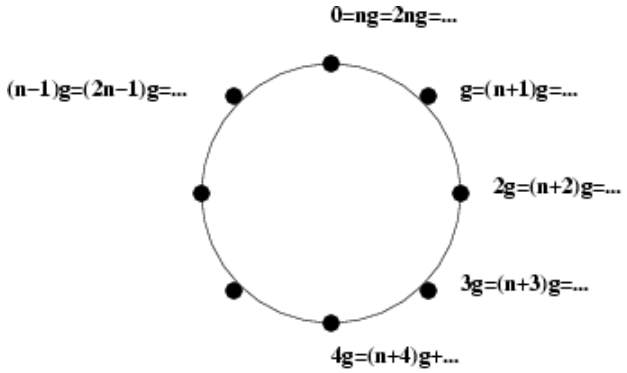


Figura 2.1: Estructura cíclica en un grupo cíclico

Sea p un número primo. El conjunto de los enteros módulo p, $\{0, 1, 2, 3, \ldots, p-1\}$, junto con las operaciones de adición y la multiplicación módulo p, conforman un campo finito de orden p. El campo se denota $\mathbb{F}_{\text{|}}$ y p es el módulo de $\mathbb{F}_p$. Cualquier entero a puede ser mapeado dentro del conjunto $\mathbb{F}_p$. a mod p se define como el residuo $r \in 0 \leq r \leq p-1$ obtenido al dividir a entre p. A esta operación se le llama *reducción módulo p*.

2.1.3. Campos Finitos Binarios

Los campos finitos de orden 2^m son llamados *campos binarios*. La aritmética definida sobre campos binarios tiene aplicaciones muy importantes en criptografía [9] [10]. Los elementos de $\mathbb{F}_{2^m}$ son representados por polinomios cuyos coeficientes están en el campo $\mathbb{F}_2$ y de grado máximo $m-1$:

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_2z^2 + a_1z + a_0 : a_i \in \{0,1\}\}. \quad (2.6)$$

Existe además el *polinomio zero* $f(x) = 0$ el cuál es definido con grado igual a $-\infty$.

Un polinomio $g(x)$ se dice *divisor* de un polimonio $f(x)$ si $f(x)$ es un polinomio múltiplo de $g(x)$: $f(x) = q(x)g(x)$ para algún polinomio $q(x)$. Un polinomio mónico es un polinomio diferente de cero con el coeficiente de mayor orden igual a 1: $f(x) = 1 + f_1x + f_2x^2 + \ldots + x^m$.

Todo polinomio $f(x)$ diferente de cero es divisible entre 1 y $f(x)$; a estos divisores se les llama *triviales*. Un polinomio $g(x)$ se llama *factor* de $f(x)$ si $g(x)$ es mónico y no es un divisor trivial de $f(x)$.

Un polinomio $g(x)$ de grado 1 o mayor y que no tiene factores es llamado *polinomio irreducible*, y un polinomio mónico irreducible es llamado *polinomio primo*.

Cualquier polinomio $f(x)$ puede ser expresado a partir de un polinomio dado, $g(x)$, de grado m como $f(x) = g(x)q(x) + r(x)$ siendo $q(x)$ un cociente y $r(x)$ un polinomio residuo. Definamos $\mathbb{R}_{\mathbb{F}_{2^m}}$ como el conjunto de todos los posible residuos producidos por $g(m)$, el cual esá compuesto de 2^m elementos.

Se define la aritmética modulo $g(x)$ sobre los elementos del conjunto de todos los posibles residuos de $g(x)$. Sea $r(x) = f(x) \bmod g(x)$ y $s(x) = h(x) \bmod g(x)$, es evidente que los residuos pueden expresarse en término de sus respectivos cocientes $q(x)$ y $t(x)$ respecto a el polinomio $g(x)$, $r(x) = f(x) - q(x)g(x)$ y $s(x) = h(x) - t(x)g(x)$. Por lo tanto:

$$f(x) + h(x) = r(x) + s(x) - (q(x) + t(x))g(x) \tag{2.7}$$

$$f(x)h(x) = r(x)s(x) - (q(x)s(x) + t(x)r(x))g(x) + q(x)t(x)g^2(x) \tag{2.8}$$

Es evidente que $(f(x) + h(x)) \bmod g(x) = (r(x) + s(x)) \bmod g(x)$ y $f(x)h(x) \bmod g(x) = r(x)s(x) \bmod g(x)$. Esto es, el residuo módulo $g(x)$ de la suma (o producto) de dos polinomios es igual a el residuo módulo $g(x)$ de la suma (o producto) de los residuos módulo $g(x)$ de dichos polinomios.

Es posible definir un campo de 2^m elementos para cualquier entero positivo $m > 1$. Los conjunto de elementos es el conjunto de polinomios residuo $\mathbb{R}_{\mathbb{F}_{2,m}}$ obtenidos a partir de un polinomio irreducible $g(x)$ de grado m. $\mathbb{R}_{\mathbb{F}_{2,m}} = \{r_0 + r_1x + \ldots + r_{m-1}^{m-1} | r_j \in \mathbb{F}_2, 0 \leq j \geq m-1\}$ el cual tiene 2^m elementos.

La suma y adición se definen módulo $g(x)$. La leyes asociativa, conmutativa y distributiva son heredadas de la aritmética común de polinomios. El conjunto $\mathbb{R}_{\mathbb{F}_{2,m}}$ junto con la adición módulo $g(x)$ conforman un grupo abeliano. Los elementos diferentes de cero de $\mathbb{R}_{\mathbb{F}_{2,m}}$ conforman un grupo abeliano bajo la multiplicación $g(x)$.

El conjunto de los residuos generados por $g(x)$, $\mathbb{R}_{\mathbb{F}_{2,m}}$, junto con las operaciones de adición y multiplicación módulo $g(x)$ conforman un campo finito el cual es denotado $GF(2^m)$ por las iniciales en inglés de *Galois Field*.

Para cualquier entero positivo m existe un polinomio irreducible $g(x) \in \mathbb{F}_2[x]$ de grado m.

Es posible considerar una secuencia de bits como una representación de un polinomio con coeficientes en $G(2)$ [11]. Por ejemplo,

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \leftarrow b(x)$$

$$b(x) = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

2.2. Curvas Elípticas

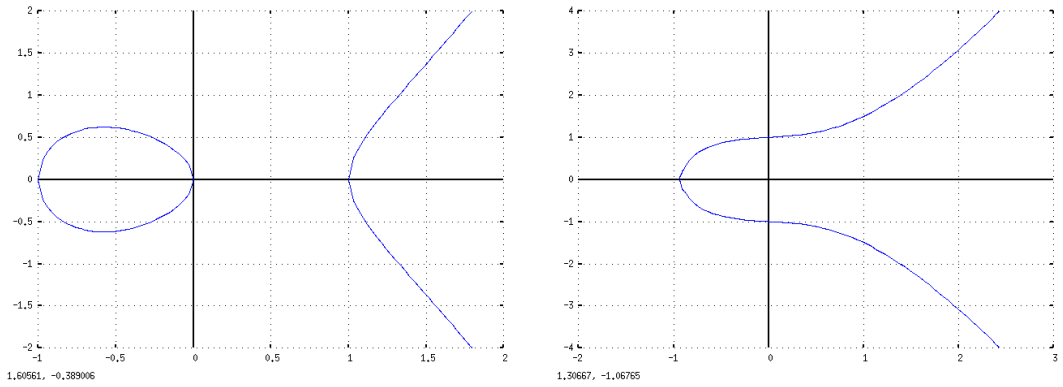
Los algoritmos criptográficos de llave pública están basados en aritmética definida sobre grupos abelianos multiplicativos y aditivos, como los que se encuentran en los campos finitos primos y binarios. En criptografía de curvas elípticas dichos grupos abelianos son definidos sobre operadores aplicados a un conjunto finito de puntos sobre una curva perteneciente a una familia en particular, las cuales son llamadas *curvas elípticas*.

Dicha familia de curvas y el grupo abeliano definido sobre el conjunto de sus puntos son definidas en la sección presente. También se describe posibles representaciones de dichos puntos, con la finalidad de obtener algoritmos computacionalmente eficientes. Finalmente se presenta la familia de curvas elípticas conocidas como *curvas binarias anómalas* o *curvas de Koblitz* y un conjunto de características explotadas para la obtención de algoritmos computacionalmente eficientes.

2.2.1. Curvas Elípticas sobre campos finitos

Una *curva elíptica* E sobre un campo K se define con la *ecuación de Weierstrass* [12]:

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \tag{2.9}$$



$E_1 : y^2 = x^3 - x$ $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

Figura 2.2: Curvas elípticas definidas sobre $\mathbb{R}$

donde $a_1, a_2, a_3, a_4, a_6 \in K$ y $\Delta \neq 0$ donde Δ es el *discriminante* de E y se define como[1]:

$$\begin{cases} \Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6 \\ d_2 = a_1^2 + 4a_2 \\ d_4 = 2a_4 + a_1 a_3 \\ d_6 = a_3^2 + 4a_6 \\ d_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2 \end{cases} \tag{2.10}$$

Δ se exige distinta de cero para garantizar la "suavidad"de la curva, es decir, garantizar la existencia de una sola tangente para cada punto sobre la curva. Sea L una extensión del campo K, la curva E/K también se considera definida sobre la extensión L. El conjunto de puntos *L-racionales* de E son aquellos que cumplen:

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x - a_6 = 0\} \cup \{\infty\} \tag{2.11}$$

donde ∞ es llamado el *punto al infinito* y se considera que satisface la ecuación de Weierstrass. ∞ es considerado un punto L-racional para cualquier extensión L de K.

La Figura 2.2 muestra dos curvas elípticas definidas sobre $\mathbb{R}$.

Las curvas que presentan mayor interés para el trabajo presentado son aquellas definidas sobre campos binarios es decir,sobre un campo base K de característica 2.

Las curvas elípticas pueden ser clasificadas de acuerdo a la característica de su campo base K y su discriminante Δ. Para curvas definidas sobre campos binarios se clasifican como:

1. *No-Supersingulares*:
   - La curva se define como:
   $$y^2 + xy = x^3 + ax^2 + b$$
   - El discriminante $\Delta = b$.
2. *Supersingulares*:
   - La curva se define como:
   $$y^2 + cy = x^3 + ax + b$$
   - El discriminante $\Delta = c^4$.

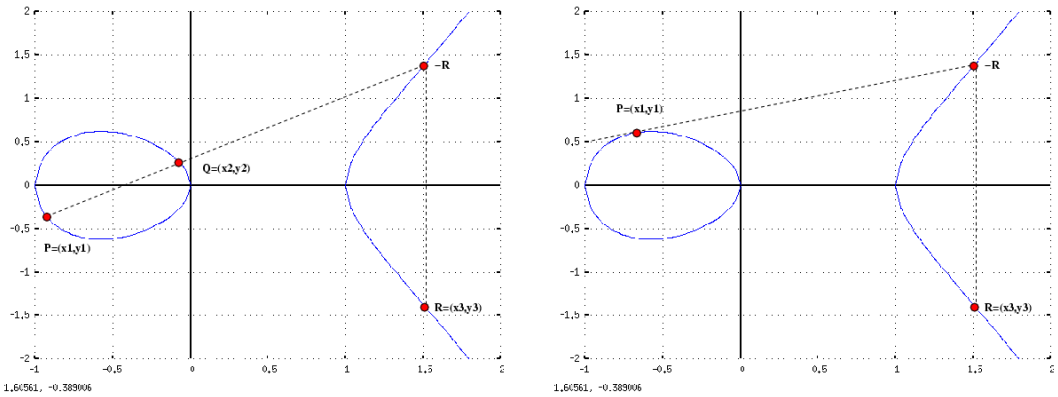
2.3. Ley de Grupo

Sea $E(K)$ el conjunto de puntos pertenecientes a una curva no-supersingular. El número de puntos en el conjunto es llamado el *orden del grupo* y se denota por $\#E(K)$.

Es posible construir un grupo abeliano a partir del conjunto de puntos y la definición de una operación de adición entre puntos que cumpla las leyes de conmutación, asociatividad, inverso aditivo, elemento identidad y cerradura.

La adición se puede definir geométricamente de una forma muy sencilla sobre los números reales. Sean dos puntos sobre E distintos entre sí $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, El punto R correspondiente a la suma entre P y Q se obtiene como sigue. Se dibuja la secante s a la curva E que interseca a los puntos P y Q. La recta s intersecará a la curva en un tercer punto, el cuál es nombrado $-R$. Se obtiene el reflejo de $-R$ sobre el eje x y este cuarto punto se define como R, es decir, la suma entre P y Q. Dicho proceso se muestra en la Figura 2.3(a).

El *doblado* de un punto se puede ver como la adición de un punto P consigo mismo. Su definición geométrica puede ser obtenida a partir de la adición. Al ser el mismo punto, la secante s se convierte en la tangente t a la



(a) Adición: $P + Q = R$ (b) Doblado: $P + P = R$

Figura 2.3: Adición y doblado de puntos geométrico.

curva E sobre el punto P. La tangente t interseca a la curva en un segundo punto el cuál se define como $-R$. R será el reflejo de $-R$ sobre el eje x. El proceso se muestra en la Figura 2.3(b). eje x.

Un grupo abeliano aditivo sobre el conjunto de puntos de una curva elíptica no-supersingular, con ecuación básica:

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

y campo finito binario base $E(\mathbb{GF}_{2^m})$ se define como sigue [1]:

1. *Identidad*: $P + \infty = \infty + P = P$ para toda $P \in E(\mathbb{GF}_{2^m})$.

2. *Inverso Aditivo*: Sea $P = (x, y) \in E(\mathbb{GF}_{2^m})$, entonces $(x, y) + (x, x + y) = \infty$. El punto $(x, x+y) \in E(\mathbb{GF}_{2^m})$ se denota por $-P$ y es conocido como el *negativo* de P.

3. *Adición entre puntos*: Sea $P = (x_1, y_1) \in E(\mathbb{GF}_{2^m})$ y $Q = (x_2, y_2) \in E(\mathbb{GF}_{2^m})$, donde $P \neq \pm Q$. Entonces $P + Q = (x_3, y_3)$ se obtiene:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{2.12}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{2.13}$$

donde $\lambda = (\frac{y_1+y_2}{x_1+x_2})$.

4. *Doblado de un punto*: Sea $P = (x_1, y_1) \in E(\mathbb{GF}_{2^m})$, donde $P \neq -P$. Definimos $P + P = 2P = (x_3, y_3)$ como:

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \tag{2.14}$$

$$y_3 = x_1^2 + \lambda x_3 + x_3 \tag{2.15}$$

donde $\lambda = x_1 + \frac{y_1}{x_1}$

2.4. Curvas de Koblitz

Sea $P(x)$ un polinomio irreducible de grado m sobre $GF(2)$. Por lo tanto, $P(x)$ genera un campo finito $GF(2^m)$ de característica dos.

Las curvas elípticas de Koblitz, también conocidas como Curvas Binarias Anómalas (curvas ABC por sus siglas en inglés *Anomalous Binary Curves*) son definidas sobre $GF(2)$ por las ecuaciones [4]:

$$E_0 : y^2 + xy = x^3 + 1 \tag{2.16}$$

$$E_1 : y^2 + xy = x^3 + x^2 + 1 \tag{2.17}$$

junto con el punto al infinito ∞, también denotado por el símbolo 0.

Las curvas son nombradas E_a de manera genérica donde $a \in \{0, 1\}$. Los grupo de puntos elípticos son generados sobre una extensión $GF(2^m)$ de $G(2)$ y el conjunto es nombrado $E_a(GF_{2^m})$.

Sea l un divisor de m. El grupo formado sobre el conjunto $E_a(GF_{2^l})$ es un subgrupo de $E_a(GF_{2^m})$. Por lo tanto el orden $\#E_a(GF_{2^l})$ divide al orden $\#E_a(GF_{2^m})$. Es de resaltar que, $\#E_0(2) = 4$ y $\#E_1(2) = 2$, por lo tanto el orden de cualquier grupo de curvas E_a generado sobre $GF(2^m)$ es múltiplo de 4 en las curvas E_0 y múltiplo de 2 en las curvas E_1.

Un número $m = nh$ se llama *casi-primo* si este puede ser factorizado como un número primo n y un número relativamente pequeño h normalmente perteneciente al intervalo $h \in [2, 4]$. Sea el grupo de puntos elípticos $E_a(GF_{2^m})$ para m primo. Si el orden $\#E_a(GF_{2^m})$ es casi-primo, entonces este puede ser factorizado como $\#E_a(GF_{2^m}) = hn$ donde n es primo y h es llamado el *cofactor* y es:

$$h = \begin{cases} 4 & \text{si } a = 0 \\ 2 & \text{si } a = 1 \end{cases}$$

2.5. Representación de Puntos

Para definir un grupo abeliano sobre $E(K)$, fué preciso definir la suma y doblado entre puntos del conjunto. Dichas operaciones fueron definidas en las ecuaciones 2.12, 2.13, 2.14 y 2.15. Sin embargo, para realizar dichas operaciones es necesario calcular una inversión de campo y varias multiplicaciones. La inversión de campo es una operación computacionalmente costosa.

Las coordenadas proyectivas permiten realizar la adición de puntos a través de un representación de cada punto como una clase de relación de equivalencia. Dicha representación es conocida como *representación en coordenadas proyectivas* mientras que la representación tradicional es conocida como *representación en coordenadas afines*. La idea fundamental es dividir el espacio K^3 en clases definidas por equivalencias. Cada punto afín puede ser relacionado uno-a-uno con una única clase de equivalencia. Ahora cada punto elíptico está representado por un conjunto de tripletas que cumplen con su correspondiente clase de equivalencia.

La suma y el doblado son redefinidos sobre la nueva representación. Ahora dichas operaciones pueden ser realizables sin necesidad de inversiones de campo. Dichas inversiones solo aparecen al convertir una representacion proyectiva en una afín. Dicha representación puede ser valiosa si se planean realizar una cantidad grande de sumas y doblado de puntos de manera sucesiva.

2.5.1. Coordenadas Proyectivas

Sean c y d enteros positivos sobre un campo K. Es posible definir una clase de equivalencia en el conjunto $K^3/\{(0,0,0)\}$ de la manera siguiente.

$$(X_1, Y_1, Z_1) \sim (X2, Y2, Z2)| \text{ si } X_1 = \lambda^c X_2, Y_1 = \lambda^d Y_2, Z_1 = \lambda Z_2.$$

La clase de equivalencia:

$$(X : Y : Z) = \left\{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K^*\right\}.$$

es llamada un *punto proyectivo* [1], y (X, Y, Z) un *punto representativo* de dicha clase, es decir, cualquier punto dentro de la clase es un punto representativo.

Específicamente, si $Z \neq 0$, $(\frac{X}{Z^c}, \frac{Y}{Z^d}, 1)$ es un punto representativo de la clase de equivalencia $(X : Y : Z)$.

Por lo tanto, si definimos el conjunto de todos los puntos proyectivos (clases de equivalencia) para cada λ posible en el campo K^* como $P(K)^* = \{(X : Y : Z) : X, Y, Z \in K, Z \neq 0\}$, obtenemos una correspondencia uno-a-uno entre el conjunto $P(K)^*$ y el conjunto de *puntos afines*:

$$A(K) = \{(x, y : x, y \in K)\}.$$

Cada punto en *sistema de coordenadas afines*, puede ser correspondido con el conjunto definido por una clase de equivalencia en particular. El conjunto de puntos pertenecientes a $P(K)^0 = \{(X : Y : Z) : X, Y, Z \in K, Z = 0\}$ es llamado la *línea al infinito*, porque esta clase no corresponde con ningún elemento en el conjunto de puntos afines.

La ecuación de Weierstrass para una curva elíptica $E(K)$ puede ser definida en coordenadas proyectivas al reemplazar x por $\frac{X}{Z^c}$ y y por $\frac{Y}{Z^d}$.

Los valores de las constantes c y d determinarán las características de la aritmética de curvas elípticas y por lo tanto la definición del algoritmo de adición de puntos en dicha representación.

2.5.2. Coordenadas de López-Dahab

Los sistemas de coordenadas proyectivas más ampliamente usadas son las *estándar*, donde $c = 1$ y $d = 1$, las *jacobianas*, con $c = 2$ y $d = 3$ y las *coordenadas proyectivas de López-Dahab*. Estas últimas presentan algoritmos para calcular la suma entre un punto en coordenadas afines y otro en coordenada proyectivas con tan solo 8 multiplicaciones. El definir esta suma entre *coordenadas mixtas* en pocas operaciones de campo hacen este sistema muy atractivo para su aplicación en el presente trabajo.

López y Dahab proponen un representación en puntos proyectivos para puntos sobre curvas elípticas donde $c = 1$ y $d = 2$ [13]. El punto proyectivo $(X : Y : Z), Z \neq 0$ corresponde al punto afín $X/Z, Y/Z^2$ y el punto ∞ corresponde a $(1 : 0 : 0)$, y finalmente, el punto negativo de $(X : Y : Z)$ es $(X : X + Y : Z)$.

Las fórmulas para calcular la suma (X_3, Y_3, Z_3) de $(X_1 : Y_1 : Z_1)$ y $(X_2 : Y_2 : 1)$ son [1]:

$$\begin{aligned} A &\leftarrow Y_2 \cdot Z_1^2 \\ B &\leftarrow X_2 \cdot Z_1 + X_1 \\ C &\leftarrow Z_1 \cdot B \\ C &\leftarrow B^2 \cdot (C + aZ_1^2) \\ Z_3 &\leftarrow C^2 \\ E &\leftarrow A \cdot C \\ X_3 &\leftarrow A^2 + D + E \\ F &\leftarrow X_3 + X_2 \cdot Z_3 \\ G &\leftarrow (X_2 + Y_2) \cdot Z_3^2 \\ Y_3 &\leftarrow (E + Z_3) \cdot F + G \end{aligned} \tag{2.18}$$

La ausencia de inversiones de campo propician la aplicación de este sistema en algoritmos donde se necesitan calcular múltiples adiciones de manera continua.

2.6. Multiplicación Escalar Elíptica

La operación central de esquemas criptográficos basados en criptografía de curvas elípticas es la multiplicación escalar elíptica, operación análoga a la exponenciación en grupos multiplicativos.

Dado un entero k y un punto $P \in E(GF_{2^m})$, la *multiplicación escalar elíptica* kP es el resultado de sumar P consigo mismo k veces.

Cada sistema criptográfico está basado en un problema matemático difícil que es improbable de resolver por medios computacionales en un tiempo razonable. El problema del logaritmo discreto es la base para la seguridad de muchos criptosistemas, incluyendo criptosistemas de curvas elípticas. Más específicamente la seguridad de curvas elípticas se basa en el problema del logaritmo discreto en curvas elípticas (ECDLP por sus siglas en inglés *Elliptic Curve Discrete Logarithm Problem*).

Las operaciones de suma y doblado de puntos pueden ser usadas para obtener la suma de cualquier número de copias de un punto ($2P$, $3P$, kP, etc.). El cálculo del punto kP es llamado la *multiplicación escalar* de un punto.

El ECDLP está basado en la dificultad de obtener el escalar k a partir de los puntos P y $Q = kP$.

2.7. Representación del Factor Escalar

La gran mayoría de algoritmos propuestos para calcular el producto kP de una forma eficiente está basado en la representación polinomial de Horner:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0 = a_0 + (a_1 + (a_2 + (\ldots + (a_{n-1} + (a_n + x)x)\ldots)x)x)x.$$

donde el escalar k es representado en forma polinómica y finalmente calculado el producto como una secuencia de Horner. La forma clásica de representación polinomial de un coeficiente k es su representación binaria $k = b_n 2^n + b_{n-1} + 2^{n-1} + \ldots + b_1 2 + b_0$ donde $b_i \in [0, 1]$.

2.7.1. Representación Binaria

El método más simple y antiguo para calcular nP está basado en la representación binaria de k. Si $k = \sum_{j=0}^{l-1} b_j 2^j$, donde cada $b_j \in \{0, 1\}$, entonces kP puede ser calculado como[6]:

$$kP = \sum_{j=0}^{l-1} b_j 2^j P = 2(..,2(2b_{l-1}P + b_{l-2}P) + ...) + b_0 P.$$

Este método requiere l doblados y $w_k - 1$ sumas, donde w_k es el peso Hamming (cantidad de coeficientes $b_j = 1$) de la representación binaria de k.

2.7.2. Representación NAF

La técnica básica para multiplicación escalar es el *método de adición y substracción*. Se basa en la *forma no adyacente* (NAF) del coeficiente k : una expansión binaria con signo, con la propiedad de que ningún coeficiente consecutivo es no cero.

$$\text{NAF}(29) = \langle\ 1,\ 0,\ 0,\ \text{-1},\ 0,\ 1\ \rangle$$

dado que $29 = 32 - 4 + 1$.

De la misma forma que todo número tiene una expansión binaria única, existe una única expansión NAF para cada entero k. Es posible obtener esta representación de varias manera. El método más utilizado es semejante a la forma en que obtenemos una representación binaria. Una representación binaria se obtiene dividiendo sucesivamente entre 2 y obteniendo los residuos 0 ó 1. Para una representación NAF se permiten residuos 0, 1 ó -1 y se escoge el que entregue un cociente par.

2.7.3. Representación WNAF

El método de adición y susbstracción puede ser generalizado para diseñar algoritmos todavía más eficientes a cambio de uso de memoria para almacenar resultados de precómputo. El método básico de ventana es llamado *método de ventana w*. Sea w un entero mayor a 1. Entonces cada entero positivo tiene un *NAF* de anchura w expresado por:

$$k = \sum_{j=0}^{l-1} u_j 2^j$$

donde:

- cada u_j distinto de cero es impar y menor que 2^{w-1} en valor absoluto;
- entre cada dos coeficientes u_j consecutivos, al menos uno no es cero.

El caso $w = 2$ es el NAF ordinario. El *NAF ancho-w* se escribe:

$$NAF_w(w) = \langle u_{l-1}, ...u_0 \rangle.$$

El Algoritmo 11 genera una expansión WNAF de un escalar positivo n. Cada vez que c es impar (inicialmente n), dentro del algoritmo, los w bits mas significativos son examinados para determinar la clase de congruencia (mod 2^w) en la cual se encuentra c. La clase de congruencia u es substraído de c, y ahora el nuevo coeficiente $c - u$ es divisible por 2^w garantizando una secuencia de $w - 1$ ceros en las siguientes iteraciones.

El peso de Hamming promedio de un *NAF ancho-w* es $(w + 1)^{-1}$. Esto contribuye directamente a una gran disminución en el número de sumas elípticas en una multiplicación escalar, a costo de uso de memoria. Sin embargo

Procedimiento 11 Algoritmo de generación de expansión W-NAF

Entrada: Un entero positivo n.
Salida: $NAF_W(n)$

```
c ← n
S ← ⟨⟩
while c > 0 do
  if c es impar then
    u ← c mods 2^w
    c ← c − u
  else
    u ← 0
  STATE Agrega u a S
  c ← c/2
Regresa S
```

el número de doblados se mantiene intacto en comparación con el Algoritmo NAF convencional.

2.7.4. Operador de Frobenius

El mapeo entre el elemento $x \in \mathbb{Z}$ y su cuadrado $x^2 \in \mathbb{Z}$ es llamado el mapeo Frobenius y puede ser aplicado de manera directa a un punto elíptico. El mapeo de Frobenius es denotado por τ :

$$\tau(x, y) := (x^2, y^2)$$

Las curvas de Koblitz son definidas sobre el campo finito $GF(2)$ y cumplen la propiedad siguiente:

Si $P = (x, y)$ es un punto en E_a entonces también lo es (x^2, y^2). Incluso es posible verificar que [3]:

$$(x^4, y^4) + 2(x, y) = \mu(x^2, y^2) \tag{2.19}$$

para todo punto (x, y) sobre E_a, donde $\mu = (-1)^{1-a}$.

Usando la notación del mapeo de Frobenius, podemos escribir la relación anterior como un equivalencia de transformaciones:

$$\tau(\tau P) + 2P = \mu\tau P.$$

Podemos escribir ahora de manera simbólica la siguiente equivalencia de funciones compuestas:

$$\tau^2 + 2 = \mu\tau \tag{2.20}$$

Al resolver la ecuación para τ podemos definir la equivalencia entre un mapeo de Frobenius con el número complejo $\tau = \frac{\mu+\sqrt{-7}}{2}$. Es decir una multiplicación escalar por $k = \frac{\mu+\sqrt{-7}}{2}$ es equivalente a un mapeo de Frobenius.

2.7.5. Representación τNAF

Cualquier entero positivo k puede ser escrito como una expansión polinómica τ no-adyacente:

$$k = \sum_{i=0}^{l-1} u_i \tau^i$$

donde cada $u_i \in \{0, \pm 1\}$ y l es la longitud de la expansión. Esta notación describe una función equivalente a la multiplicación escalar por k en términos de un expansión polinómica de τ, la cuál es un mapeo de Frobenius. Dicha expresión polinómica puede ser calculada eficientemente a través de su representación de Horner.

Un método basado en una representación NAF tradicional calcula kP sobre $E_a(G_{2^m})$ utilizando cerca de m doblados de punto y $\frac{m}{3}$ adiciones. El método análogo utilizando una representación τNAF utiliza alrededor de l mapeos de Frobenius y $\frac{l}{3}$ adiciones.

Sin embargo, la longitud de una representación τNAF de k es aproximadamente $\log_2(N(k)) = 2\log_2 k$, el cuál es el doble de la longitud de una representación NAF ordinaria.

Se ha probado que, si $\rho \equiv k \bmod \delta$ donde $\delta = \frac{\tau^m-1}{\tau-1}$, entonces $kP = \rho P$ para todos los puntos de orden n en $E_a(F_{2^m})$ [3]. La estrategia a seguir es encontrar $\rho = k \bmod \delta$ con una norma muy pequeña.

2.7.6. Expansión WτNAF

Es posible procesar w bits al mismo tiempo para obtener mejoras en tiempos de cómputo. El propósito es generar una expansión análoga a los algoritmos clásicos WNAF, donde es posible aumentar la cantidad de ceros en

la expansión pero requiere del cómputo previo y almacenamiento de múltiplos del punto elíptico uP. La expansión WτNAF de k representa una equivalencia entre la multiplicación escalar kP y la expresión:

$$\alpha_{u_0}P + \tau\alpha_{u_1}P + \tau^2\alpha_{u_2}P + \ldots + \tau^{l-1}\alpha_{u_{l-1}}P.$$

Es posible mantener la esencia de una expansión WNAF al momento de calcular una expansión WτNAF, al mantener una analogía con el Algoritmo 11. Esta vez, el número a expandir es un complejo $r_0 + r_1\tau \in \mathbb{Z}[\tau]$. Cuando se encuentra un elemento $r_0 + r_1\tau$ impar, se debe determinar a qué congruencia de clase, mod τ^w, pertenece. Es posible substraer un representativo de dicha congruencia y obtener un nuevo $r_0 + r_1\tau$ divisible por τ^w.

En analogía con el caso ordinario WNAF, es posible determinar la clase de congruencia de $r_0 + r_1\tau$ examinando los w bits más significativos de la *combinación adecuada* de r_0 y r_1.

Para obtener dicha combinación adecuada es necesario definir:

$$t_k = 2U_{k-1}U_k^{-1} \bmod 2^k \tag{2.21}$$

donde U_k es conocida como *la serie de Lucas* [2]. Cada elemento dentro de la secuencia es impar, por lo tanto t_k es un entero bien definido módulo 2^k el cual es par pero no divisible por 4. Por lo tanto se cumple:

$$t_k^2 - \mu t_k + 2 \equiv 0 \bmod 2^k \tag{2.22}$$

Es necesario notar que t_k cumple, sobre los números enteros, con la característica expresada la igualdad 2.20 que cumple τ en el espacio complejo $\mathbb{Z}[\tau]$. Por lo tanto la correspondencia $\tau \mapsto t_k$ induce un homomorfismo de anillo entre $\mathbb{Z}[\tau]$ y $\mathbb{Z}/2^k\mathbb{Z}$ vía:

$$\phi_k : \mathbb{Z}[\tau] \mapsto \mathbb{Z}/2^k\mathbb{Z}$$

$$u_0 + u_1\tau \mapsto u_0 + u_1 t_k \tag{2.23}$$

Se ha demostrado [2] que dado $\alpha \in \mathbb{Z}[\tau]$, entonces $\phi_k(\alpha) = 0$ si y solo si α es divisible por τ^k. Por lo tanto las clases de congruencia impares mod τ^k corresponden bajo ϕ_k con los elementos impares de $\mathbb{Z}/2^k\mathbb{Z}$.

Dichas observaciones son usadas para generar un algoritmo capaz de generar una expansión WτNAF análogo a el método WNAF común.

Sin embargo, en la literatura aparecen dos métodos propuestos por Solinas. En el primer método propuesto en [2] en 1997, observa que los números:

$$\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)$$

son incongruentes módulo τ^w. Por lo tanto propone calcular y almacenar los múltiplos uP para toda u impar en $2 < u < 2^{w-1}$.

En el segundo método publicado en [3],observa que los números:

$$\pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}$$

tal que $\alpha_u = u \bmod \tau^w$, también son incongruentes módulo τ^w. Proponiendo esta vez, calcular y almacenar los puntos $\alpha_u P$. Ambos métodos producen resultados satisfactorios, en la tesis presentamos la implementación de estos métodos.

2.7.7. Características Principales en las Representaciones

Cada representación y su aplicación en el cálculo de la multiplicación escalar expresan algoritmos computacionales compuestos por sucesiones de sumas entre puntos elípticos y doblados de punto. La eficiencia de los algoritmos basados en alguna representación polinomial del escalar está ligada directamente con dos elementos: la longitud de la representación y la cantidad de ceros en ella.

Representación	**Longitud**	**# Sumas de punto**	**# Doblados de punto**	**# Cuadrados de campo**	**Precálculo**
Binaria	m	$\frac{m}{2}$	$\frac{m}{2}$	—	—
NAF	m	$\frac{m}{3}$	$\frac{2m}{3}$	—	—
WNAF	m	$\frac{m}{w+1}$	$\frac{wm}{w+1}$	—	Tabla de $2^{w-1}-1$ múltiplos de m bits.
τNAF	$m+a+3$	$\frac{m+a+3}{3}$	—	$\frac{6(m+a+3)}{3}$	—
WτNAF	$m+a+3$	$\frac{m+a+3}{w+1}$	—	$\frac{3w(m+a+3)}{w+1}$	Tabla de $2^{w-1}-1$ múltiplos de m bits.

Cuadro 2.1: Tabla comparativa para diferentes representaciones polinomiales de un escalar para una curva elíptica $E_a(GF_{2^m})$ en coordenadas proyectivas.

Para los algoritmos τNAF aparece una nueva característica de la representación que repercute de manera directa en la eficiencia de estos métodos. En algoritmos tradicionales NAF la cantidad de ceros en una representación implica la misma cantidad de operaciones de doblado. En algoritmos τNAF

los ceros implican operaciones de cuadrado de campo. Esto implica que la eficiencia alcanzable por implementaciones de algoritmos τNAF en comparación con algoritmos NAF clásicos depende de la relación entre las eficiencias de los algoritmos de doblado de puntos y el de cuadrado de campo finito.

La Tabla 2.1 muestra una tabla comparativa de los cálculos que implican cada algoritmo. La eficiencia que puede alcanzar cada representación depende de la plataforma de implementación, de la eficiencia de las operaciones básicas (suma de puntos, doblado de punto y cuadrado de campo) y finalmente de la forma en que se exploten las características específicas de cada representación.

Capítulo 3

Algoritmos para Hardware Reconfigurable

Cualquier algoritmo para realizar una multiplicación escalar kP basado en una representación polinomial de escalar k puede ser descrito (independientemente de si son calculadas de manera secuencial o concurrente) en dos etapas: el cálculo de la expansión polinomial del escalar k y el cálculo de la representación de Horner de la multiplicación kP correspondiente.

Considerar la multiplicación escalar τNAF como el conjunto de dos procesos diferentes nos da la capacidad de clasificar la aritmética implícita en los algoritmos de una manera natural. La generación de una expansión polinomial τNAF del escalar k requiere de cálculos sobre el campo $\mathbb{Z}$ para números particularmente grandes. En contraste, el cómputo de la expresión de Horner correspondiente está basado en aritmética de campos finitos binarios. Es interesante el contraste de la eficiencia esperada en cada aritmética. La aritmética $G(2^m)$ es más eficiente que la aritmética $\mathbb{Z}$ sobre una plataforma de hardware. Una implementación clásica en software presenta dicha comparación de eficiencia en orden inverso.

Solinas ha presentado algoritmos computacionales para ambos procesos [3] fáciles de interpretar e implementar en software. Para nuestro propósito es necesario presentar adaptaciones de dichos procesos para una arquitectura en hardware. Una arquitectura en dicha plataforma nos provee de la capacidad de realizar ciertas secuencias aritméticas en un solo ciclo de reloj. Dichas secuencias, al ser calculadas en el mismo ciclo de reloj, pueden ser consideradas atómicas y en consecuencia como un conjunto de operaciones simples calculadas en paralelo.

Para el propósito de nuestra implementación, se escoge la curva propuesta por NIST $K-233$ [14] definida sobre el campo binario $GF(2^{233})$. El seleccionar dicha curva y aritmética de manera previa nos permitirá optimizar nuestra arquitectura.

En el presente capítulo se estudian los algoritmos presentados por Solinas en [3] y adaptaciones presentadas por Julio López [15]. Dichos algoritmos son adaptados a un esquema de hardware y optimizados para la curva elíptica $K-233$ y su aritmética implícita.

3.1. Expansión WτNAF

El propósito del proceso de expansión WτNAF es expresar el escalar k como una expansión polinomial de τ.

$$k = \sum_{i=0}^{l-1} u_i \tau^i.$$

Sin embargo la longitud l de la expansión es aproximadamente $2\log_2 k$. Dicha longitud es el doble de una expansión NAF binaria. Es posible reducir dicha longitud si se encuentra un número $\rho = k \bmod \delta$, donde $\delta = \frac{\tau^m - 1}{\tau - 1}$, con norma pequeña.

El proceso de expansión del escalar k se encuentra compuesto por los siguientes algoritmos.

3.1.1. Reducción Parcial

Sea $\mathbb{Z}[\tau]$ el anillo formado por los polinomios de τ con coeficientes enteros. Dado que $\tau^2 = \mu\tau - 2$, todos los polinomios $\alpha \in \mathbb{Z}[\tau]$ pueden ser expresados en una forma canónica $\alpha = a_0 + a_1\tau$.

Sea $P \in E(\mathbb{F}_{2^m})$. Recordemos que por el teorema pequeño de Fermat:

$$x^{2^m} \equiv x \bmod 2^m.$$

esta última equivalencia puede ser escrita en términos del mapeo de Frobenius.

$$\tau^m x \equiv x \bmod 2^m. \tag{3.1}$$

Por lo tanto, aplicando el teorema pequeño de Fermat en el mapeo de Frobenius del punto elíptico P obtenemos:

$$(\tau^m - 1)P = \tau^m P - P = P - P = \infty \tag{3.2}$$

Concluimos que $\gamma \equiv k \bmod (\tau^m - 1)$ y que $kP = \gamma P$ para $P \in E(\mathbb{F}_{2^m})$. De una manera análoga se demuestra que $\rho \equiv k \bmod \delta$, donde $\delta = (\tau^m - 1)/(\tau - 1)$ y $kP = \rho P$ para todos los puntos P de orden n en $E(\mathbb{F}_{2^m})$.

Es necesario encontrar $k = \delta\kappa + \rho$, donde el cociente κ y el residuo ρ están en $\mathbb{Z}[\tau]$, con la norma de ρ particularmente pequeña.

Los siguiente algoritmos fueron propuestos por Solinas en [3] para encontrar dicho residuo.

Procedimiento 12 División aproximada entre r

Entrada: *Parámetros de la curva*: s_i, r, V_m. Un entero positivo $n < \frac{r}{2}$
Salida: $\lambda_i = s_i \frac{n}{r}$ con C bits de aproximación.

1: $n' \leftarrow \lfloor \frac{n}{2^{m-K-2+a}} \rfloor$
2: $g' \leftarrow s_i n'$
3: $h' \leftarrow \lfloor \frac{g'}{2^m} \rfloor$
4: $j' \leftarrow V_m h'$
5: $l' \leftarrow Round\left(\frac{g'+j'}{2^{K-C}}\right)$
6: **Regresa** $\lambda' := \frac{l'}{2^C}$

Procedimiento 13 Reducción Parcial Módulo $\delta = \frac{(\tau^m-1)}{(\tau-1)}$

Entrada: *Parámetros de la curva*: m, a, s_0, s_1, r. Factor Escalar n.
Salida: Enteros r_0, r_1 especificando $r_0 + r_1\tau \equiv n \bmod \frac{(\tau^m-1)}{(\tau-1)}$

1: $d_0 \leftarrow s_0 + \mu s_1$
2: $\lambda_0 \leftarrow s_0 n/r$ aproximado C bits usando Alg. 12.
3: $\lambda_1 \leftarrow s_1 n/r$ aproximado C bits usando Alg. 12.
4: $(q_0, q_1) \leftarrow \text{RoundOff}(\lambda_0, \lambda_1)$
5: $r_0 \leftarrow n - d_0 q_0 - 2 s_1 q_1$
6: $r_1 \leftarrow s_1 q_0 - s_0 q_1$
7: **Regresa** r_0, r_1

El algoritmo 14 es una variante de la función *Redondeo* usada para calcular $\rho = c \bmod d$, donde c, d y ρ son números reales y es posible calcularlo de

acuerdo a la ecuación 3.3.

$$\rho = c - \text{ Redondeo}(c/d)d \tag{3.3}$$

Donde *Redondeo* se define como en la ecuación 3.4.

$$\text{Redondeo}(\lambda) = \lfloor \lambda + 1/2 \rfloor \tag{3.4}$$

El proceso de *Rounding Off* calcula el equivalente al proceso del redondeo sobre un número complejo $\lambda = \lambda_0 + \lambda_1\tau$.

Procedimiento 14 *Rounding Off*

Entrada: Enteros reales λ_0, λ_1, especificando $\lambda = \lambda_0 + \lambda_1\tau$.
Salida: Enteros reales q_0, q_1, especificando $q_0 + q_1\tau = Round(\lambda)$.

1: $f_0 \leftarrow$Round(λ_0)
2: $f_1 \leftarrow$Round(λ_1)
3: $\eta_0 \leftarrow \lambda_0 - f_0$
4: $\eta_1 \leftarrow \lambda_1 - f_1$
5: $h_0 \leftarrow 0$
6: $h_1 \leftarrow 0$
7: $\eta \leftarrow 2\eta_0 + \mu\eta_1$
8: **if** $\eta \geq 1$ **then**
9: **if** $\eta_0 - 3\mu\eta_1 < -1$ **then**
10: $h_1 \leftarrow \mu$
11: **else**
12: $h_0 \leftarrow 1$
13: **else**
14: **if** $\eta_0 + 4\mu\eta_1 \geq 2$ **then**
15: $h_1 \leftarrow \mu$
16: **if** $\eta < -1$ **then**
17: **if** $\eta_0 - 3\mu\eta_1 \geq 1$ **then**
18: $h_1 \leftarrow -\mu$
19: **else**
20: $h_0 \leftarrow -1$
21: **else**
22: **if** $\eta_0 + 4\mu\eta_1 < -2$ **then**
23: $h_1 \leftarrow -\mu$
24: $q_0 \leftarrow f_0 + h_0$
25: $q_1 \leftarrow f_1 + h_1$
26: Salida q_0, q_1

3.1.2. Método de Ventana

Existen dos algoritmos propuestos por Solinas para implementar algoritmos de ventana w en distintas publicaciones. El algoritmo WτNAF propuesto en [2] propone el precálculo y almacenamiento de los número impares $\pm 1, \pm 3, \ldots, \pm 2^{w-1} - 1$, como múltiplos de P. En dicho algoritmo, se toman dichos impares como los representantes de la clase de congruencia mod τ^w. Los impares u pueden ser vistos como complejos con componente τ igual a cero, $u + 0\tau$. Cada vez que se obtiene la clase de congruencia u a la que pertenece $r_0 + r_1\tau$, basta con substraer u de r_0, pues la componente en τ de u es cero como sucede en el Algoritmo 15.

Procedimiento 15 Coeficientes de la expansión WτNAF

Entrada: *Parámetros de la curva:t m, a, s_0, s_1, t_w. Escalar k*
Salida: $RTNAF_W(k)$

1: Calcular $(r_0, r_1) \leftarrow k \bmod \delta$ usando algoritmo 13
2: Iniciar $S \leftarrow \langle\rangle$
3: **while** $r_0 \neq 0$ o $r_1 \neq 0$ **do**
4: **if** r_0 es impar **then**
5: $u \leftarrow r_0 + r_1 t_w \bmod s 2^w$
6: $r_0 \leftarrow r_0 - u$
7: **else**
8: $u \leftarrow 0$
9: Agregar u en S
10: $(r_0, r_1) \leftarrow (r_1 + \frac{\mu r_0}{2}, \frac{-r_0}{2})$
11: **Regresa** S

Solinas publica en el 2000 un nuevo método de multiplicación WτNAF, esta vez se definen $\alpha_i = i \bmod \tau^w$ para $i \in 1, , 3, 5, \ldots, 2^{w-1} - 1$ como los elementos representativos de la clase de congruencia mod τ^w, pues se ha demostrado [3] que dichos complejos no son divisibles por τ^w. Ahora k puede ser expresada como $k = \sum_{i=0}^{l-1} u_i \tau^i$ donde $u_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$. El Algoritmo 16 es una variación del algoritmo 15, donde esta vez los elementos representativos, $\alpha_i = i \bmod \tau^w$, pueden contener componente compleja $\alpha_0 + \alpha_1\tau$.

Cuando en alguna división sucesiva ρ el resultado no es divisible entre τ el residuo α_u es distinto de cero. Sin embargo $\frac{(r_0, r_1) - \xi(\beta_u, \gamma_u)}{\tau}$ será divisible entre τ^{w-1} por lo tanto, los siguientes $w - 1$ digitos serán ceros.

Procedimiento 16 Coeficientes de la expansión WτNAF

Entrada: *Parámetros de la curva:t m, a, s_0, s_1, t_w. Elementos representativos:* $\alpha_u = \beta_u + \gamma_u\tau$ *para* $u = 1, 3, \ldots, 2^{w-1} - 1$. *Escalar* k

Salida: $RTNAF_W(k)$

1: Calcular $(r_0, r_1) \leftarrow k \bmod \delta$ usando algoritmo 13
2: Iniciar $S \leftarrow \langle\rangle$
3: **while** $r_0 \neq 0$ o $r_1 \neq 0$ **do**
4: **if** r_0 es impar **then**
5: $u \leftarrow r_0 + r_1 t_w \bmod s\ 2^w$
6: **if** $u > 0$ **then**
7: $\xi \leftarrow 1$
8: **else**
9: $\xi \leftarrow -1$
10: $u \leftarrow -u$
11: $r_0 \leftarrow r_0 - \xi\beta_u$
12: $r_1 \leftarrow r_1 - \xi\gamma_u$
13: **else**
14: $u \leftarrow 0$
15: Agregar ξu en S
16: $(r_0, r_1) \leftarrow (r_1 + \frac{\mu r_0}{2}, \frac{-r_0}{2})$
17: **Regresa** S

3.1.3. Algoritmos para Hardware Reconfigurable

Dado que se propone optimizar la arquitectura para un caso de estudio (la curva K-233), se puede obtener el tamaño de palabra óptimo para realizar la aritmética entera solicitada en los algoritmos. El tamaño en bits de las constantes utilizadas para el caso de la curva K-233 con polinomio irreducible $\alpha^{233} + \alpha^{74} + 1$ es:

1. k (232 bits máximo)
2. s_0 (116 bits)
3. s_1 (117 bits)
4. V_m (118 bits)

Tomando las constantes como referencia y los Algoritmos 12, 13 y 16 propuestos por Solinas se concluye que el tamaño máximo de los operandos en las adiciones es de 237+C bits, donde C es el factor de aproximación para la división aproximada. La cantidad de bits para llevar a cabo la aritmética se toma de 256 bits y con un factor de aproximación de 5 bits. Para las multiplicaciones la cantidad de bits óptimo para representar una cantidad es de 120+C bits, por lo tanto la palabra de 256 bits sigue siendo suficiente.

A partir de la definición del tamaño de bits para la aritmética es posible definir operadores básicos a utilizar en el algoritmo paralelo:

- Sumador-Restador de 256 bits.
- Multiplicador de 128 bits.
- Operador *mods* de 128 bits.
- Cuatro registros de propósito general de 256 bits.
- Corrimientos de bits, así como multiplicaciones y divisiones por 2 se realizan de un manera muy eficiente reordenando la representación binaria.

Debido al alto costo de un multiplicador de 128 bits, el algoritmo es paralelizado suponiendo que una sola unidad de multiplicación está disponible. Esto reduce las posibilidades para el proceso, pero las operaciones simples

Procedimiento 17 Reducción Parcial módulo δ : Versión Paralela

Entrada: Parámetros de la curva: m,a,s_0, s_1,r. Escalar n.

Salida: Registros R_0 y R_1 almacenando $r_0 + r_1\tau \equiv n \bmod \frac{\tau^m-1}{\tau-1}$

1: $A \leftarrow s_0 \lfloor \frac{n}{2^{m-K-2+a}} \rfloor$
2: $R_0 \leftarrow V_m \lfloor \frac{A}{2^m} \rfloor$
3: $R_0 \leftarrow \frac{Round(\frac{A+R_0}{2^{K-C}})}{2^C}$
4: $A \leftarrow s_1 \lfloor \frac{n}{2^{m-K-2+a}} \rfloor$
5: $R_1 \leftarrow V_m \lfloor \frac{A}{2^m} \rfloor$
6: $R_1 \leftarrow \frac{Round(\frac{A+R_1}{2^{K-C}})}{2^C}$
7: $A \leftarrow s_0 R_0$
8: $U \leftarrow 2s_1 R_1$
9: $R_1 \leftarrow s_1 R_0$
10: $R_1 \leftarrow R_1 - A$
11: $R_0 \leftarrow d_0 R_0$
12: $R_0 \leftarrow R_0 U$
13: $R_0 \leftarrow n - R_0$

basadas en reordenamiento de bits, al ser económicas en recursos y tiempo de cómputo en hardware, son importantes al paralelizar el algoritmo.

El Algoritmo 14 puede verse como un cómputo sobre la parte fraccionaria de dos divisiones, donde la parte fraccionaria es siempre respresentada por C bits (5 bits para nuestro caso de estudio). Dicho cómputo tiene la finalidad de decidir el *redondeo* sobre cada coordenada de un número complejo λ, lo cuál se reduce a sumar 0, 1 ó -1 a cada coordenada. Al ser aritmética de 5 bits, es posible implementar el proceso de decisión de manera completa en un solo ciclo de reloj. El algoritmo 18 presenta la secuencia seguida para calcular los valores h_0 y h_1 utilizados en las líneas 24 y 25 del Algoritmo 14 para realizar el *redondeo* de λ.

Las comparaciones en hardware son implementadas de forma especializada para números con parte entera representada con 2 bits y parte fraccionaria representada con 5 bits. La línea 9 del Algoritmo 18 garantiza que las operaciones realizadas de manera interna nunca rebasan dicho tamaño de representación.

La Tabla 3.1 muestra las formas de calcular dichas comparaciones a partir de operaciones simples a nivel de bits de un número $B = s(b_6b_5.b_4b_3b_2b_1b_0)_2$. donde s es un bit representando el signo (0 es positivo y 1 es negativo).

Procedimiento 18 Rounding Off: Versión Hardware

Entrada: λ_0, λ_1 representados por 5 bits y un 6 bit de signo
Salida: $h_0, h_1 \in \{-1, 0, 1\}$

if $parte\_fraccionaria(\lambda_0) < \frac{1}{2}$ **then**
  $\eta_0 \leftarrow parte\_fraccionaria(\lambda_0)$
else
  $\eta_0 \leftarrow \sim signo(\lambda_0) | complemento_{2,5bits}(parte\_fraccionaria(\lambda_0))$
if $parte\_fraccionaria(\lambda_1) < \frac{1}{2}$ **then**
  $\eta_1 \leftarrow parte\_fraccionaria(\lambda_1)$
else
  $\eta_1 \leftarrow \sim signo(\lambda_1) | complemento_{2,5bits}(parte\_fraccionaria(\lambda_1))$
$A = 2\eta_0 - \eta_1$, $B = \eta_0 + 3\eta_1$, $C = \eta_0 - 4\eta_1$
$C_1 = A \geq 1$, $C_2 = B < -1$, $C_3 = C \geq 2$,
$C_4 = A < -1$, $C_5 = B \geq 1$, $C_6 = C < -2$.
if (C_1 and C_2) or ($\sim C_1$ and C_3) **then**
  $h_1 \leftarrow -1$
else if (C_4 and C_5)(or)($\sim C_4$ and C_6) **then**
  $h_1 \leftarrow 1$
else
  $h_1 \leftarrow 0$
if C_1 and $\sim C_2$ **then**
  $h_0 \leftarrow 1$
else if C_4 and $\sim C_5$ **then**
  $h_0 \leftarrow -1$
else
  $h_0 \leftarrow 0$

Comparación	**Operación de Bits**
$B \geq 1$	$\sim s$ and (b_6 or b_5)
$B < -1$	s and (b_6 or (b_5 and (b_4 or ... or b_0)))
$B \geq 2$	$\sim s$ and b_6
$B < -2$	s and (b_6 and (b_5 or (b_4 or ... or b_0)))

Cuadro 3.1: Comparaciones a Nivel Hardware

Los Algoritmos 19 y 20 calculan la expansión WτNAF. Los algoritmos son análogos a los algoritmos NAF generales. Se basan en la división sucesiva del número $\rho \in \mathbb{Z}[\tau]$ entre τ permitiendo residuos $\pm 1, \pm 3, \ldots, \pm 2^{w-1} - 1$ en el Algoritmo 15 o residuos $\pm\alpha_i = \pm i \bmod \tau^w$ en el Algoritmo 16 para $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$.

El Algoritmo 15 ha sido optimizado para un esquema de *pipeline* en una arquitectura de hardware en el Algoritmo 19.

Procedimiento 19 Expansión de coeficientes WτNAF: Versión Paralela. Ver: [2]

Entrada: : Parámetros de la curva: m, a, s_0, s_1, t_w. Escalar reducido: $\rho = (R_0, R_1)$
Salida: $R\tau NAF_W(\rho)$

1: **if** $R_0 \neq 0$ o $R_1 \neq 0$ **then**
2: $A \leftarrow R_1 \cdot t_w$
3: $U \leftarrow (R_0 \cdot A)$ mods 2^w, Almacena U
4: $R_0 \leftarrow R_0 - U$
5: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
6: **while** $R_1 \neq -\mu\frac{R_0}{2}$ o $R_0 \neq 0$ **do**
7: **if** R_1 es impar **then**
8: $A \leftarrow R_1 \cdot t_w$, Almacena $U_{contador}$
9: $U \leftarrow (R_0 \cdot A)$ mods 2^w, Almacena U
10: $R_0 \leftarrow R_0 - U$, Reinicia $U_{contador}$
11: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
12: **else**
13: Incrementa $U_{contador}$
14: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$

El Algoritmo 16 es paralelizado de la misma forma en el Algoritmo 20, sin embargo necesita las constantes α_u almacenadas en una memoria ROM y requiere una adición más y un ciclo de direccionamiento de las constantes α_u.

Se sabe que la cantidad mínima de ceros consecutivos es de $w-1$ para una ventana de w bits. Cada vez que aparece una secuencia de ceros en la cadena, éstos son contados. La longitud de una secuencia de ceros es almacenada para su uso posterior. La primera ronda del ciclo *while* original fué extraida debido al conteo de ceros. El Algoritmo 20 genera una expansión WτNAF con el siguiente formato:

Sean $w_i \in [1, 3, 5, \ldots, 2^{w-1}-1]$ los elementos distintos de cero pertenecientes a la expansión WτNAF. Las secuencias de ceros consecutivos se representan

Procedimiento 20 Expansión de coeficientes WτNAF: Versión Paralela. Ver: [3]

Entrada: : Parámetros de la curva: m, a, s_0, s_1, t_w. Memoria ROM almacenando $\alpha_u = (\beta, \gamma)$ para $u = 1, 3, \ldots, 2^{w-1} - 1$. Escalar reducido: $\rho = (R_0, R_1)$

Salida: $R\tau NAF_W(\rho)$

1: **if** $R_0 \neq 0$ o $R_1 \neq 0$ **then**
2: $A \leftarrow R_1 \cdot t_w$
3: $U \leftarrow (R_0 \cdot A)$ mods 2^w, Almacena U
4: Direcciona $\alpha_{abs(U)} = (\beta, \gamma)$, $signoU = \{-1, 1\}$, de acuerdo al signo de U.
5: $R_0 \leftarrow R_0 - signoU\beta$
6: $R_1 \leftarrow R_1 - signoU\gamma$
7: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
8: **while** $R_1 \neq -\mu\frac{R_0}{2}$ o $R_0 \neq 0$ **do**
9: **if** R_1 es impar **then**
10: $A \leftarrow R_1 \cdot t_w$, Almacena $U_{contador}$
11: $U \leftarrow (R_0 \cdot A)$ mods 2^w, Almacena U
12: Direcciona $\alpha_{abs(U)} = (\beta, \gamma)$, $signoU = \{-1, 1\}$, de acuerdo al signo de U.
13: $R_0 \leftarrow R_0 - signoU\beta$
14: $R_1 \leftarrow R_1 - signoU\gamma$, Reinicia $U_{contador}$
15: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
16: **else**
17: Incrementa $U_{contador}$
18: $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$

por $z_i \in [w-1, 2w-2]$. La expansión WτNAF puede ser representada con el formato $w_0, z_0, w_1, z_2, \ldots, z_{i-1}, w_i$.

Las condiciones del ciclo *while* en la línea 8 del Algoritmo 20 fueron adaptadas a un esquema de pipeline. Debido a que los registros R_0 y R_1 no han sido actualizados en el ciclo siguiente a su último cálculo, normalmente es necesario introducir *burbujas de tiempo*, es decir ciclos de reloj que no realizan ninguna operación con el fin de esperar un resultado necesario para continuar los cálculos. Sin embargo es posible realizar comparaciones equivalentes, con valores previos de los registros, a las solicitadas por el Algoritmo 16, con el fin de evitar dichos ciclos ociosos:

1. R_0 será cero si y solo si $R_1 = -\mu\frac{R_0}{2}$.
2. R_1 será cero si y solo si $R_0 = 0$.
3. R_0 será impar si y solo si R_1 es impar, porque $-\mu\frac{R_0}{2}$ es par.

Dichas equivalencias son obtenidas de manera directa de la línea 16 del Algoritmo 16.

3.2. Aritmética de Curvas Elípticas

La operación básica en un grupo abeliano es la adición. En este caso se ha definido la adición entre puntos de una curva elíptica. Otra operación básica es el doblado de un punto. Esta operación es definida como un caso particular de la suma, la adición de un punto consigo mismo. Sin embargo se define de manera independiente para poder optimizar su implementación.

López y Dahab propusieron una representación proyectiva para puntos elípticos donde $c = 1$ y $d = 2$ así como un algoritmo para calcular la suma de un punto en coordenadas afines con otro punto en su representación proyectiva. La suma se obtiene en coordendas proyectivas [13].

El siguiente algoritmo se encuentra publicado en [1]. Está optimizado para calcular una adición entre un punto en el sistema de coordenas de López-Dahab (LD) y un punto en el sistema de coordenadas afines y ser implementado sobre una plataforma de software.

Los algoritmos propuestos por López y Dahab presentados en [1] han sido paralelizados utilizando funciones atómicas y han sido optimizados para el

Procedimiento 21 Suma de puntos en coordenas LD-afines

Entrada: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD coordinates, $Q = (x_2, y_2)$ en coordenadas afines sobre E/K.
Salida: $P + Q = (X_3 : Y_3 : Z_3)$ en coordenadas LD.

1: **if** $Q = \infty$ **then**
2: **Regresa** P
3: **if** $P = \infty$ **then**
4: **Regresa** $(x_2 : y_2 : 1)$
5: $T_1 \leftarrow Z_1 \cdot x_2$
6: $T_2 \leftarrow Z_1^2$
7: $X_3 \leftarrow X_1 + T_1$
8: $T_1 \leftarrow Z_1 \cdot X_3$
9: $T_3 \leftarrow T_2 \cdot y_2$
10: $Y_3 \leftarrow Y_1 + T_3$
11: **if** $X_3 = 0$ **then**
12: **if** $Y_3 = 0$ **then**
13: usa Algoritmo 22 para calcular
14: $(X_3 : Y_3 : Z_3) = 2(x_2 : y_2 : 1)$
15: **Regresa** $(X_3 : Y_3 : Z_3)$
16: **else**
17: **Regresa** ∞
18: $Z_3 \leftarrow T_1^2$
19: $T_3 \leftarrow T_1 \cdot Y_3$
20: **if** $a = 1$ **then**
21: $T_1 \leftarrow T_1 + T2$
22: $T_2 \leftarrow X_3^2$
23: $X_3 \leftarrow T_2 \cdot T_1$
24: $T_2 \leftarrow Y_3^2$
25: $X_3 \leftarrow X_3 + T_2$
26: $X_3 \leftarrow X_3 + T_3$
27: $T_2 \leftarrow x_2 \cdot Z_3$
28: $T_2 \leftarrow T_2 + X_3$
29: $T_1 \leftarrow Z_3^2$
30: $T_3 \leftarrow T_3 + Z_3$
31: $Y_3 \leftarrow T_3 \cdot T_2$
32: $T_2 \leftarrow x_2 + y_2$
33: $T_3 \leftarrow T_1 \cdot T_2$
34: $Y_3 \leftarrow Y_3 + T_3$
35: **Regresa** $(X_3 : Y_3 : Z_3)$

Procedimiento 22 Doblado de puntos en coordenadas LD

Entrada: $P = (X_1 : Y_1 : Z_1)$ en coordenadas LD sobre E/K.
Salida: $2P = (X_3 : Y_3 : Z_3)$ en coordenadas LD.

1: **if** $P = \infty$ **then**
2: **Regresa** ∞.
3: $T_1 \leftarrow Z_1^2$
4: $T_2 \leftarrow X_1^2$
5: $Z_3 \leftarrow T_1 \cdot T_2$
6: $X_3 \leftarrow T_2^2$
7: $T_1 \leftarrow T_1^2$
8: $T_2 \leftarrow T_1 \cdot b$
9: $X_3 \leftarrow X_3 + T2$
10: $T_1 \leftarrow Y_1^2$
11: **if** $a = 1$ **then**
12: $T_1 \leftarrow T_1 + Z_3$
13: $T_1 \leftarrow T_1 + T_2$
14: $Y_3 \leftarrow X_3 \cdot T1$
15: $T_1 \leftarrow T_2 \cdot Z_3$
16: $Y_3 \leftarrow Y_3 + T_1$
17: **Regresa** $(X_3 : Y_3 : Z_3)$

Procedimiento 23 Suma de puntos en coordenadas LD-afines: Versión Paralela

Entrada:	$Q = (Q_x : Q_y : Q_z)$ en coordendas LD , $P = (P_x, P_y)$ en coordenadas afines sobre K-233: $f(z) = z^{233} + z^{74} + 1$ / $GF(2^{233})$
Salida:	$P + Q = (Q_x : Q_y : Q_z)$ en coordendas LD.
ciclo 1:	**if** $P_z = 0$ **then** regresa $(x_2 : y_2 : 1)$. **else** $Q_x \leftarrow Q_x + Q_z \cdot P_x$.
ciclo 2:	$Q_y \leftarrow Q_y + Q_z^2 \cdot P_y$
ciclo 3:	$T_1 \leftarrow Q_z \cdot Q_x$, $Q_z \leftarrow T_1^2$
ciclo 4:	**if** $Q_x = 0$ **then** **if** $Q_y = 0$ **then** Doblado De Puntos($P_x, P_y, 1$). **else** regresa $[1, 0, 0]$ **else** $T_3 = T_1 \cdot Q_y$.
ciclo 5:	$Q_x = (Q_x^2 \cdot T_1) + (Q_y^2) + T_3$
ciclo 6:	$T_2 = P_x \cdot Q_z + Q_x$
ciclo 7:	$Q_y = (T_3 + Q_z) \cdot T_2$
ciclo 8:	$Q_y + Q_z^2 \cdot (P_x + P_y)$

Procedimiento 24 Doblado de punto es coordenadas LD: Versión Paralela

Entrada:	$Q = (P_x : P_y : 1)$ en coordenadas LD sobre K-233: $f(z) = z^{233} + z^{74} + 1$ / $GF(2^{233})$
Salida:	$2Q = (Q_x : Q_y : Q_z)$ en coordenadas LD.
ciclo 1:	$Q_z = P_x^2$.
ciclo 2:	$Q_x := Q_z^2 + 1$.
ciclo 3:	$Q_y := Q_x \cdot (P_y^2 + 1) + Q_z$.

campo finito $GF(2^{233})$. Se utilizan los registros Qx, Qy y Qz para almacenar un punto elíptico y registros de propósito general Tn.

Usando los Algoritmos 23 y 24, una suma de puntos elípticos puede ser calculado en 8 ciclos de reloj y un doblado de punto en 3 ciclos de reloj.

Se pueden localizar 3 funciones atómicas básicas para el diseño de una arquitectura de hardware:

1. $Q_x = (Q_x^2 \cdot T_1) + (Q_y^2) + T_3$: Se observa una adición entre tres operandos uno de ello es un multiplicación y cualquier operando debe ser posible de ser elevado al cuadrado.

2. $Q_y = (T_3 + Q_z) \cdot T_2$: Se observa una multiplicación entre dos operandos, donde uno de ellos es una adición.

3. $T_1 = Q_z \cdot Q_x$ en paralelo con $Q_z = T_1^2$: Una multiplicación es realizada al mismo tiempo que otro registro es elevado al cuadrado. Esta operación se considera como una copia entre registros con una elevación al cuadrado, debido a la sencillez de esta operación.

4. Los destinos de las *copias-cuadrados* son siempre registros Q_n.

3.3. Multiplicación Escalar

Los algoritmos propuestos por Solinas en [2] y [3] para el cálculo de una multiplicación escalar, una vez que la expansión τNAF del escalar se ha obtenido, solamente se diferencían por los múltiplos que se usan de P. Ambos algoritmos para calcular la multiplicación escalar a partir de la expansión WτNAF se basa en su expresión de Horner.

El algoritmo se compone por una sección de precómputo y finalmente por el cálculo de la expresión de Horner de acuerdo a la expansión WτNAF obtenida previamente.

El Algoritmo 25 primero calcula a $2^{w-1} - 1$ múltiplos de P. Los múltiplos usados dependen del algoritmos utilizado de los dos propuestos por Solinas. Cuando el punto P es fijo, es posible almacenar estos puntos en memoria para mejorar el desempeño del algoritmo.

En ECDSA, el algoritmo generador de firmas digitales siempre realiza la multiplicación escalar en el punto base de la curva elíptica $P \in E(F_q)$. Dado que el desarrollo está enfocado a la generación de firmas, es posible precalcular los correspondientes múltiplos y almacenarlos en memoria.

Procedimiento 25 Método TNAF con ventana de w bits para mutiplicación escalar elíptica

Entrada: $TNAF_w(\rho') = \sum_{i=0}^{l-1} u_i \tau^i$ donde $\rho' = k \bmod \delta$, $P \in E_a(F_{2^m})$
Salida: kP

if Se usa algoritmo Solinas en [2] **then**
  Calcular $P_u = uP$, para $u \in \{1, 3, 5, ..., 2^{w-1} - 1\}$
else if Se usa algoritmo Solinas en [3] **then**
  Calcular $P_u = \alpha_u P$, para $u \in \{1, 3, 5, ..., 2^{w-1} - 1\}$ donde $\alpha_i = i \bmod \tau^w$ for $i \in \{1, 3, ..., 2^{w-1} - 1\}$
$Q \leftarrow \vartheta$
for i from $l - 1$ downto 0 **do**
  $Q \leftarrow \tau Q$
  if $u_i \neq 0$ **then**
    Sea u tal que $\alpha_u = u_i$ o $\alpha_{-u} = -u_i$
    if $u > 0$ **then**
      $Q \leftarrow Q + P_u$
    else
      $Q \leftarrow Q - P_{-u}$
Regresa Q

El Algoritmo 25 tiene que ser optimizado para su implementación en hardware. Este proceso sigue el mismo formato que los anteriores algoritmos, es decir, se buscan funciones que puedes ser ejecutadas de manera atómica en un solo ciclo de reloj tomando en cuenta la cantidad de recursos que toman operaciones clave, como puede ser la suma de campo y el elevar al cuadrado.

El Algoritmo 20 provee una expansión del escalar k con la forma: w_0, z_0, w_1, z_2, ..., z_{i-1}, w_i donde $w_i \in [1, 3, 5, \ldots, 2^{w-1} - 1]$ y $z_i \in [w - 1, 2w - 2]$. Es decir las secuencias de ceros consecutivos son representadas por su extensión. Esto facilita el cálculo de una secuencia de ceros completa en un solo ciclo de reloj.

El Algoritmo 26 está diseñado para su implementación en hardware con la capacidad de calcular varias elevaciones al cuadrado en secuencia en un solo ciclo de reloj.

Procedimiento 26 Método TNAF con ventana de w bits para mutiplicación escalar elíptica: Versión Hardware

Entrada: $W\tau NAF(\rho')$ dado como: $w_0, z_0, w_1, z_2, \ldots, z_{i-1}, w_i$ donde $w_i \in [1, 3, 5, \ldots, 2^{w-1} - 1]$ y $z_i \in [w - 1, 2w - 2]$
Algoritmo [2] Almacenamiento no-volátil: $P_u = uP$, for $u \in \{1, 3, 5, ..., 2^{w-1} - 1\}$
Algoritmo [3] Almacenamiento no-volátil: $P_u = \alpha_u P$, for $u \in \{1, 3, 5, ..., 2^{w-1} - 1\}$
Salida: kP

1: **if** $R_0 \neq 0$ o $R_1 \neq 0$ **then**
2: $Q \leftarrow \vartheta$
3: **for** i from $\frac{2l}{w-1} - 1$ downto 0 **do**
4: **if** i es impar **then**
5: $Q \leftarrow \tau^{w-1} Q$
6: $W\tau NAF_i \leftarrow W\tau NAF_i - (w - 1)$
7: **if** $W\tau NAF_i \neq 0$ **then**
8: $Q \leftarrow \tau^{w-1} Q$
9: **else**
10: Direcciona P_u con $W\tau NAF_j$
11: $Q \leftarrow \tau Q$
12: **if** $u > 0$ **then**
13: $Q \leftarrow Q + P_u$
14: **else**
15: $Q \leftarrow Q - P_{-u}$

3.4. Resumen

Los algoritmos implementados dentro de nuestra aquitectura de hardware reconfigurable son:

- Los Algoritmos 17 y 18 están implementados para realizar el proceso de reducción parcial. El Algoritmo 18 está diseñado para calcularse en un ciclo de reloj.
- Los Algoritmos 19 y 20 calculan la expansión WτNAF a partir de diferentes representantes de clase y su tabla de precómputo es distina. Su diseño está pensado para trabajar de manera coordinada con el diseño en hardware propuesto en el Capítulo 3 con dos etapas de pipeline.
- Los Algoritmos 23 y 24 implementan los algoritmos de ley de grupo (suma y doblado) en curvas elípticas propuestos por López y Dahab en coordenadas proyectivas. Su optimización principal es el uso de una multiplicación de campo por ciclo de reloj.
- El Algoritmo 26 está diseñado para leer los múltiplos de precómputo y los elementos de la expansión WτNAF de bloques de memoria.

Cada algoritmo fué diseñado para calcular la mayor cantidad de operaciones por ciclo de reloj, de acuerdo con la arquitectura presentada en el Capítulo 3.

Capítulo 4

Arquitectura de Hardware Reconfigurable

Los algoritmos para realizar una multiplicación basados en expansiones NAF del factor pueden ser divididos en dos etapas: la expansión del factor escalar en coeficientes NAF y el cálculo de la expresión de Horner correspondiente.

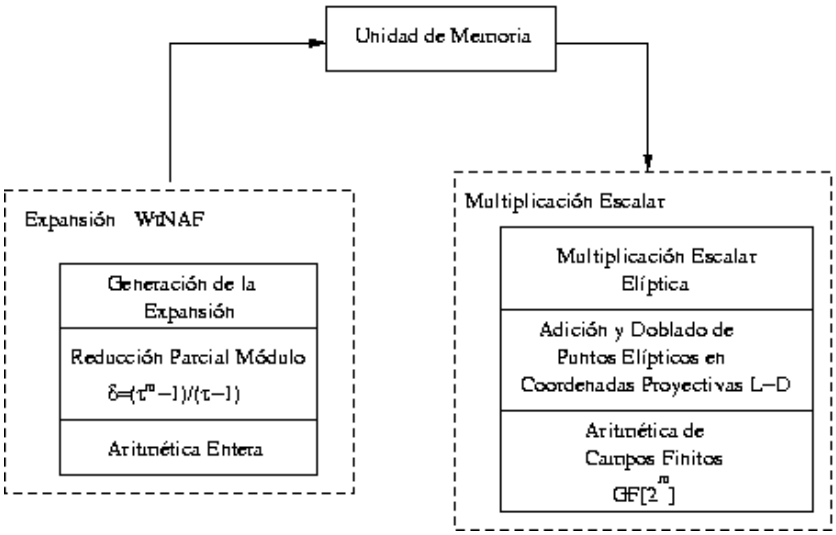


Figura 4.1: Diseño de Arquitectura para Multiplicación Escalar

Para una multiplicación escalar de curvas elípticas, dicha división implica una clasificación de la aritmética inherente a cada proceso. En la arquitec-

tura presentada a continuación se explota dicha clasificación al diseñar dos unidades aritméticas independientes trabajando de manera coordinada.

4.1. Sistema Expansión-Multiplicación

La arquitectura propuesta para el cálculo de una multiplicacióne escalar elíptica basada en un algoritmo WτNAF se basa en un esquema productor consumidor.

Se presentan dos unidades de proceso trabajando de forma independiente. La primera unidad de proceso está diseñada para realizar los algoritmos dedicados a realizar la expansión de un factor escalar de 233 bits en coeficientes de una expresión polinomial $\mathbb{Z}[\tau]$ con una ventana de 8 bits. Los algoritmos para los cuáles la arquitectura ha sido optimizada son los Algoritmos 17 y 20 presentados en el capítulo 3.

La segunda unidad de proceso está dedicada a realizar la expresión de Horner correspondiente a la expansión WτNAF y obtener el resultado de la multiplicación escalar. Los Algoritmos 23, 24 y 26 son la base para optimizar dicha unidad de proceso.

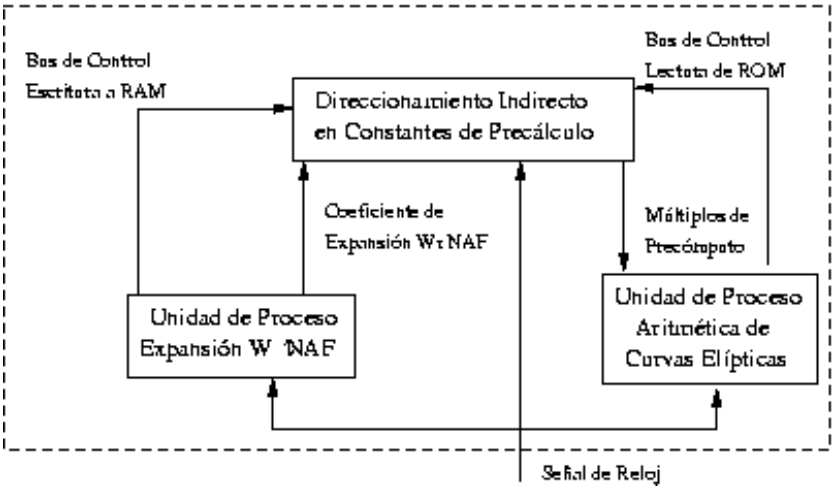


Figura 4.2: Arquitectura para Multiplicación Escalar

El Algoritmo 26 requiere de la expansión WτNAF como entrada para poder calcular la expresión de Horner correspondiente. El Algoritmo 20 genera dicha expansión del factor escalar empezando por el coeficiente menos

significativo. Sin embargo el Algoritmo 26 requiere leer el coeficiente más significativo al inicio de su cómputo. Esta dependencia entre ambas etapas impone una restricción importante. La unidad de proceso de curvas elípticas (UPCE) necesita esperar a que la unidad de proceso de expansión τ (UPEτ) genere la expansión por completo.

Esto implica que se pueden coordinar mediante un bloque de memoria intermedio dedicado a almacenar la expansión generada por UPEτ para finalmente ser leída en orden inverso por UPCE. La unidad de proceso de curva elípticas no iniciará su algoritmo hasta que la unidad de proceso de expansión τ lo indique a través de una línea de control.

La longitud de una expansión τNAF de ventana w es $l \leq m + a$ para la curva $E_a(\mathbb{F}_{2^m})$. Una característica importante en una expansión τNAF con una ventana de w bits es que de cualquier w dígitos consecutivos a lo más uno no es cero. Esto implica que cuando encontremos un coeficiente cero en la expansión, entonces deberá existir una cadena completa de $w-1$ ceros consecutivos mínimo. Por lo tanto, los ceros consecutivos por cadena pueden ser contados y almacenar el resultado como un solo elemento. Sean los coeficientes no-cero de una expansión WτNAF representados por: $w_i \in [1, 3, 5, \ldots, 2w-1-1]$. La expansión generada es un conjunto de elementos w_i y cadenas de cero consecutivas: $w_0, 0 \ldots 0, w_1, 0 \ldots 0, w_2, \ldots, w_{i-1}, 0 \ldots 0, w_i$. Si los ceros dentro de cada secuencia de ceros consecutivos son contados, la secuencias de ceros consecutivos pueden representarse como: $z_i \in [w - 1, 3w - 3]$. La expansión WτNAF es representada y es almacenada como: $w_0, z_0, w_1, z_1, \ldots, z_{i-1}, w_1$.

Los coeficientes de la expansión son almacenadas en una memoria RAM de 8x64 bits. Cada coeficiente puede ser representado como una palabra de 8 bits y se realizaron análisis estadísticos, los cuáles se presentan en el apéndice B, dando como resultado que 64 bloques de memoria son suficientes para almacenar cualquier expansión NAF. Las memorias RAM son costruidas a partir de bloques básicos BRAM integrados en las plataformas FPGA de Xilinx.

Un algoritmo basado en una ventana de precálculo ofrece mejoras en tiempos de cómputo significativas al procesar un conjunto de w bits como una unidad. El desempeño es optimizado al precalcular y almacenar de manera estática múltiplos del punto elíptico sobre el cuál se realiza la multiplicación escalar. El Algoritmo 26 requiere que, dada una ventana de precómputo de w bits, el punto P y sus múltiplos $3P, 5P, \ldots, (2^{w-1} - 1)P$ sean almacenados previamente. La arquitectura presentada fué optimizada para una ventana de

precómputo de $w = 8$ bits, por lo tanto, una memoria ROM de 233x64 bits por coordenada es suficiente para almacenar los 64 múltiplos precalculados. Dado que los múltiplos de P son almacenados en coordenadas afines, bastan con dos memorias ROM para almacenar los puntos. Los múltiplos almacenados corresponden a múltiplos del punto generador de la curva $K-233$, como se requiere en el algoritmo generador de firma digital.

El Algoritmo 26 accede a los múltiplos de P precalculados a partir de los coeficientes de la expansión WτNAF de manera indirecta. Cada coeficiente distinto de cero representa el múltiplo que debe ser sumado en la expresión de Horner. Por lo tanto, el calcular dicho múltiplo dentro de la arquitectura consiste en direccionar el múltiplo al que hace referencia el coeficiente actual dentro del algoritmo. Sin embargo, los múltiplos que han sido almacenados son impares, por lo tanto, es necesario realizar un ajuste de índices para que correspondan las localidades de memoria. El índice i donde se almacena el múltiplo uP se obtiene:

$$i = \lfloor \frac{1}{2}(u-1) \rfloor \tag{4.1}$$

A través de dicha relación es posible indexar el múltiplo correspondiente. El bloque presentado en la Figura 4.3 engloba las localidades de memoria para las constantes de precálculo y los coeficientes de la expansión.

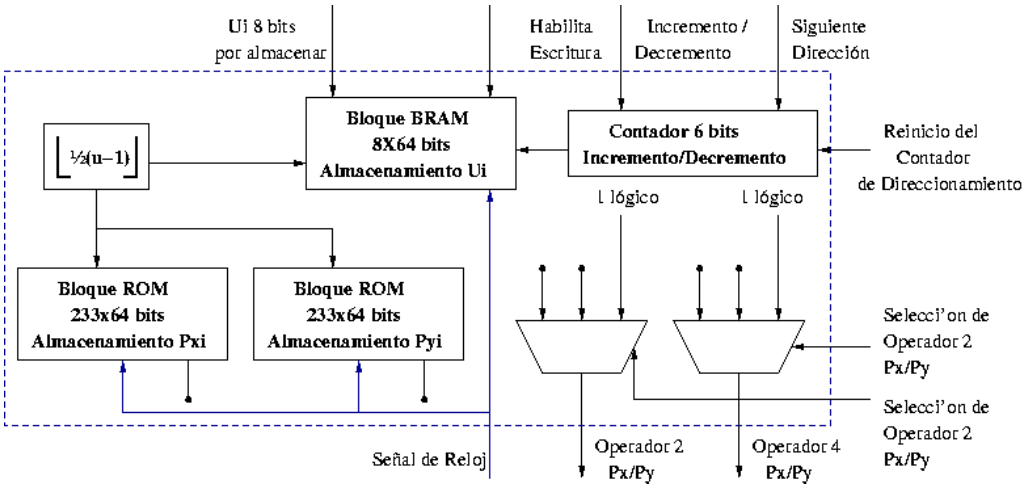


Figura 4.3: Direccionamiento Indirecto de Constantes Precalculadas

El direccionamiento indirecto de los múltiplos se realiza internamente. Un contador permite almacenar de manera secuencial los coeficientes al ser calculados. El mismo contador y la aritmética de direccionamiento indirecto, permite la lectura de los múltiplos requeridos por el Algoritmos 26 de manera secuencial, en el mismo orden en que son requeridos. De esta forma dicho algoritmo accesa a los múltiplos de P de manera transparente, nunca conoce los elementos de la expansión pero es capaz de leer los múltiplos que corresponden en cada iteración de la expresión polinomial τ de Horner de kP.

4.2. Unidad Principal de Proceso: Expansión

La unidad de proceso dedicada a la generación de la expansión WτNAF está compuesta por: una unidad aritmética que engloba las operaciones en aritmética entera para el cómputo de los Algoritmos 17 y 20 así como un bloque de registros. Una máquina de estados la cual contiene la microprogramación de los algoritmos correspondientes en palabras de un bus de control de 18 bits y la lógica de control para contar secuencias de ceros en la expansión generada.

El contador de ceros permite acumular la cantidad de ceros cada que empieza una secuencia consecutiva de ellos. La máquina de estados controla a través del bus *Selección Ui* el valor que es exteriorizado, el coeficiente NAF si es distinto de cero o el conteo de los ceros que se ha acumulado. La máquina de estados rige la secuencia presente en el *Bus de Control*, el cuál provee a la unidad aritmética y logíca de los algoritmos para la generación de la expansión.

Las terminales externas que permiten la interacción con la unidad de proceso son:

- *Señal de Reloj:* Señal de reloj utilizada en los dispositivos síncronos de la arquitectura, como los son registros, *latches*, el contador y la máquina de estados.
- *Reestablecer:* Señal que en transición positiva indica a la máquina de estados su regreso al estado inicial y reestablece el valor original de los registros internos.
- *Listo:* Señal que en estado alto indica éxito en la generación de todos los coeficientes de la expansión WτNAF. La unidad de proceso de curvas

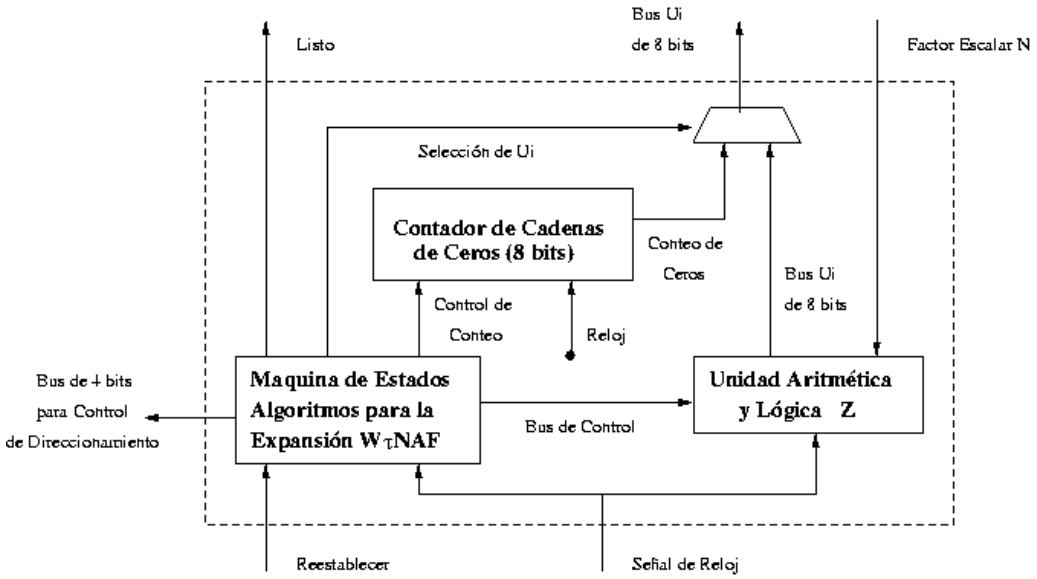


Figura 4.4: Unidad Principal de Proceso: Expansión τ NAF

elípticas interpreta la señal como un permiso para comenzar a leer la expansión y realizar la multiplicación.

- *Factor Escalar N:* Bus de 233 bits sobre el cuál es necesario tener presente el factor escalar por el cuál se desea multiplicar al punto generador de la curva $K-233$.
- *Bus U_i :* Bus de 8 bits en el cuál se transmite cada coeficiente que es generado.
- *Bus de Control de Direccionamiento:* Bus de Control de 4 bits destinado a controlar la escritura en RAM en el bloque de direccionamiento indirecto mostrado en la Figura 4.3 de los coeficientes NAF generados.

4.2.1. Unidad Aritmética y Lógica: Expansión WτNAF

La unidad aritmética y lógica (UAL) para los algoritmos de expansión fué diseñada para ejecutar un conjunto de micro operaciones las cuáles expresen todas la funciones atómicas presentadas en los Algoritmos 17, 19 y 20 presentados en el capítulo 3.

Dichas micro operaciones son ejecutadas dentro de un esquema de *pipe-line* de dos etapas: *Selección de Operadores-Cómputo Aritmético* y *Actualización de Registros*.

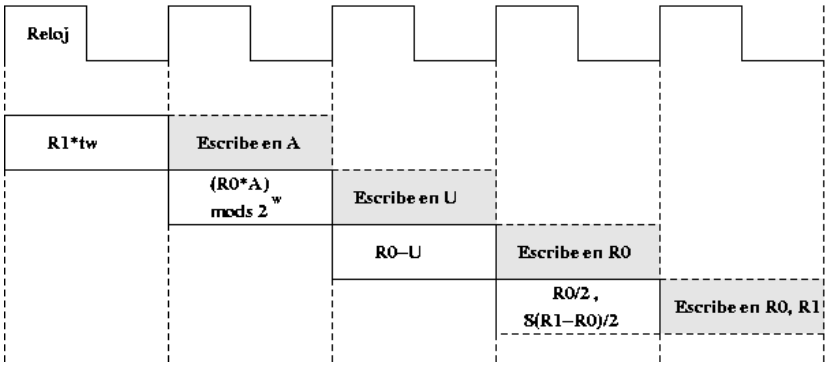


Figura 4.5: Flujo de ejecución en Pipe-Line de dos etapas

Este proceso de *pipe-line* implica problemas de lectura después de escritura *RAW* (*Read-After-Write*) inherentes a la arquitectura. La forma más simple de enfrentar dichos problemas es la inserción de ciclos inactivos (también conocidos como *burbujas de tiempo*) en la microprogramación. Sin embargo dicho problema fué abordado y resuelto al momento del diseño de algoritmos paralelos en el capítulo 3, evitando inclusive burbujas de tiempo.

Las micro operaciones realizadas por la unidad aritmética y lógica son establecidas por un bus de control de 19 bits. La unidad aritmética lógica es presentada en la Figura 4.6. Los operadores principales son la multiplicación de 128 bits y la adición-substracción de 256 bits. Dentro de los bloques dedicados a la selección de sus operandos, se encuentran operaciones simples basadas en corrimientos y reordenamiento de bits.

Las comparaciones realizadas en los algoritmos son calculadas en el mismo ciclo correspondiente a la aritmética utilizando comparadores dedicados dentro de la unidad de registros. De esta manera es posible omitir un ciclo de reloj independiente para cada comparación.

Los bloques básicos que constituyen la unidad aritmética son:

- *2xROM 5x64:* $\alpha \ mod \ \tau^w$: Se tienen dos bloques de memoria ROM de 64 elementos de 5 bits cada uno. Dichos bloques almacenan las componentes constantes $\alpha_u = u \bmod \tau^w = \beta_u + \gamma_u \tau$ utilizados al implementar el Algoritmo 20.

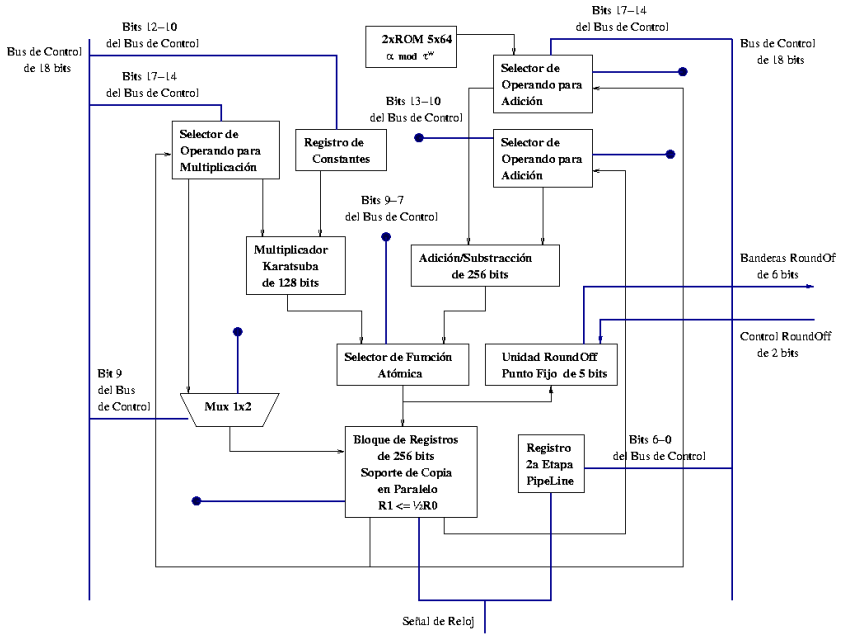


Figura 4.6: Unidad Aritmética y Lógica: Expansión τ NAF

- *Registro de Constantes*: Dentro de este registro se encuentran almacenadas todas las constantes utilizadas a través de los Algoritmos 17 y 20. Las constantes almacenadas y su respectivo código de control, así como las respectivas terminales del bus de control se presentan en la Tabla 4.1.

Cuadro 4.1: Bus de Control: Bits 12-10

Bits 12-10 del Bus de Control	Nombre de la Constante
000	s0
001	s1
010	Vm
011	d0=s0+s1
100	tw

- *Selección del Operando Izquierdo para el Multiplicador*: Este bloque determina el operando izquierdo de 128 bits para el multiplicador. Nótese que el operando derecho siempre corresponde a una constante. Este operador es seleccionado entre el factor escalar N y los registros R_0, R_1, A o U. Dos funciones de corrimiento son incluidas en el bloque. Es posible realizar un corrimiento de 107 bits a la derecha sobre el factor

escalar N o un corrimiento de 233 bit a la derecha al operador que tenga como fuente uno de los registros. El código de control es descrito por la Tabla 4.2.

Cuadro 4.2: Bus de Control: Bits 18-15

Bits 18-15 del Bus de Control	**Operando Seleccionado**
00XX	n
10XX	$\lfloor \frac{n}{2^{107}} \rfloor$
01CR	$CodigoRegistro$
10CR	$\lfloor \frac{CodigoRegistro}{2^{233}} \rfloor$

los registros son indexados con el código mostrado en la Tabla 4.3.

Cuadro 4.3: Código de Registros (CR)

Código de Registros (CR)	**Nómbre del Registro**
00	R0
01	R1
10	A
11	U

- *Multiplicador de 128 bits*: El multiplicador está construido a partir de tres multiplicadores de 64 bits generados en VHDL con la herramienta proporcionada por Xilinx *Core Generator*. Dichos multiplicadores están basados en los multiplicadores de 18 bits incluidos dentro de los FPGA de Xilinx. Esto disminuye el uso de unidades básicas (*slices*) del dispositivo. El múltiplicador de 128 bits es construido a partir de una arquitectura Karatsuba-Offman.

- *Selección de los Operandos para la Adición-Substracción*: Estos bloques seleccionan los operandos de 256 bits del bloque sumador-restador. El operando es seleccionado entre el factor escalar N o algún registro interno. El operando sleccionado puede ser multiplicado por -1, $\frac{1}{2}$ o ambos. El código para los buses de control se muestra en las Tablas 4.4 y 4.5.

Cuadro 4.4: Bus de Control: Bits 19-15

Bits 19-15 del Bus de Control	**Nombre de la Constante**
[0-1:$\frac{1}{2}$]00XX	n
[0-1:$\frac{1}{2}$]01XX	$-n$
[0-1:$\frac{1}{2}$]01CR	$CodigoRegistro$
[0-1:$\frac{1}{2}$]11CR	$-CodigoRegistro$

Cuadro 4.5: Bus de Control: Bits 14-10

Bits14-10 del Bus de Control	**Nombre de la Constante**
[0-1:$\frac{1}{2}$]0000	1
[0-1:$\frac{1}{2}$]0100	-1
[0-1:$\frac{1}{2}$]0001	β
[0-1:$\frac{1}{2}$]0101	$-\beta$
[0-1:$\frac{1}{2}$]0010	γ
[0-1:$\frac{1}{2}$]0110	$-\gamma$
[0-1:$\frac{1}{2}$]01CR	$CodigoRegistro$
[0-1:$\frac{1}{2}$]11CR	$-CodigoRegistro$

- *Adición-Substracción de 256 bits* El sumador-substractor de 256 bits fué generado en código VHDL a partir de la herramienta *CoreGenerator* proporcionada por Xilinx.

- *Selección de Función Atómica*: Este bloque selecciona la operación principal, multiplicación o adición, y permite operaciones simples sobre su resultado. Dichas operaciones son realizadas mediante corrimiento, selección y reordenamiento de bits. Una vez seleccionada la multiplicación como operación principal, el resultado puede ser multiplicado por 2, dividido entre 2 o incluir una operación *mod* 4. La adición-substracción puede ser el operando en un corrimiento de 214 bits a la derecha o de una operación *mods* 256.

Cuadro 4.6: Bus de Control: Bits 9-7

Bits 9-7 del Bus de Control	**Función Seleccionada**
000	$OP1 * OP2$
001	$2 * OP1 * OP2$
010	$\frac{OP1*OP2}{2}$
011	$(OP1 * OP2)\ mod\ 4$
100	$OP1 + OP2$
110	$\lfloor \frac{OP1+OP2}{2^{105}} \rfloor$
101	$(OP1 + OP2)\ mods\ 256$

- *Unidad de Round Off*: Esta unidad presenta aritmética de punto fijo, representando la parte fraccionaria de un cociente previo con 5 bits de resolución. La aritmética implementada corresponde al Algoritmo 18. Dado que la implementación está basada en aritmética de 8 bits, es posible realizar el algoritmo completo en un ciclo de reloj. La unidad cuenta internamente con 2 registros de 6 bits cada uno para almacenar las fracciones de λ_0 y λ_1. Una vez que ambas registros tienen su valor correpondiente almacenado, se necesita un ciclo de reloj para que

en el bus *Banderas RoundOff* se encuentren los valores h_0 y h_1, los cuales pueden ser -1, 0 ó 1, para poder realizar la operación *Round Off* correspondiente al vector (λ_0, λ_1).

- *Bloque de Registros 4x256 bits*: Dentro del bloque se encuentran 4 registros de 256 bits. Cada registro tiene un propósito específico dentro del desarrollo de los algoritmos de expansión y son nombrados de acuerdo a este. El registro A es accesado exclusivamente por la unidad aritmética y se dedica al almacenamiento de resultados parciales o temporales. Los registros R_0 y R_1 almacenan resultados intermedios del proceso de reducción del factor escalar y el resultado final (λ_0, λ_1). El registro U almacena cada coeficiente generado de la expansión por un ciclo de reloj. Dentro del bloque de registros se implementa una copia entre los registros R_0 y R_1 independiente al proceso de almacenamiento. Durante el proceso de copia el registro fuente puede ser multiplicado por -1, $\frac{1}{2}$ o ambos.

Cuadro 4.7: Control de Registro Destino

Bits 6-5 del Bus de Control	**Registro Destino**
CR	Registro Destino

Cuadro 4.8: Control de Registros

Bit 4 del Bus de Control	**1 : Reinicia registros**
Bit 3 del Bus de Control	**1 : Inhabilita Registros**

Cuadro 4.9: Control de Copia entre Registros

Bit 2 del Bus de Control	**1 : Copia R_0 a R_1**
Bit 1 del Bus de Control	**1 : Multiplica R_0 por -1 durante la copia**
Bit 0 del Bus de Control	**1 : Multiplica R_0 por $\frac{1}{2}$ durante la copia**

4.2.2. Unidad de Registros 256x4 bits: Copia Paralela $R_1 \leftarrow -\frac{1}{2}R_0$

En el Algoritmo 16 propuesto por Solinas presenta en la línea 10 una operación sobre un vector. La operación es realizada de manera independiente sobre cada coordenada $(r_0, r_1) \leftarrow (r_1 + \frac{\mu r_0}{2}, \frac{-r_0}{2})$. Dicha operación está dentro de un ciclo el cual se espera realice cerca de m iteraciones para una curva

elíptica definida sobre el campo finito $GF(2^m)$. En el Algoritmo 20 propuesto en el capítulo 3, dicha operación es realizada en un solo ciclo de reloj en la línea 15.

Para poder realizar dicha operación en un solo ciclo de reloj es necesario tener la capacidad de almacenar los registros R_0 y R_1 simultáneamente, lo cuál es imposible con un solo bus de acceso al bloque de registro. El duplicar dicho bus es considerado costoso si se le da acceso a todos los registros.

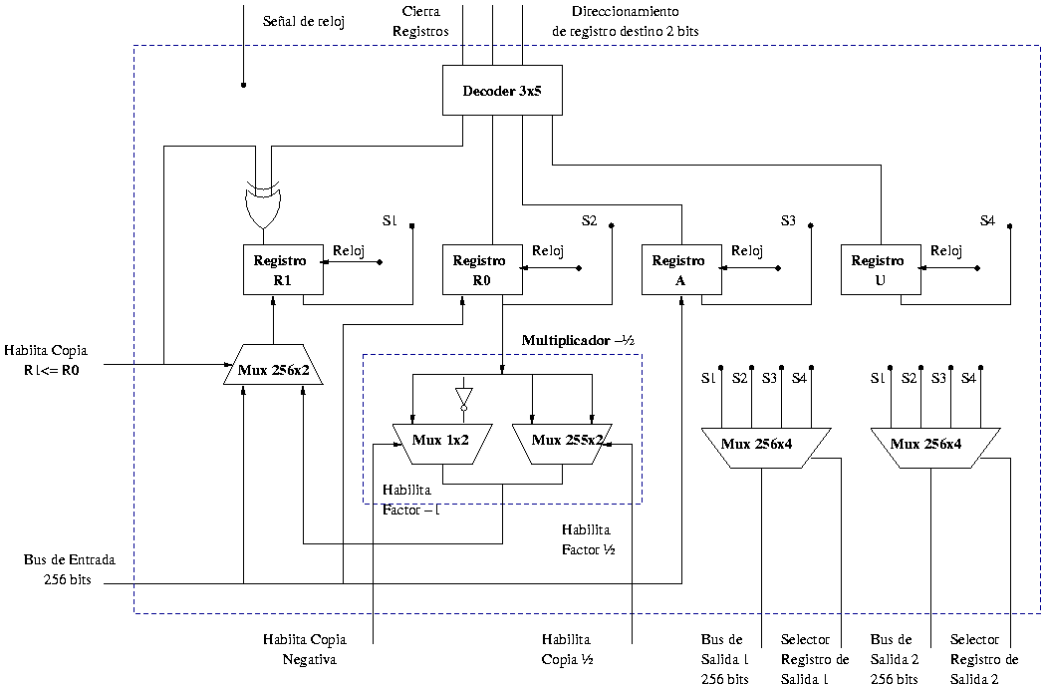


Figura 4.7: Bloque de Registros con soporte de copia paralela

Para evitar el duplicado del bús se da acceso al registro R_0 de copiarse al registro R_1 de manera interna al bloque de registros. Esto simplifica el diseño y elimina el costo de la lógica requerida por un segundo bus externo con acceso a todos los registros.

La lógica para copiar dicho registro incluye la aritmética necesaria para realizar la multiplicación por el factor $\lfloor -\frac{1}{2}R_0 \rfloor$. Dicha lógica es simple y es implementada a partir de dos multiplexores y un inversor, como puede apreciarse en la Figura 4.7.

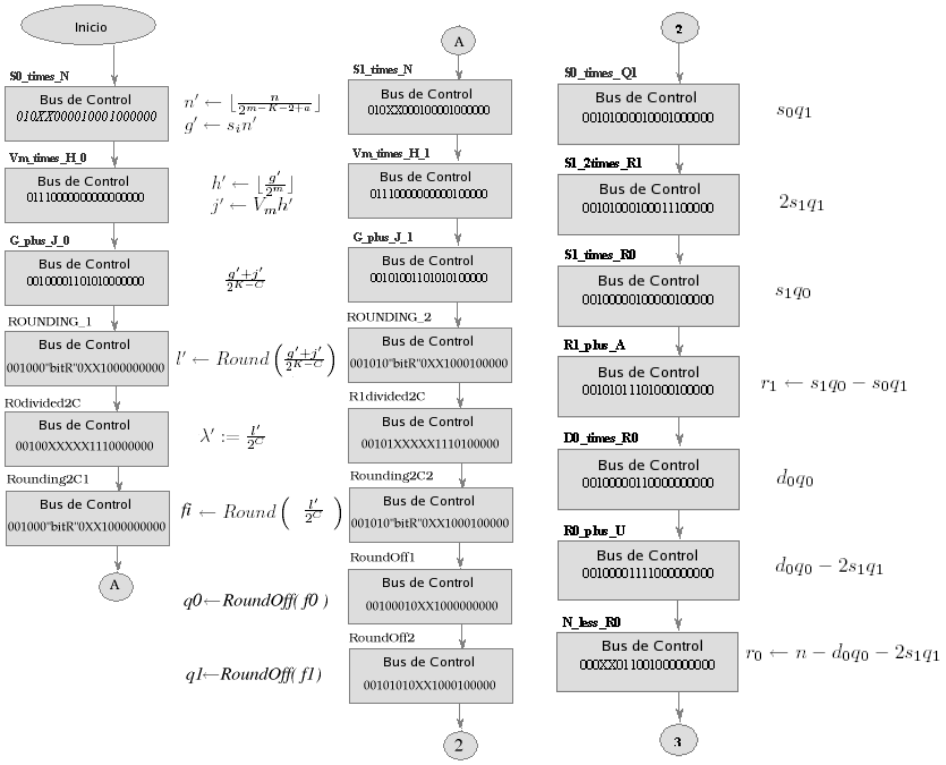


Figura 4.8: Máquina de Estados: Reducción Parcial módulo δ

4.2.3. Máquina de Estados

La unidad aritmética y lógica está controlada por una máquina de estados. Este conjunto de estados generan las señales correspondientes en el bus de control para realizar el Algoritmo 17 y los dos diferentes algoritmos de Solinas 19 y 20. El algoritmo encargado de realizar la reducción parcial del factor escalar ha sido modelado en la máquina de estados de la Figura 4.8.

De esta forma, el proceso de reducción es posible realizarlo en 13 ciclos de reloj. Los estados de la máquina son los siguientes:

- INICIO: Estado de espera. Una señal externa le indica el inicio del proceso.

- $S_0\_times\_N$: Calcula el paso 1 del Algoritmo 17 con s_0 como una constante.

- $V_m\_times\_H\_0$: Calcula el paso 2 del Algoritmo 17 con V_m como una constante.
- $G\_plus\_J\_0$: Calcula el paso 3 del Algoritmo 17.
- $ROUNDING1$: Calcula el paso 3 del Algoritmo 17.
- $R0divided2C$: Calcula el paso 3 del Algoritmo 17.
- $Rounding2C1$: Prepara el último paso del Algoritmo 14.
- $S_1\_times\_N$: Calcula el paso 4 del Algoritmo 17 con s_1 como una constante.
- $V_m\_times\_H\_1$: Calcula el paso 5 del Algoritmo 17 con V_m como una constante.
- $G\_plus\_J\_1$: Calcula el paso 6 del Algoritmo 17.
- $ROUNDING2$: Calcula el paso 6 del Algoritmo 17.
- $R1divided2C$: Calcula el paso 6 del Algoritmo 17.
- $Rounding2C2$: Prepara el último paso del Algoritmo 14.
- $RoundOff1$: Calcula el Algoritmo 18.
- $RoundOff2$: Calcula el Algoritmo 18.
- $S_0\_times\_Q_1$: Calcula el paso 7 del Algoritmo 17 con s_0 como una constante.
- $S_1\_2times\_R_1$: Calcula el paso 8 del Algoritmo 17 con s_1 como una constante.
- $S_1\_times\_R_0$: Calcula el paso 9 del Algoritmo 17 con s_1 como una constante.
- $R_1\_plus\_A$: Calcula el paso 10 del Algoritmo 17, obtiene R1 reducido.
- $D_0\_times\_R_0$: Calcula el paso 11 del Algoritmo 17 con d_0 como una constante.
- $R_0\_plus\_U$: Calcula el paso 12 del Algoritmo 17.

- $N\_less\_R_0$: Calcula el paso 13 del Algoritmo 17, obtiene R0 reducido.

Solinas publicó dos algoritmos para calcular la multiplicación escalar con ventanas de precómputo en [2] y [3]. Ambos algoritmos fueron adaptados e implementados en la misma arquitectura de hardware reconfigurable.

El Algoritmo 19 implementa el algoritmo propuesto en [2] y es programado a partir de la máquina de estados descrita con el diagrama de flujo presentado en la Figura 4.9. Todas las comparaciones que se encuentran en el algoritmo son realizadas en el mismo ciclo de reloj de ejecución. Las comparaciones son lo suficientemente simples para no afectar el tiempo de propagación.

El diagrama de flujo se compone de 6 estados principales. Esta máquina de estados permite obtener un coeficiente cero de la expansión en un ciclo de reloj y un coeficiente distinto de cero en 4 ciclos de reloj. Los estados son los siguientes:

- $COMIENZA\_EXPANSION$: La comparación del paso 1 y los pasos 2 y 14 están implementados en este estado. Note que las dos posibles opciones de la comparación se encuentran incluidas.

- $NUEVA\_U$ Estado que realiza el paso 3 y 9.

- $ACTUALIZA\_R_0$: Estado que realiza el paso 4 y 10.

- $ACTUALIZA\_R_0R_1A$: Estado que realiza el paso 5 y 11.

- $DECISION$: Estado que implementa las comparaciones presentes en los pasos 6 y 7 del Algoritmo 20 y realiza los pasa 8,13 y 14. 13 y 14 son realizados en el mismo ciclo.

- $FINAL$: Estado Final. La expansión se ha terminado.

El algoritmo propuesto en [3] es adaptado en el Algoritmo 20 y es programado según la maquina de estados presentada en la Figura 4.10.

Dicha máquina de estados implementa dos ciclos de reloj adicionales, incluyendo la adición de la componente compleja $r_1 = r_1 - \gamma u$ y un ciclo de direccionamiento a la memoria ROM $DIRECCIONA\_ALPHAu$ para acceder a los elementos β_u y γ_u

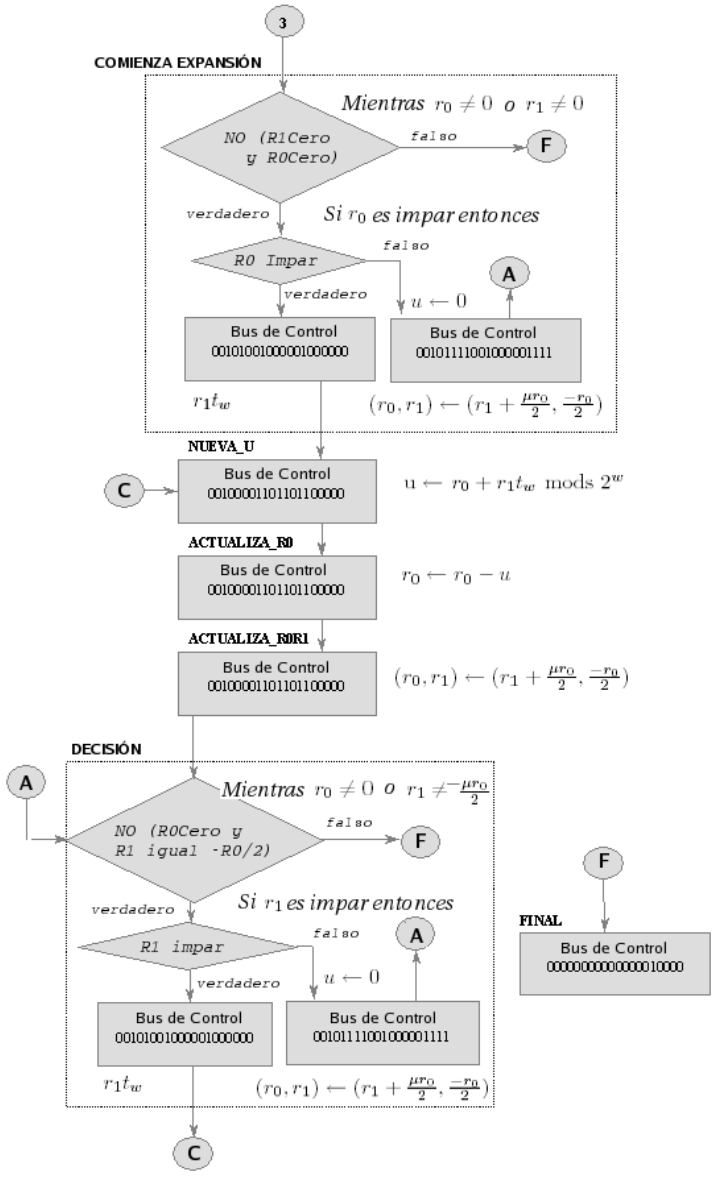


Figura 4.9: Máquina de Estados: Expansión WτNAF.

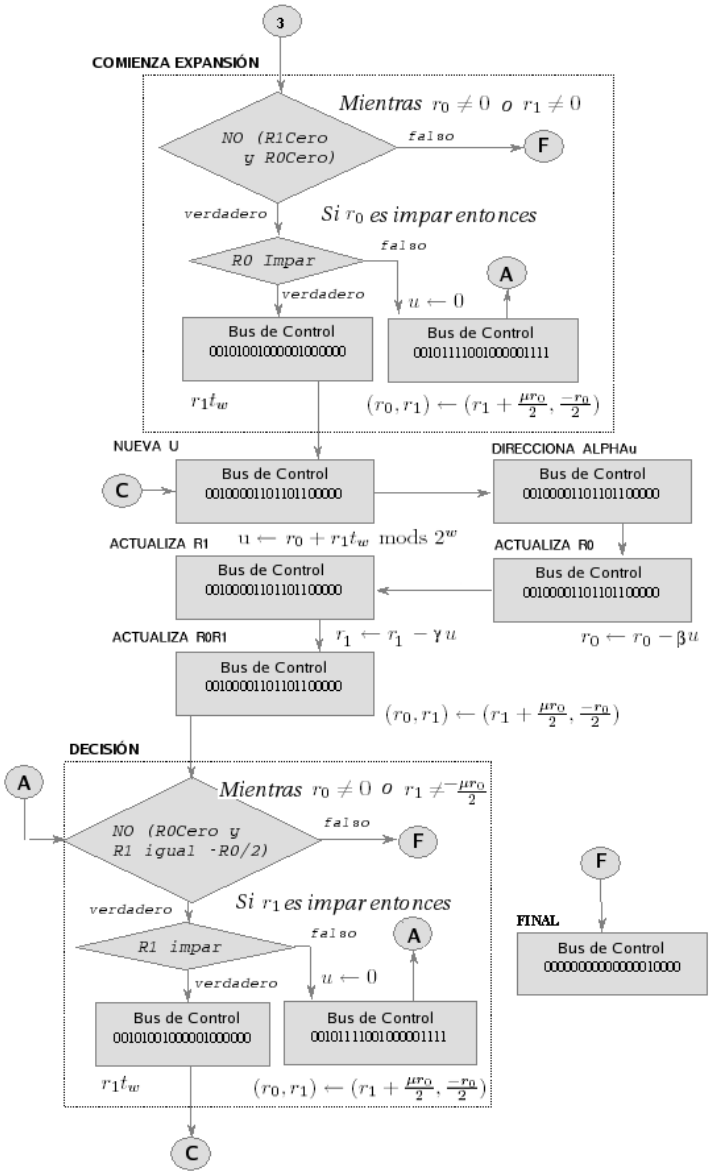


Figura 4.10: Máquina de Estados: Expansión WτNAF.

4.3. Unidad Principal de Proceso: Curvas Elípticas

La aritmética base para esta unidad de control es la aritmética de campos finitos sobre la cuál se define una curva elíptica. Para nuestro caso de estudio,

se implementa la multiplicación escalar de un punto sobre la curva $K-233$. Dicha curva y su aritmética se define sobre el campo finito binario $GF(2^{233})$.

Los algoritmos que son implementados sobre esta arquitectura son tres. La ley de grupo sobre puntos de curvas elíticas implica dos algoritmos principales, la adición entre puntos y el doblado de un punto correspondientes a los Algoritmos 23 y 24. El tercer algoritmo a implementar es la interpretación de la expansión WτNAF como una expresión polinomial de Horn, propuesto en versión paralela en el Algoritmo 26.

En el capítulo 3 se presentaron estrucuturas para funciones atómicas necesarias en la implementación de algoritmos paralelos de suma, doblado y multiplicación escalar. En la presente sección proponemos una arquitectura de hardware capaz de calcular dichas estructuras por ciclo de reloj.

Otra característica importante incluida en esta arquitectura es la capacidad de aprovechar la secuencias de ceros consecutivos en una expansión WτNAF. Analizando el Algoritmo 26 se podrá corroborar que cada cero en la expansión corresponde con elevar al cuadrado cada coordenada de un punto elíptico. Una secuencia de $w-1$ ceros consecutivos implica elevar al cadrado $w-1$ veces consecutivas un punto. La arquitectura presentada, aprovecha un buen diseño de elevación al cuadrado en campos finitos binarios para calcular eficientemente 7 cuadrados en un mismo ciclo de reloj, suficiente para una ventana de precómputo de 8 bits.

La arquitectura se divide en una unidad aritmética y lógica de campos finitos y una máquina de estados coordinando la aritmética para implementar los algoritmos propuestos.

4.3.1. Unidad Aritmética y Lógica

En la Figura 4.11 se presenta una arquitectura aritmética capaz de calcular una amplia gama de funciones aritméticas en un solo ciclo de reloj. La arquitectura está planteada para trabajar en el mismo esquema de pipeline de dos etapas propuesto para la unidad de proceso principal dedicada a la expansión.

La unidad aritmética principal tiene un multiplicador de campo finito Karatsuba-Offman. Dicho multiplicador demanda cerca del 70 % de los recursos de hardware de la arquitectura. También se presentan sumadores de campo bastantes económicos en requerimentos de tiempo y espacio.

La unidad aritmética es controlada por un bus de control de 32 bits capaz de generar una amplia variedad de combinaciones de operaciones aritméticas

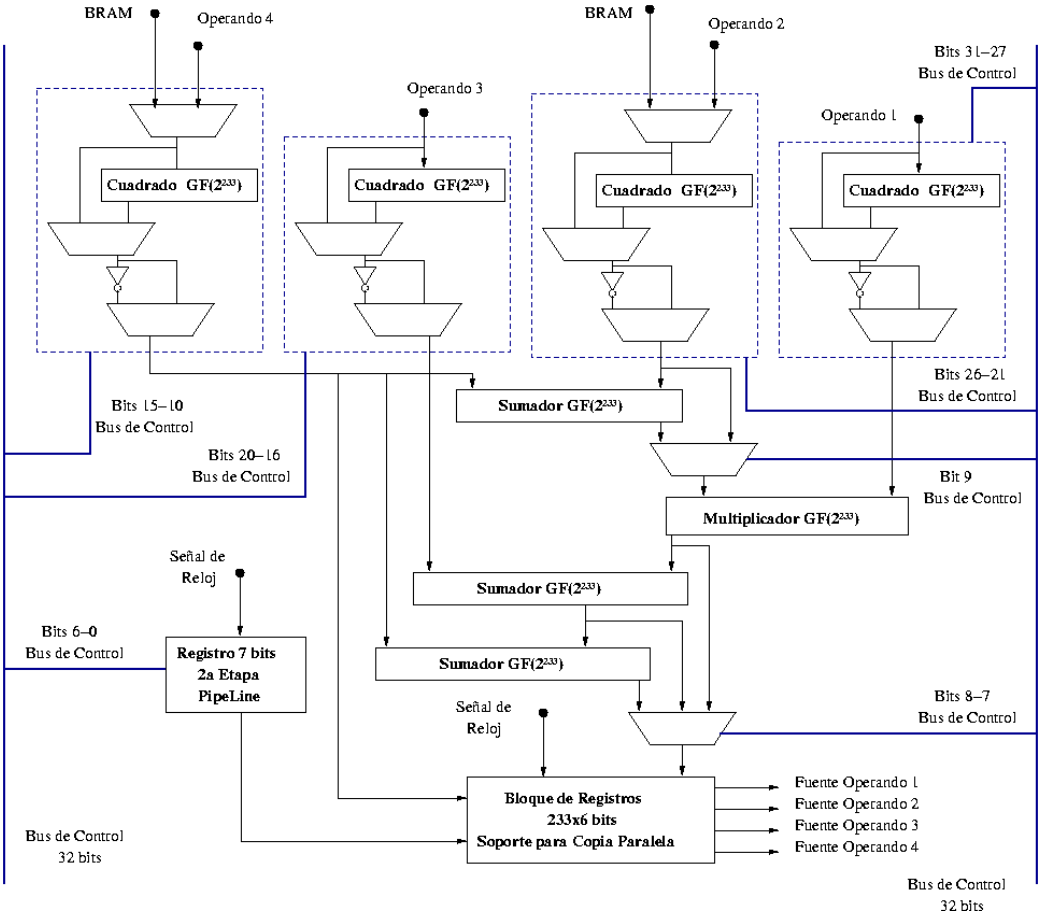


Figura 4.11: Unidad Aritmética y Lógica: Curvas Elípticas

con hasta 4 operandos como entrada. La unidad está adaptada para trabajar de manera conjunta con el bloque de direccionamiento indirecto y leer los múltiplos de P almacenados en su memoria ROM como posibles operandos.

Los bloques básicos dentro de esta unidad aritmética son:

- *Selector de Operando 1:* Este operando selecciona como operando 1 de una posible función compuesta cualquiera de los 6 registros internos de la unidad. Además eleva al cuadrado una o dos veces de manera opcional al operando seleccionado. Las terminales del bus de control correspondientes se presentan en la Tabla 4.10.

bit 31: Habilita Cuadrado	**bit 30:** Habilita Cuadrado	**bit 29-27** Código de Registro

Cuadro 4.10: Bits 31-27 del bus de Control: Aritmética de Curvas Elípticas

Es posible hacer referencia a los registros internos el código numérico presentado en la Tabla 4.11.

Código de Registro(CR)	**Nombre del Registro**
000	Q_x
001	Q_y
010	Q_z
011	T_1
100	T_2
101	T_3

Cuadro 4.11: Código de Registros Q_i

- *Selector de Operando 2:* Este bloque selecciona un segundo operador entre cualquiera de los registros internos y alguna coordenada de los múltiplos precalculados almacenados en ROM dentro de la unidad de direccionamiento indirecto. El operando seleccionado puede ser elevado al cuadrado y/o sumado con 1 de manera opcional.

bit 26: Habilita Incremento	**bit 25:** Habilita Cuadrado	**bit 24:** Pxy / Registro Selector	**bit 29-21** CCP / CR

Cuadro 4.12: Bits 26-21 del bus de Control: Aritmética de Curvas Elípticas

Los múltiplos precalculados y almacenados en memoria ROM son direccionados a partir del siguiente código numérico llamado Código de Coordenadas Precalculadas (CCP). Note que la constante "1"es direccionable como un método de conversión entre coordenadas afines y coordenadas proyectivas, simplemente asignandole a la coordenadas P_z el número 1.

- *Selector de Operando 3:* Este bloque selecciona el tercer operando para la función atómica a ser configurada en la arquitectura. El operando es seleccionado solamente entre los registros internos y es capaz de elevar al cuadrado o incrementar en uno al operando seleccionado de manera opcional.

Código de Coordenadas Pre.(CCP)	**Nombre de la Coordenada**
X00	P_x
X01	P_y
X10	1
X11	$P_x + P_y = -P_y$

Cuadro 4.13: Código de Coordenadas Precalculadas (CCP)

bit 20: Habilita Incrementot	**bit 19:** Habilita Cuadrado Selector	**bit 18-16** Código Registro

Cuadro 4.14: Bits 20-16 del bus de Control: Aritmética de Curvas Elípticas

- *Selector del Operando 4:* El cuarto operador puede ser seleccionado entre los registros internos de la arquitectura y los múltiplos almacenados en ROM. El operando puede ser incrementado en uno, elevado al cuadrado o ambos de manera opcional.

bit 15: Habilita Incremento	**bit 14:** Habilita Cuadrado	**bit 13:** Pxy/Registro Selector	**bit 12-10** CR/CCP

Cuadro 4.15: Bits 15-10 del Bus de Control: Aritmética de Curvas Elípticas

- *Multiplexor de Adición Intermedia:* La operación principal en la arquitectura es la multiplicación. Solo una multiplicación puede ser calculada por ciclo de reloj. El primer operando dentro del multiplicador se denomina *Operando1*, seleccionado por el módulo correspondiente, el segundo operando entre *Operando2* y *Operando2 + Operando4*. El multiplexor de adición intermedia configura una de estas dos opciones.

Segundo Operando del Multiplicador	**Bit 9 del Bus de Control**
Operando1 * *Operando2*	0
Operando1 * (*Operando2* + *Operando4*)	1

Cuadro 4.16: Bit 9 del bus de Control: Aritmética de Curvas Elípticas

- *Multiplexor Selector de Función:* Es posible construir una función atómica más copleja a partir de la multiplicación anterior, nombrada a partir de aqui como *MR*. Es posible sumar los operandos *Operando3* y *Operando 4* a *MR* de manera opcional. El bus de control puede configurar el multiplexor selector de función para agregar estos operandos.

Función Seleccionada	Bit 8-7 del Bus de Control
MR	00
MR + Operando3	01
MR + Operando3 + Operando4	10

Cuadro 4.17: Bits 8-7 del Bus de Control: Aritmética de Curvas Elípticas

- *Bloque de Registros 233x6 bits con Soporte para Copia Paralela:* El bloque está compuesto por un conjunto de 6 registros de 233 bits cada uno. Los registros denominados T_1, T_2 y T_3 son registros internos de propósito general para almacenar pasos intermedios en los algoritmos implementados. Los registros denominados Q_x, Q_y y Q_z son registros con un propósito más específico. En ellos se acumula el progreso del cálculo de la expresión de Horner, por lo tanto, en estos registros siempre se encuentra almacenado las coordenadas proyectivas de un punto sobre la curva y finalmente el resultado de la multiplicación escalar.

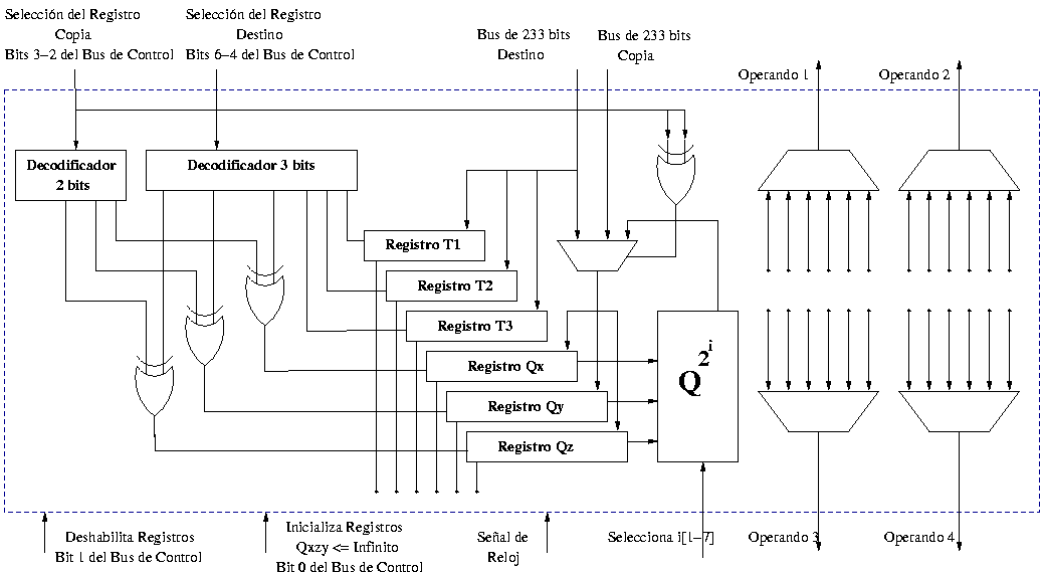


Figura 4.12: Bloque de Registros ECC

Es posible elegir un registro como destino de alguna composición aritmética e incluso permite copiar el contenido de cualquier registro hacia alguno de los registros Q. El registro fuente para la copia se se-

bit 6-4: Selecciona Registro Destino	**bit 3-2:** Selecciona Destino de Copia	**bit 1:** Deshabilita Registros	**bit 0:** Reestablece Registros

Cuadro 4.18: Bits 6-0 del Bus de Control: Aritmética de Curvas Elípticas

lecciona por medio del bloque selector del *Operando4*. Esto permite almacenar dos resultados aritméticos simultáneos sin necesidad de duplicar un bus externo de manera completa.

Finalmente dentro de este bloque se presenta una unidad capaz de elevar al cuadrado el contenido de un registro Q hasta 7 veces y guardarse en el mismo. Es decir, provee la capacidad de efectuar $Q_n \leftarrow Q_n^{2^i}$ donde $i \in [2, 7]$ en un solo ciclo de reloj. La unidad de proceso dedicada a la expansión cuenta la cantidad de ceros en una sola secuencia, al combinar dicha característica con la capacidad de calcular 7 cuadrados en un ciclo, es posible mejorar los tiempos de cómputo de manera considerable.

Las terminales del bus de control dedicadas a configurar el bloque de registros se presenta a continuación:

Los registros de propósito general T_n son reestablecidos con valor 0, los registros acumulador Q_n son reestablecidos como el punto elíptico ∞ en coordenadas proyectivas LD (1,0,0).

4.3.2. Máquina de Estados : Aritmética de Curvas Elípticas

La unidad aritmética lógica debe ser programada para realizar los Algoritmos 23, 24 y 26. El diseño tiene la capacidad de realizar todas las combinaciones aritméticas solicitadas por cada ciclo de reloj en los algoritmos anteriores. El doblado es un caso especial de la suma, así que el algoritmo de adición decide, después de un par de comprobaciones, cuando debe ser ejecutado un doblado. La máquina de estados proporcionada fusiona ambos algoritmos en el diagrama de flujo de la Figura 4.13.

Las líneas 11-15 del Algoritmo 26 en el capítulo 3 presentan el cálculo de una suma elíptica dentro del proceso de multiplicación escalar. La línea 11: $Q \leftarrow \tau Q$ eleva al cuadrado cada coordenada de Q antes de realizar un proceso de adición. Elevar al cuadrado un registro es una operación aritmética que es

fácilmente integrada en una función atómica dentro de nuestra arquitectura. Es posible integrar el cuadrado de cada coordenada en el algoritmo de adición hasta que dicho registro sea sobreescrito. Este cálculo redundante permite ahorrarnos el ciclo de reloj correspondiente a la línea 11 del algoritmo.

El diagrama de flujo presentado en la Figura 4.13 integra los algoritmos de adición y doblado y agrega la línea 11 del Algoritmo 26 como un cálculo redundante sobre los registros Q_i.

El segundo diagrama de flujo corresponde a la integración de la máquina de estados de la Figura 4.13 con el Algoritmo 26 de multiplicación escalar.

El diagrama de flujo de la Figura 4.14 salta al diagrama de flujo de la Figura 4.13 cada vez que necesita realizar una adición de puntos elípticos. El Algoritmo 26 presenta en las líneas 3-8 mapeos τ en las iteraciones impares. Esto es debido al formato de la expansión WτNAF que es leída de memoria donde cada elemento impar corresponde a un conteo de ceros dentro de una secuencia de ceros consecutivos $(w_0, z_0, w_1, z_1, \ldots, z_{i-1}, w_1)$. Este conjunto de cuadrados de campo sobre los registros $Q_i^{2^n}$ es realizable en un ciclo de reloj por la unidad aritmética lógica.

El diagrama de flujo implementado realiza la secuencia de cuadrados de campo consecutivos en bloques de 7 operaciones por ciclo de reloj, como se aprecia en la Figura 4.14.

4.4. Implementacion de Aritmética de Campos Finitos

La eficiencia de las arquitecturas aritméticas propuestas en los capítulos anteriores dependen de los algoritmos con los cuáles son programadas y, de manera aún más cercana, con la implementación de su aritmética básica. Los diseño presentados han sido optimizados para trabajar con la ley de grupo definida en el grupo de puntos sobre la curva elíptica $K-233$. Dicha curva ha sido definida sobre el campo finito binario $GF(2^{233})$ con el polinomio irreducible $z^{233} + z^{74} + 1$.

Los algoritmos paralelos propuestos en el capítulo 3 permiten analizar las operaciones aritméticas básicas utilizadas en ellos, así como su respectiva importancia. Los operaciones que conforman los algoritmos para definir la multiplicación escalar sobre grupos de curvas elípticas son la adición, multiplicación, cuadrado de campo y reducción de campo.

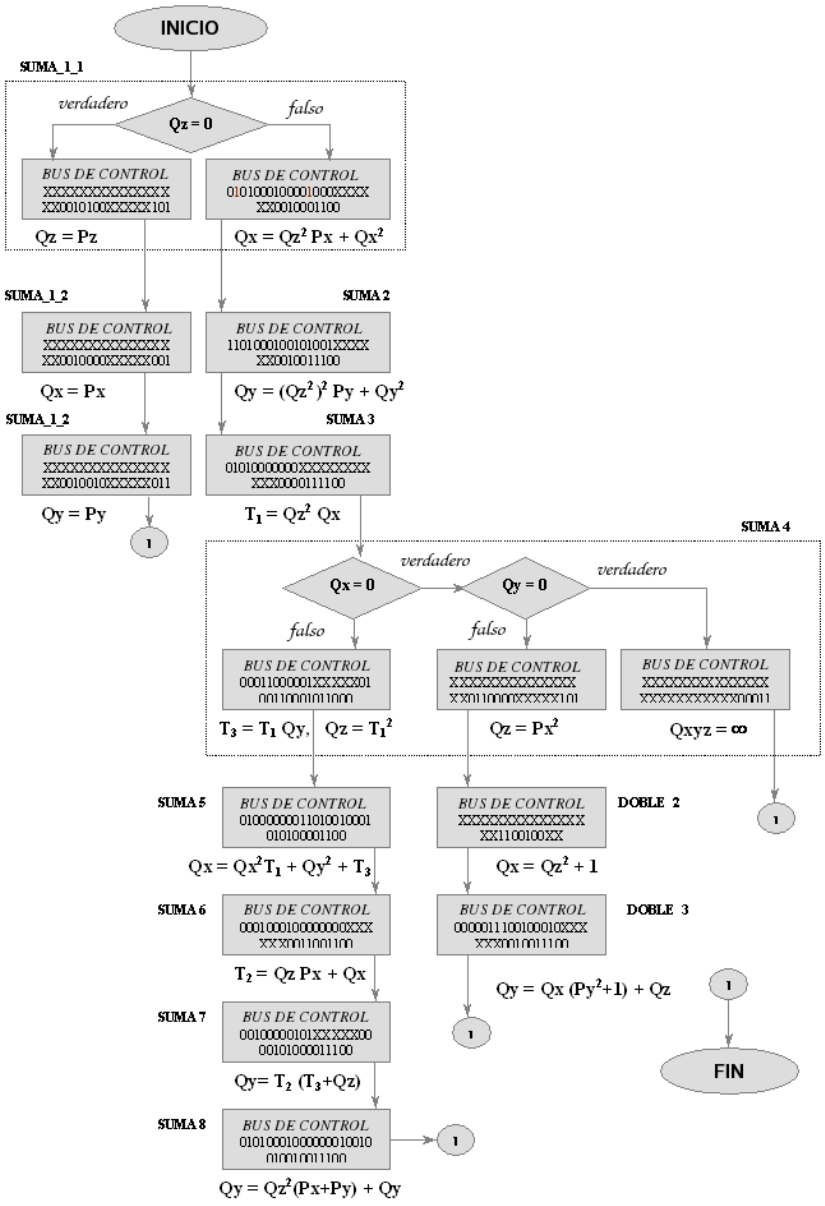


Figura 4.13: Máquina de Estados: τQ/Adición de Puntos Elípticos

La multiplicación ha sido considerada durante el proceso de diseño como una operación computacionalmente costosa. Existen una amplia variedad de algoritmos de multiplicación. Para nuestra arquitectura se opta por un

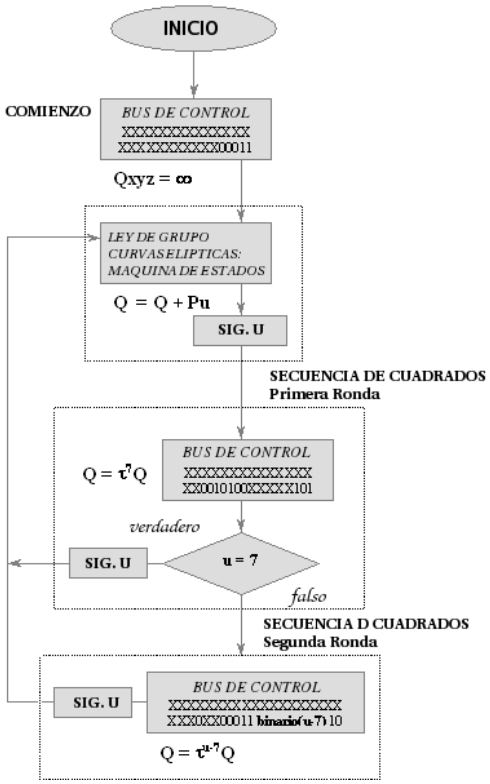


Figura 4.14: Máquina de Estados: Multiplicación Escalar Elíptica

diseño paralelo Karatsuba-Offman. Queda abierta la discusión sobre el uso de algoritmos más compactos en recursos de hardware como son los algoritmos seriales y el impacto de ellos sobre la arquitectura propuesta.

El cuadrado y reducción de campo son operaciones que requieren una eficiencia particularmente alta, debido a su integración en funciones atómicas dentro de los algoritmos paralelos propuestos. Se presenta un diseño particularmente eficiente de estas operaciones el cuál permite la integración de ellos sin afectar negativamente el periodo del ciclo de reloj en la arquitectura.

La suma sobre campos finitos binarios es considerada una operación que es simple y eficiente en plataformas de hardware. Esto se debe a que su implementación consiste en sumar bit a bit cada operando por medio de una compuerta XOR de manera paralela. Su bajo coste en recursos de hardware y en tiempo de cómputo permitió integrar la adición de campo binario en funciones atómicas de manera sencilla.

El resto del capítulo presenta el diseño de las operaciones de campo binario utilizadas en la implementación de la arquitectura propuesta en el capítulo 4.

4.4.1. Multiplicación $GF(2^{233})$

La multiplicación de A y $B \in \mathrm{GF}[2^{233}]$ puede ser realizada en dos etapas. Primero se realiza un multiplicación polinomial de A y B para finalmente realizar una operación de reducción de campo módulo el polinomio irreducible $f(z) = z^{233}+x^{74}+1$. La multiplicación polinomial puede ser diseñado a partir del algoritmo *divide y vencerás* Karatsuba-Offman

El algoritmo de Karatsuba-Offman requiere de $O(n^{log_2 3})$ operaciones de bit para multiplicar dos enteros de n bits [16]. Sean $A = x^{\frac{m}{2}}A_H + A_L$ y $B = x^{\frac{m}{2}}B_H + B_L$, entonces:

$$C = x^m A_H B_H + (A_H B_H + A_L B_L + (A_H + A_L)(B_H B_L))x^{\frac{m}{2}} + A_L B_L = x^m C_H + C_L \tag{4.2}$$

donde $C = AB$. La multiplicación Karatsuba-Offman define una multiplicación de m bits a partir de dos operandos polinomiales de $\frac{m}{2}$ bits. Esta definición permite diseñar multiplicadores grandes a partir de otros más pequeños de manera recursiva. Esta construcción recursiva puede ser representada con el diagrama presentado en la Figura 4.15.

Una propiedad muy útil e interesante de un multiplicador Karatsuba-Offman es la posibilidad de ser paralelizado de manera eficiente en plataformas de hardware. En la ecuación 4.2 se encuentran tres multiplicaciones básicas $A_H B_H$, $A_L B_L$ y $(A_H + A_L)(B_H + B_L)$ independientes entre sí.

Considerando la independencia entre las multiplicaciones internas y el bajo costo computacional de una adición de campo binario es posible paralelizar el algoritmo en una estructura de hardware presentada en la Figura 4.16.

donde $M_1 = A_H B_H$, $M_2 = B_H B_L$, $M_3 = (A_H + A_L)(B_H + B_L)$ y $M_4 = (A_H B_H)_L 2^{\frac{m}{4}} + (A_L B_L)_H$.

Una vez realizadas las tres multiplicaciones internas es posible completar una multiplicación de m bits a través de la concatenación de las adiciones presentadas en la Figura 4.16.

Este método puede ser aplicado de una manera muy simple en multiplicadores de 2^n bits. Sin embargo nuestra arquitectura necesita multiplicadores

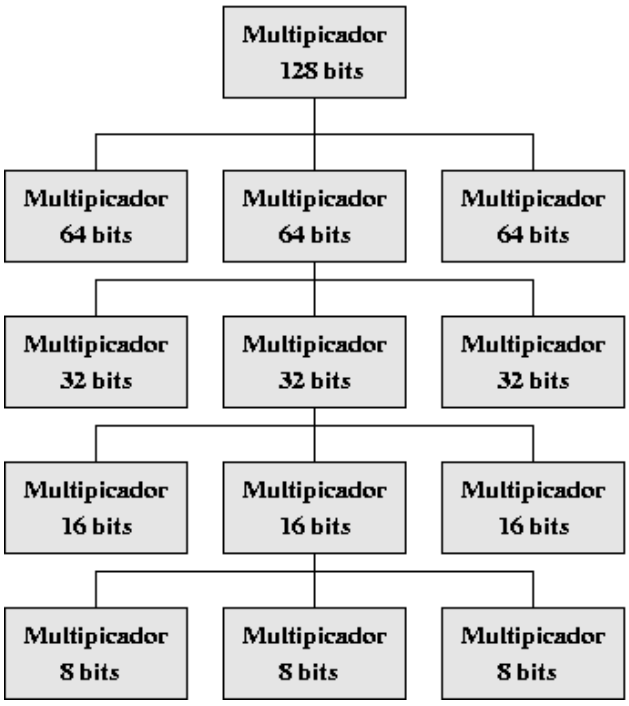


Figura 4.15: Árbol jerárquico de una estructura multiplicativa Karatsuba

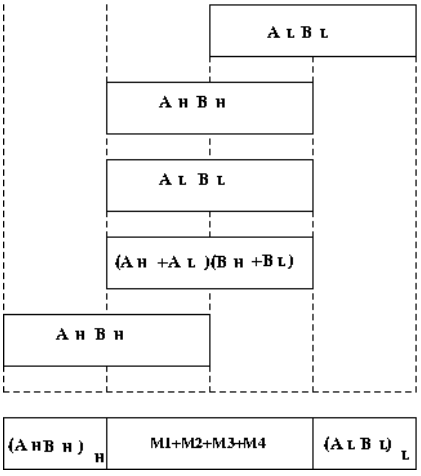


Figura 4.16: Proceso de paralelización en un multiplicador Karatsuba

de 233 bits. Una solución es el diseño de un multiplicador de 256 bits. Esta solución implica que los 23 bits más significativos nunca sean utilizados

desaprovechando recursos de hardware.

Una mejor solución es construir un multiplicador de 128 bits y completar el multiplicador a partir de una diseño Karatsuba-Offman a partir de multiplicadores de 105, 41 y 9 bits [17]. Como se muestra en el diagrama de la Figura 4.17.

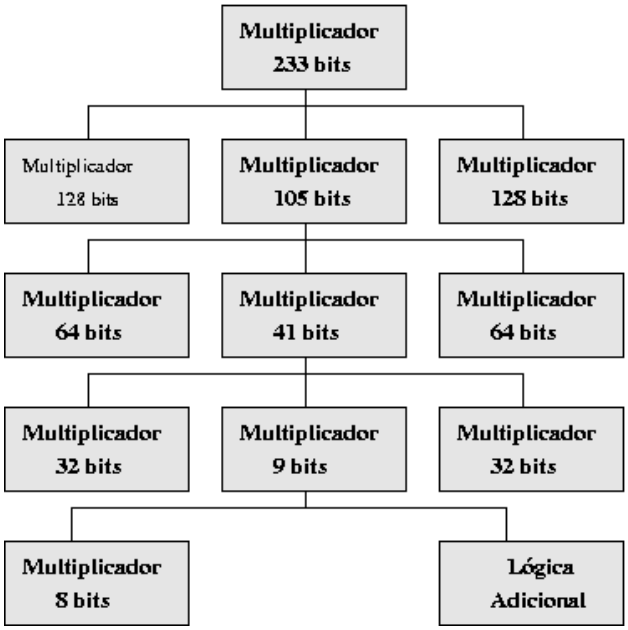


Figura 4.17: Árbol jerárquico de una estructura multiplicativa Karatsuba

El diseño propuesto e implementado para nuestra arquitectura es el presentado en la Figura 4.18. La estrategia de un diseño Karatsuba binario de 233 bits necesita cerca de 2,218 compuertas lógicas menos que un diseño Karatsuba convencional de 256 bits.

Reducción Modular : Polinomio Irreducible $f(z) = z^{233} + z^{74} + 1$

La multiplicación polinomial es una operación básica en la arquitectura, sin embargo es necesario que el resultado de la multiplicación esté dentro del conjunto definido para el campo finito $GF(2^{233})$. El proceso de llevar la multiplicación C a un elemento del campo, C mod $z^{233} + z^{74} + 1$, es llamado reducción modular.

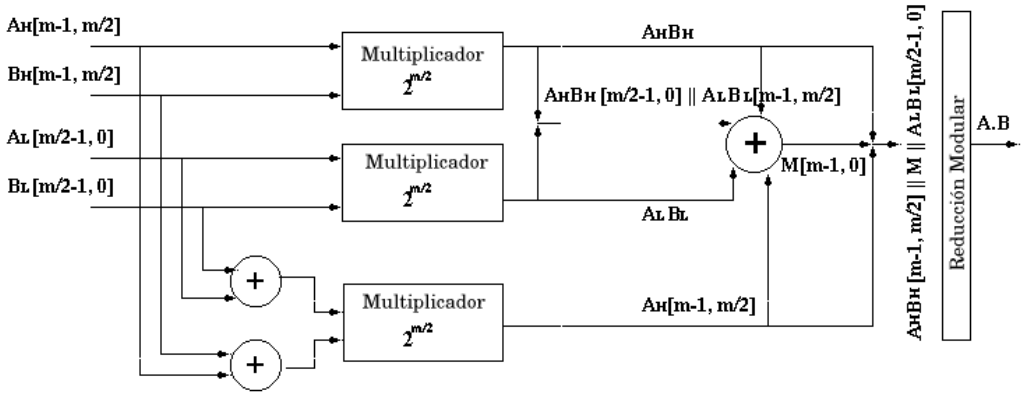


Figura 4.18: Arquitectura Karatsuba Binario

El algoritmo para la reducción modular en campos finitos binarios está basado en la observación:

$$C(z) = c_{2m-2}z^{2m-2} + \ldots + c_m z^m + c_{m-1}z^{m-1} + \ldots + c_1 z + c_0$$

$$\equiv (c_{m-2}z^{2m-2} + \ldots + c_m)r(z) + c_m z^m + c_{m-1}z^{m-1} + \ldots + c_1 z + c_0 (\mathrm{mod} f(z)). \tag{4.3}$$

La expresión 4.3 sugiere un algoritmo eficiente para la reducción modular de un número C en el campo $GF(2^m)$ si este tiene a lo mas $2m-2$ bit. El algoritmo es implementado a partir de operaciones simples, como son la XOR (adiciones $GF(2)$)y operaciones de superposición de bits (multiplicaciones por z^m).

Debido a que el polinomio irreducible en el campo finito binario $GF(2^{233})$ es un trinomio, el proceso de reducción es particularmente eficiente. La Figura 4.19 describe el proceso de reducción de un polinomio de $2m-2$ bits, para $m = 233$, con el polinomio $z^{233} + z^{74} + 1$.

Cuadrado con Reducción Modular: $z^{233} + z^{74} + 1$

El elevar al cuadrado un polinomio binario es una operación lineal. Usando una representación sobre un campo finito binario $G(2^m)$ es posible elevar al

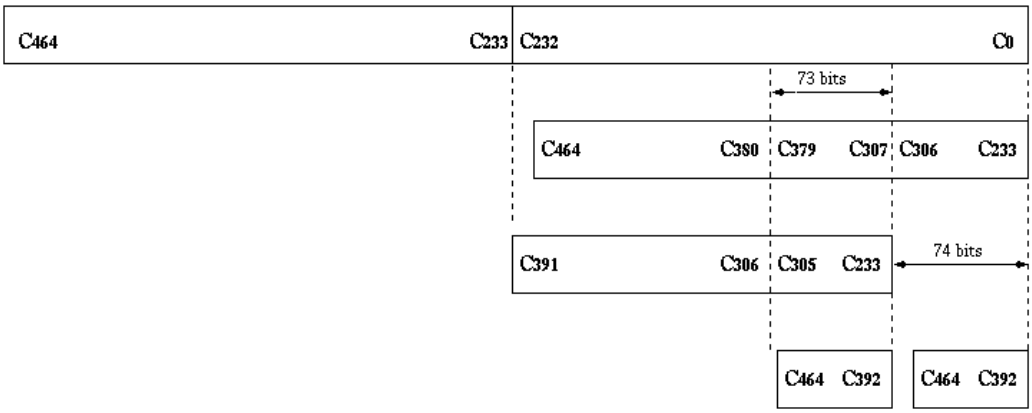


Figura 4.19: Arquitectura Reducción Modular: $z^{233} + z^{74} + 1$

cuadrado un polinomio de una manera muy simple. Sin embargo el resultado es un polinomio de $2m - 2$ bits el cuál necesita ser reducido.

La representación binaria de $A(z)^2$ se obtiene insertando un bit cero entre bits consecutivos de una representación binaria de $A(z)$.

$$\text{Si } A(z) = \sum_{i=0}^{m-1} a_i z^i$$

$$\text{Entonces } A(z)^2 = \sum_{i=0}^{m-1} a_i z^{2i}$$

Este proceso es costoso sobre un plataforma de software por la administración de arreglos de datos, sin embargo bajo un plataforma de hardware resulta ser un proceso muy eficiente, sin necesidad de operaciones aritméticas ni intercambio de registros.

La mitad de la representación binaria de $A(z)^2$ está compuesta de ceros. Esta propiedad puede ser aprovechada durante el proceso de reducción modular. El Algoritmo 4.4.1 propone un algoritmo bastante eficiente para calcular $A(z)^2$ mod $z^{233} + z^{74} + 1$ basado en el proceso de reducción modular clásico aprovechando esta característica. El desempeño obtenido supera incluso a una reducción modular genérica.

Procedimiento 27 $A(z)^2 \bmod z^{233} + z^{74} + 1$

Entrada:

$A(z) = \sum_{i=0}^{m-1} a_i z^i$

Salida:

$A(z)^2 \bmod z^{233} + z^{74} + 1$

1. Para los bits i desde 0 hasta 73
   - 1.1 Si i es impar entonces $c_i = a_{\frac{i}{2}}$ xor $a_{\frac{i+392}{2}}$
   - 1.2 Si i es par entonces $c_i = a_{\frac{i+232}{2}}$
2. Para los bits i desde 74 hasta 146
   - 2.1 Si i es par $c_i = a_{\frac{i}{2}}$ xor $a_{\frac{i+318}{2}}$
   - 2.2 Si i es impar $c_i = a_{\frac{i+233}{2}}$ xor $a_{\frac{i+159}{2}}$
3. Para los bits i desde 147 hasta 232
   - 3.1 Si i es par $c_i = a_{\frac{i}{2}}$
   - 3.2 Si i es impar $c_i = a_{\frac{i+233}{2}}$ xor $a_{\frac{i+159}{2}}$
4. Regresar $C(z) = \sum_{i=0}^{m-1} c_i z^i$.

Capítulo 5

Evaluación y Comparación de Resultados

El diseño propuesto para realizar el producto escalar de curvas elípticas sobre curvas de Koblitz ha sido codificado en VHDL [18]. El código VHDL fué sintetizado y simulado para el dispositivo FPGA de Xilinx XC2V4000.

Esto permite comprobar:

- El funcionamiento de la arquitectura de pipeline y las secuencias de lectura y escritura sobre los bloques de registro.
- El funcionamiento de las máquinas de estado que contienen la programación de los algoritmos paralelos.
- El correcto desempeño de las unidades aritméticas.
- Comprobar la cantidad de ciclos de reloj necesarios por proceso.
- Los requerimientos de hardware sobre una plataforma FPGA.

El diseño de toda la arquitectura y los algoritmos paralelos fueron optimizados para realizar aritmética de grupo sobre la curva elíptica recomendada por NIST [14] K-233 la cuál está definida sobre el campo finito binario $GF(2^{233})$ y su polinomio irreducible $\alpha^{233} + \alpha^{74} + 1$. Sin embargo, la arquitectura y sus algoritmos pueden ser adaptados para otras curvas de Koblitz o campos finitos de manera simple.

5.1. Recursos de Hardware

La mayor parte de los recursos de la arquitectura se encuentran en los operadores aritméticos. La eficiencia de la implementación de la arquitectura propuesta depende de la implementación del cuadrado de campo finito binario debido a su amplio uso e integración en operaciones atómicas. El implementar un bloque de 7 cuadrados de campo en un ciclo de reloj exige la implementación de un cuadrado de campo compacto y rápido, debido a que los tiempos de propagación se acumulan en cada operador.

Un cuadrado de campo finito para el campo $GF(2^{233})$ fué sintetizado en 88 *look-up tables* (LUTs) con un tiempo de propagación de 5.061 nanosegundos. La cantidad de recursos del FPGA utilizados permite integrar la operación en las funciones atómicas propuestas en los algoritmo paralelos. Utilizando siete bloques de cuadrado finito es posible sintetizar una pila de cuadrados para realizar la operación $\tau^i Q$ donde $i \in [2, 7]$. Dicha pila de cuadrados de campo requieren 1082 LUTs y presenta un tiempo de propagación mínimo de 11.519 nanosegundos.

Sin embargo, el operador principal dentro de la arquitectura es el multiplicador de campo finito binario. Para la paralelización de los algoritmos se consideró un solo multiplicador eficiente en tiempo de cómputo. Dicho multiplicador fué diseñado a partir de un modelo Karatsuba-Offman.

La Tabla 5.1 presenta una comparación entre multiplicadores Karatsuba-Offman implementados a partir de 4 bits hasta 128 bits. Es posible apreciar los costos en términos de compuertas lógicas y unidades básicas de FPGA en *slices*. De la misma manera, es posible apreciar el comportamiento de los tiempos de propagación obtenidos después de la sintetización de cada multiplicador para el dispositivo XC2V4000.

La Tabla 5.2 se enfoca en los recursos utilizados por los multiplicadores Karatsuba binarios utilizados desde 9 bits hasta 233 bits. Este último multiplicador es el utilizado en la arquitectura como multiplicador principal.

Los resultados presentados en las tablas se refieren a multiplicadores Karatsuba polinomiales.

El proceso de reducción implementado, considerado un bloque independiente, se basa en el algoritmo presentado en el capítulo anterior. La implementación sobre el FPGA XC2V4000 resulta en un operador eficiente en recursos de espacio y tiempo de cómputo. Una reducción de campo está sintetizada en 134 *slices* y tiene un tiempo de propagación de 5.061 nanosegundos. Es importante notar que es posible elevar al cuadrado con menos recursos de

Multiplicador # de bits	**Costo en Multiplicadores**	**Costo en Compuertas Lógicas**	**Costo en Slices**	**Tiempos de Propagación**
4 bits	-	9 XOR 16 AND	6	5.799 ns
8 bits	3 M_4+ 32 XOR	59 XOR 48 AND	26	6.721 ns
16 bits	3 M_8+ 64 XOR	241 XOR 144 AND	95	9.087 ns
32 bits	3M_{16}+ 128 XOR	851 XOR 432 AND	323	10.078 ns
64 bits	3M_{32}+ 256 XOR	2,809 XOR 1,296 AND	1,047	11.658 ns
128 bits	3M_{64}+ 512 XOR	8,939 XOR 3,888 AND	3,298	14.150 ns

Cuadro 5.1: Recursos utilizados en multiplicadores Karatsuba Clásicos

hardware que el necesario para una reducción.

La Tabla 5.3 muestra la cantidad de recursos utilizados en el FPGA XC2V4000 en el proceso de sintetización del código VHDL.

Es importante notar que el 42.33 % del espacio utilizado por el diseño total de multiplicación escalar es utilizado por el multiplicador de campo binario Karatsuba sobre $G(2^{233})$.

5.2. Tiempo de Cómputo

El tiempo en que es posible calcular una multiplicación por un escalar del punto generador de $K-233$ está en función de la cantidad de ciclos de reloj necesarios para ejecutar los algoritmos paralelos propuestos y del periodo de reloj mínimo en el cual la arquitectura sintetizada presenta un comportamiento estable. Dicho periodo de reloj está establecido por los tiempo de propagación de las señales internas.

La cantidad de ciclos de reloj necesarios para ejecutar los algoritmos pueden ser calculados a partir de las características principales de la arquitectura y de una expansión WτNAF. Los Algoritmos 15 y 16 propuestos en [2] y [3] respectivamente, tienen desempeños distintos. La diferencia en

Multiplicador # de bits	**Costo en Multiplicadores**	**Costo en Compuertas Lógicas**	**Costo en Slices**	**Tiempo de de Propagación**
9 bits	1 M_8+ 16 XOR + 2 AND	75 XOR 72 AND	35	8.199 ns
41 bits	2 M_{32}+ 1 M_9+ 32 XOR	1,812 XOR 936 AND	587	11.139 ns
105 bits	2 M_{64}+1M_{41}+ 64 XOR	7,640 XOR 3,528 AND	2,678	13.461 ns
233 bits	$2M_{128} + 1M_{105}$+ 128 XOR	25,984 XOR 11,304 AND	9,459	14.497 ns

Cuadro 5.2: Recursos utilizados en multiplicadores Karatsuba Binarios

los tiempos de cómputo entre ambos algoritmos se debe a:

1. El Algoritmo 15 necesita de 4 ciclos de reloj para calcular un elemento distinto de cero en la expansión, mientras que el Algoritmo 16 requiere de 6 ciclos de reloj para la misma actividad.

2. Los longitudes promedio y máximas de las expansiones generadas por dichos algoritmos son distintas, de acuerdo al Apéndice B. El Algoritmo 15 genera expansiones con una longitud promedio de 244 y el Algoritmo 16 genera expansiones con una longitud promedio de 230.5. Ambas expansiones para una aritmética $GF(2^{233})$ con ventana de precomputo $\omega = 8$.

Supongamos que la arquitectura ha sido implementada sobre el campo finito $GF(2^m)$. Por lo tanto tendremos una expansión WτNAF de longitud muy cercana a m.

Sea ω la longitud de la ventana de precómputo y ξ la longitud promedio de la expansión . Por lo tanto se espera encontrar dentro de la expansión $\xi\frac{1}{\omega-1}$ elementos distintos de cero y $\xi\frac{\omega}{\omega-1}$ elementos cero. Para generar dicha expansión, se necesitan 21 ciclos de reloj para realizar el Algoritmo 17 de reducción parcial.

El Algoritmo 19 ejecuta un ciclo de reloj por cada elemento cero generado y un elemento distinto de cero necesita haber ejecutado 4 ciclos de reloj de manera previa. Por lo tanto el proceso de generación de una expansión

Nombre del Módulo	**Número de Slices**	**Número de LUTs**	**BRAM**	Mult. **18x18**	**Periodo de Reloj Mínimo**
Cuadrado de campo $GF[2^{233}]$	88	153	—	—	—
Arreglo de 7 Cuadrados de Campo	1082	2003	—	—	—
Multiplicador Karatsuba $GF[2^{233}]$	9,588	16,674	—	—	—
Multiplicador $\mathbb{Z}$ 128 bits	0	0	—	48	—
Buffer de Memoria Compartida	611	1,182	9	—	—
Unidad de Proceso de Curvas Elípticas	15,945	29,078	—	—	19.333ns
Unidad de Proceso de Expansión	5,478	9,541	2	48	37.119ns
Multiplicador Escalar Elíptico	22,265	39,762	11	48	36.253ns

Cuadro 5.3: Recursos utilizados por los módulos principales de la arquitectura.

WτNAF sobre un campo finito $G(2^{233})$ necesita de :

$$\#\text{Ciclos de Reloj} = 4\frac{\xi}{\omega+1} + 1\frac{\xi\omega}{\omega+1} + 17 = \frac{\xi(4+\omega)}{\omega+1} + 21 \tag{5.1}$$

El Algoritmo 20 ejecuta un ciclo de reloj por cada elemento cero generado y un elemento distinto de cero necesita haber ejecutado 6 ciclos de reloj de manera previa. Por lo tanto el proceso de generación de una expansión WτNAF sobre un campo finito $G(2^{233})$ necesita de :

$$\#\text{Ciclos de Reloj} = 6\frac{\xi}{\omega+1} + 1\frac{\xi\omega}{\omega+1} + 17 = \frac{\xi(6+\omega)}{\omega+1} + 21 \tag{5.2}$$

De una manera similar es posible conocer el número de ciclos esperados al realizar una multiplicación escalar dada la expansión WτNAF. Esta vez conocemos que el Algoritmo 26 requiere realizar una suma elíptica-mapeo τ

con cada elemento distinto de cero y cada elemento cero implica un mapeo τ.

El Algoritmo 23 requiere de ocho ciclos de reloj para realizar una suma elíptica y la microprogramación de la arquitectura puede incluir el mapeo τ en dicho algoritmo como se aprecia en la Figura 4.13 de su correpondiente diagrama de flujo.

También se sabe que en una expansión WτNAF con una ventana de precómputo de ω bits, un coeficiente distinto de cero en la expansión implica que, por lo menos, los siguientes $\omega - 1$ coeficiente serán ceros. Nuestra arquitectura contiene tres bloques que comprenden secuencias de hasta 7 cuadrados de campo en secuencia (se necesitan tres, puesto que es necesario aplicar el mapeo de Frobenius a cada coordenada de la representación proyectiva del punto). Por lo tanto es posible considerar que cada coeficiente cero en la expansión requiere de $\frac{1}{7}$ de ciclo de reloj.

Por lo tanto, el proceso de evaluación de la multiplicación escalar dada la expansión WτNAF requiere:

$$\#\text{Ciclos de Reloj} = 8\frac{\xi}{\omega+1} + \frac{1}{7}\frac{\xi\omega}{\omega+1} = \frac{\xi(w+56)}{7w+7}. \tag{5.3}$$

Nuestra arquitectura fué codificada en VHDL y sintetizada para una ventana de precómputo de ω=8 bits y para el campo finito $GF(2^{233})$. Por lo tanto, la generación de una expansión WτNAF utilizando el Algoritmo 19 es calculada en 346.33 ciclos de reloj mientras que el Algoritmo 20 necesita de 379.55 ciclos de reloj . Una evaluación de la multiplicación escalar requiere de 247.87 ciclos de reloj para el Algoritmo 19 y 234.15 para el Algoritmo 20 utilizando sus respectivas longitudes de expansión promedio. La Tabla 5.4 compara los dos algoritmos implementados.

Algoritmo	**Longitud Promedio de la Expansión**	**# Ciclos para generar Expansión**	**# Ciclos para calcular Horner**	**Tiempo Promedio de Cómputo**
19	244	346.33	247.87	17.64μs
20	230.5	379.55	234.15	18.61μs

Cuadro 5.4: Comparación de desempeño entre los Algoritmos 19 y 20.

En la Tabla 5.3 se observa una diferencia considerable entre los periodos de reloj de las dos principales unidades de proceso. Ambas unidades de

proceso son independientes entre sí y es posible que trabajen con relojes independientes. Sin embargo la herramienta proporcionada por Xilinx para síntesis, aún cuando proporciona divisores de reloj, no obtiene predicciones de síntesis coherentes con división de reloj.

Se espera que en una implementación sobre un dispositivo físico pueda efectuarse dicha división obteniendo un ciclo de reloj independiente a cada unidad de proceso. Respetando los periodos de reloj de manera independiente a cada unidad de proceso es posible obtener una multiplicación escalar en: 14.08μ segundos + 4.53 μ segundos = 18.61 μ segundos.

Si se utiliza un reloj global para la arquitectura completa, el periodo de reloj mínimo es 36.253 $n$segundos. En este caso es posible obtener una multiplicación escalar en 22.25 μ segundos.

La gráfica ilustrada en la Figura 5.1 fué obtenida en el simulador de Xilinx ModelSim XE II 5.8c utilizando un reloj principal con un periodo de 39 ns y un divisor de frecuenciade reloj, de esta forma la unidad de curvas elípticas trabaja con un ciclo de reloj de 19.5 ns.

El tiempo de cómputo de una multiplicación escalar en la simulación es de 20.22 μ segundos.

5.3. Comparación y Evaluaciones Finales

A lo largo de los últimos años y a partir de la estandarización de algoritmos criptográficos basados en el problema del logaritmo discreto en curvas elípticas se han presentado diversos diseños e implementaciones de los algoritmos para criptografía de curvas elípticas en plataformas diversas.

En la presente sección se presenta una comparación del trabajo presentado en esta tesis con otros trabajos publicados previamente. Las comparaciones se enfocan en tres trabajos previos publicados entre los años 2000 y 2004. Estos trabajos son de especial interés por compartir la misma plataforma de diseño, FPGAs, o el desarrollo de criptografía de curvas elípticas sobre la familia de Koblitz.

El primer trabajo con el cual nuestro trabajo es comparado fué publicado en el año 2000 por Gerardo Orlando y Christof Paar con el título *A High Performance Reconfigurable Elliptic Curve Processor for GF(2m)*"[19]. Las características principales de su trabajo son:

- El diseño presentado y los resultados finales son enfocados en un procesador genérico para curvas elípticas. Es decir, presenta la capacidad de

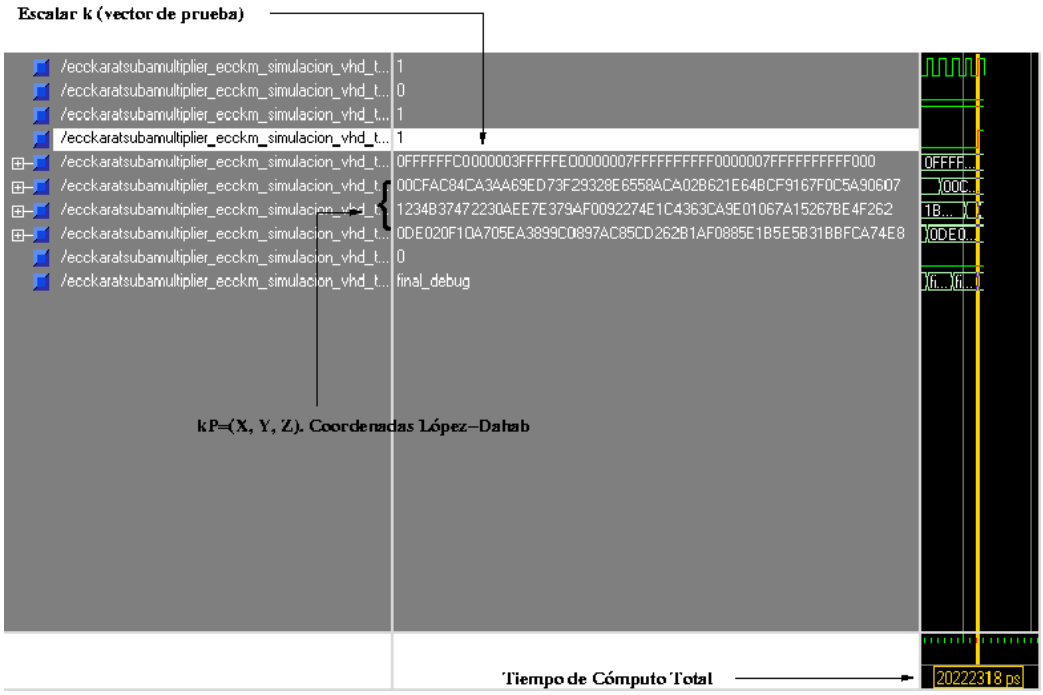


Figura 5.1: Resultados de tiempo de cómputo sobre un simulación realizada en la herramienta Xilinx ModelSim XE II 5.8c

calcular aritmética sobre la ley de grupo sobre curvas elípticas y permite ser programado para generar algoritmos más complejos como lo es la multiplicación escalar elíptica.

- Su arquitectura no presenta ninguna optimización para algún algoritmo de multiplicación escalar en particular.
- La ley de grupo de curvas elípticas es implementada utilizando una representación proyectiva de las coordenadas de los puntos elípticos.
- El multiplicador de campo finito sobre el cuál se encuentra implementada la ley de grupo de curvas elípticas es serial.
- Su unidad aritmética está diseñada sobre un modelo *pipeline*.
- El campo finito base implementado es $GF(2^{167})$.

El diseño se presenta como una unidad de proceso no especializada y las mediciones de desempeño son presentadas a partir de la programación de los algoritmos de multiplicación escalar de Montgomery y el algoritmo básico de multiplicación binario de suma y doblado. Se considera al trabajo de Orlando y Paar como importante pues presenta características similares como los son la plataforma de desarrollo sobre FPGA y el diseño *pipeline*, así como contrastes importantes. Los contrastes principales son el uso de un multiplicador serial de 16 bits, el diseño sin especialización y el campo finito base $GF(2^{167})$.

El siguiente trabajo considerado relevante para la comparación de resultados es el trabajo publicado por Nazar A. Saqib, Francisco Rodríguez y Arturo Díaz titulado *A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication over GF(2m)* [20] en el año 2004. Saqib presenta una arquitectura implementada sobre hardware reconfigurable con las siguientes características:

- El diseño está dedicado al cálculo de la multiplicación escalar elíptica.
- El diseño se presenta optimizado para el algoritmo de Motgomery de multiplicación escalar.
- Las coordenadas de puntos elípticos son representadas en coordenadas proyectivas.
- La multiplicación de campo finito se realiza a partir de dos multiplicadores Karatsuba.
- El campo finito base en la implementación de la aritmética de curvas elípticas es el campo finito binario $GF(2^{191})$.

El trabajo es considerado importante pues presenta una arquitectura dedicada y optimizada en un algoritmo particular de curvas elípticas. Además presenta el uso de un diseño paralelo de multiplicadores de campo finito binario Karatsuba similar al presentado en esta tésis. El uso de coordenadas proyectivas presenta otro punto de comparación con el trabajo presentado en este documento.

El diseño publicado por Jonathan Lutz titulado *High Performance Elliptic Curve Cryptographic Co-Processor* [21] en el año 2004 se presenta como otro antecedente al trabajo presentado en esta tésis. El trabajo presentado

por Lutz es de particular interés pues su diseño está dedicado a multiplicación escalar en curvas de Koblitz utulizando un expansión τNAF sobre una plataforma de hardware reconfigurable. Las principales características de su propuesta son:

- Diseño dedicado y optimizado para curvas de Koblitz utilizando el algoritmo de expansión τNAF.
- Representación de puntos elípticos utilizando coordenadas proyectivas López-Dahab.
- La multiplicación de campo binario es calculada utilizando un multiplicador serial de 4 a 41 bits.
- La implementación se presenta sobre el campo finito binario $GF(2^{163})$

Referencia	Plataforma	Campo	kP	Area	Velocidad/Area
[19]	XCV400E	$G2^{167})$	$210\mu s$	3,002 LUTs	1.586
[21]	XCV2000E	$GF(2^{191})$	$75\mu s$	10,017 LUTs	1.331
[20]	XCV3200E	$GF(2^{163})$	$76\mu s$	10,000 LUTs	1.315
Nuestra propuesta	XC2V4000	$GF(2^{233})$	$17{,}64\mu s$	39,762 LUTs	1.610

Cuadro 5.5: Tabla Comparativa de Desempeños

El hecho de estar dedicado a curvas de Koblitz utilizando un algoritmo de expansión τNAF presenta al diseño de Lutz como un referencia óptima para los resultados obtenidos en nuestra propuesta.

La Tabla 5.5 presenta una comparación de los resultados obtenidos en cada trabajo publicado en contraste con la propuesta presentada en esta tésis.

Se presentan comparaciones entre plataforma de desarrollo, campo finito binario base, tiempos estimados de cálculo de una multiplicación escalar, recursos de hardware utilizados y relación velocidad/área.

Es importante observar como, a pesar de que nuestra propuesta fué implementada sobre un campo finito báse de 233 bits, presenta el mejor tiempo de cómputo para una multiplicación escalar. Sin embargo la cantidad de

unidades básicas de FPGA utilizadas en considerablemente mayor, como consecuencia de el campo finito utilizado y del uso de un multiplicador paralelo Karatsuba.

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1. Conclusiones

En el presente trabajo de tesis proponemos un diseño para el cálculo de multiplicación escalar en curvas de Koblitz. Este diseño presenta características notables que han permitido obtener una multiplicación escalar en un tiempo particularmente pequeño, Entre dichas características podemos anotar:

1. Dos unidades de proceso independientes especializadas en aritméticas distintas, $\mathbb{Z}$ y $GF(2^{233})$.

2. Un flujo de datos en las unidades aritméticas basado en un esquema de *pipeline* de dos etapas.

3. Paralelización a nivel de composición de funciones en unidades atómicas.

4. Diseños paralelos para unidades aritméticas básicas de campos finitos binarios de alto desempeño.

5. Diseño especializado para el cálculo de una multiplicación escalar elíptica aplicable en generación de firmas digitales.

6. Optimización y especialización en el algoritmo WτNAF para curvas de Koblitz.

El diseño siempre estuvo dirigido a obtener el menor tiempo de cómputo posible aplicando técnicas de paralelización que podían ser costosas en términos de espacio y recursos de hardware. Un claro ejemplo de esto es la decisión de utilizar un diseño paralelizado de un multiplicador $G(2^{233})$ Karatsuba-Offman. Dicho multiplicador ofrece un diseño estructurado y ofrece tiempos de cómputo atractivos, sin embargo el uso de aritmética de campos finitos de 233 bits repercute en la cantidad de hardware utilizado para su implementación. Dicho multiplicador ocupa el 42.33 % de los recursos de hardware del diseño total.

Aún cuando la elección de una aritmética de campos finitos de 233 bits afecta el espacio necesario para la implementación existe una característica particular del campo $GF(2^{233})$ que beneficia el diseño. El campo finito binario $G(2^{233})$ tiene como polinomio generador el trinomio $\alpha^{233} + \alpha^{74} + 1$. El hecho de que el polinomio generador sea un trinomio beneficia de manera directa a el proceso de reducción y por consiguiente al proceso de cuadrado de campo con reducción. Esta característica en particular permite calcular una cantidad relativamente grande de cuadrados de campo en un mismo ciclo de reloj considerablemente pequeño y en poco espacio de hardware.

Estas dos últimas premisas nos permiten observar que la elección de un campo finito base repercutirá en la velocidad de cómputo y en el espacio requerido. Las características de la implementación de su aritmética tienen que ser tomadas en cuenta para el diseño global de la arquitectura.

También es importante observar que las adaptaciones de los algoritmos de multiplicación escalar de Koblitz en un ambiente de paralelismo por funciones atómicas dependerán fuertemente de la eficiencia y características esperadas de cada operador sobre algún campo finito en específico.

Los algoritmos propuestos por Solinas para curvas de Koblitz poseen un gran potencial en implementaciones sobre arquitecturas de hardware donde la paralelización puede ser aplicada en sus formas más diversas. En hardware es posible obtener implementaciones de aritmética de campos finitos binarios increíblemente eficientes. Dichos diseños pueden ser aplicados sobre los algoritmos para multiplicación escalar en curvas de Koblitz y con ellos mejorar significativamente los tiempos de cómputo. Un ejemplo de la aplicación de dichos operadores es el uso dado al proceso de elevar al cuadrado sobre el campo finito binario $G(2^{233})$.

La eficiencia de este operador y la característica principal de una expansión τNAF son poderosas aliadas. El tener secuencias de $\frac{1}{\omega-1}$ ceros en una expansión con ventana de precómputo de ω bits y su correspondencia con el

mapeo de Frobenius τ aplicada a un punto elíptico P, permiten el cálculo de hasta 7 elevaciones al cuadrado de manera atómica en un solo ciclo de reloj. El uso dado ha este operador permite una aceleración del algoritmo en cerca del 50 %.

Sin embargo, los algoritmos de multiplicación escalar sobre curvas de Koblitz no han sido aplicadas sobre plataformas de hardware ampliamente. La implementación de Lutz [21] es la principal de ellas obteniendo excelentes tiempos de cómputo.

La arquitectura fué diseñada con dos unidades de proceso independientes que separan la aritmética sobre $\mathbb{Z}$ y la aritmética de campos finitos $GF(2^{233})$. La eficiencia de sus implementaciones presentan grandes diferencias. Una implementación clásica sobre una plataforma de software para microprocesadores comerciales ofrece unidades aritméticas especializadas en aritmética entera y de punto flotante, siendo necesario implementar aritmética de campos finitos con aritmética entera como base. La consecuencia inmediata es un desempeño notablemente mejor para cálculos de aritmética entera y de punto flotante. Sin embargo una plataforma de hardware permite el diseño especializado de operaciones aritméticas. La aritmética de campos finitos binarios presenta características que permiten su implementación en plataformas de hardware con un muy alto desempeño.

Las observaciones anteriores pueden apreciarse en la Tabla 5.3 donde la unidad aritmética de expansión tiene un periodo mínimo de reloj de 36.853ns y la unidad aritmética de curvas elípticas presenta un periodo mínimo de reloj de 19.281ns. La unidad aritmética que implementa campos finitos es notablemente más rápida que la unidad de aritmética entera.

Dicho contraste entre las velocidades de las unidades aritméticas y su independencia funcional sugieren el uso de relojes independientes para cada unidad de proceso en una implementación física. De esta manera se presentan dos tiempos de cómputo para la multiplicación escalar con el diseño propuesto. Utilizando una señal de reloj es posible realizar el cálculo en 22.25 μsegundos e idealmente con dos señales de reloj independientes para cada unidad de proceso es posible calcular la multiplicación escalar en 18.61 segundos.

6.2. Trabajo a Futuro

Es preciso notar que los resultados obtenidos, tanto en tiempo de cómputo como en área utilizada, dependen de manera directa con varios parámetros de diseño. Es necesario observar el comportamiento de la arquitectura propuesta con diversos cambios en ella.

La aplicación de los cuadrados de campo fue realizable debido al excelente rendimiento obtenido por su implementación en campo finito binario base con un trinomio irreducible. Es necesario adaptar la arquitectura para un campo finito binario con un pentanomio irreducible y comparar el rendimiento obtenido con este diseño.

El uso de un multiplicador de campo finito binario diseñado para una implementación paralela del multiplicador Karatsuba-Offman consume el 40 % de los recursos de hardware demandados por la arquitectura. Es necesario adaptar diferentes multiplicadores de campo y observar la relación velocidad/espacio obtenida en la práctica. Nam Su Chang propone un diseño para un multiplicador Karatsuba [22] el cuál disminuye la cantidad de hardware utilizado a partir de eliminar cómputo redundante. Sería de interés aplicar este tipo de multiplicadores paralelos de alto desempeño en la arquitectura propuesta. Tambien es importante observar que en la arquitectura propuesta por Lutz [21] se utilizan multiplicadores de campo seriales.

Finalmente es necesario experimentar el comportamiento del uso de dos relojes independientes para cada unidad de proceso utilizando un divisor de reloj. El nuevo desempeño debe ser analizado sobre un FPGA debido a que el sintetizador proporcionado por Xilinx no considera esta división de reloj al momento de presentar sus resultados.

Apéndice A

Tecnologías de Hardware Reconfigurable

El principal objetivo del diseño de implementaciones de aplicaciones criptográficas es el ofrecer un compromiso entre capacidad de cómputo y recursos computacionales. La selección de una plataforma de desarrollo requiere el conocimiento de las necesidades de la aplicación y el entorno donde se necesitan de servicios criptográficos.

El uso software en procesadores de propósito general se presenta como una selección apropiada para aplicaciones donde el flujo de información que será procesada no requiere de una tratamiento en tiempo real. Aplicaciones sobre redes de baja y media velocidad pueden hacer uso de dichos procesadores. Es común la implementación de bibliotecas criptográficas para familias de procesadores, como lo son la familia Pentium. Dichas bibliotecas pueden ser optimizadas para el uso de los recusos específicos de cada procesador o familia de procesadores. Ejemplo de dichas características son los registros dedicados a operaciones de multimedia o coprocesadores de punto flotante. Las soluciones presentadas por Certicom [23] en software para protección de e-mail (movianMail) y almacenamiento de información en dispositivos móviles (movianCrypt) son ejemplos del uso efectivo de dichas plataformas. El desempeño obtenido de soluciones implementadas sobre plataformas de software dependen del correcto uso de las características específicas que pudieran encontrarse en algún procesador de propósito general, del desempeño del propio procesador y de la habilidad del programador. La principal ventaja de las plataformas de software es la flexibilidad obtenida en etapas de desarrollo y de mantenimiento. La gran cantidad de paradigmas

de diseño de software, como la programación orientada a objetos, permiten una buena administración de un proyecto. Las etapas de mantenimiento son relativamente sencillas, dado que, previo un buen diseño de la aplicación, es posible modificar el código del proyecto. La reutilización de software vía el uso de bibliotecas permite el rápido desarrollo de una aplicación.

Sin embargo, existen aplicaciones donde la demanda de poder de cómputo excede las posibilidades de un procesador de propósito general. Normalmente dicho procesador no se encuentra dedicado a la aplicación criptográfica en exclusiva. Dicho procesador está diseñado para atender diversos procesos a la vez, sin ser particularmente eficiente en ninguno de ellos. Aplicaciones como transmisión de voz, datos y video en tiempo real necesitan de un desempeño mínimo en sus procesos de seguridad para garantizar un límite mínimo de calidad en la transmisión. Soluciones implementadas sobre software especializado no son capaces de garantizar dicha calidad en la mayoría de casos. La solución es escoger soluciones desarrolladas sobre plataformas de hardware especializado. Sin embargo el ciclo de diseño y desarrollo en tecnologías VLSI (*Very-large-scale integration*) implica altos costos. El mantenimiento posterior de los sistemas (como la actualización o corrección de los algoritmos implementados) no resulta viable por la dificutad de obtener diseños modulares y de bajo costo.

Una solución que provee la alta especialización de los diseños VLSI con la flexibilidad en el diseño y el mantenimiento posterior del software la presenta el hardware reconfigurable. Dispositivos de hardware reconfigurable como los FPGA (*Field Programmable Gate Arrays*) ofrecen la posibilidad de un ciclo de desarrollo veloz sobre diseños de hardware especializados con un alto grado de flexibilidad.

Entre las ventajas que encontramos en los dispositivos FPGA de última generación podemos listar [24]:

- Un ciclo de diseño y desarollo pequeño para obtener prototipos funcionales.
- Las herramientas de diseño disponibles para FPGAs son más baratas que sus equivalentes ASIC.
- Potencial inherente al FPGA para reprogramación rápida, sencilla y con bajo costo, así como la posibilidad de realizar pruebas con diseños experimentales.

- Las simulaciones y pruebas de desarrollos sobre FPGAs son realizables sobre simulaciones previas a la implementación y sobre implementaciones sobre el dispositivo físico.

Dispositivos FPGA

Los dispositivos FPGA son dispositivos capaces de ser configurados en tiempo de ejecución. Es decir, tienen la capacidad de cambiar su configuración según la aplicación lo demande en un momento determinado del proceso en ejecución.

Los dispositivos FPGA son circuitos integrados prefabricados como arreglos de bloques lógicos idénticos con recursos para ruteo en interconección [25] La configuración o programación se realiza en un tiempo de orden de minutos. La capacidad de ser reprogramados cualquier número de veces reduce el costo de realizar cambios en el diseño, dado que el mismo dispositivo puede seguir siendo utilizado.

Los miles de elementos básicos que componen un FPGA son conocidos como CLBs (*Configuration Logic Block*) conectados entre sí a través de interconecciones programables. Los CLBs pueden ser configurados para realizar una función específica básica y conectarse entre ellos de manera dinámica para constituir un sistema funcional.

Algunos dispositivos FPGA contienen elementos más complejos y especializados en el circuito integrado capaces de interconectarse con otros elementos básicos funcionales de manera dinámica.

La familia de dispositivos Virtex-II de Xilinx está constituido por bloques de entrada/salida IOBs (*input output blocks*) y bloques internos configurables CLBs. IOB programables proveen la interfaz entre las terminales externas del dispositivo y la configuración lógica interna. La lógica configurable interna en esta familia de dispositivos contiene 4 tipos principales de dispositivos organizados en un arreglo regular como se muestra en la Figura A.1.

- CLBs, los cuáles proveen de elementos funcionales para lógica combinacional y síncrona, incluyendo elementos básicos de almacenamiento. Cada CLB tiene asociado un buffer de 3 estados BUFT para manejo de ruteo.

- Bloques de memoria RAM, los cuales proveen módulos de almacenamiento de 18k bits de RAM de puerto dual.

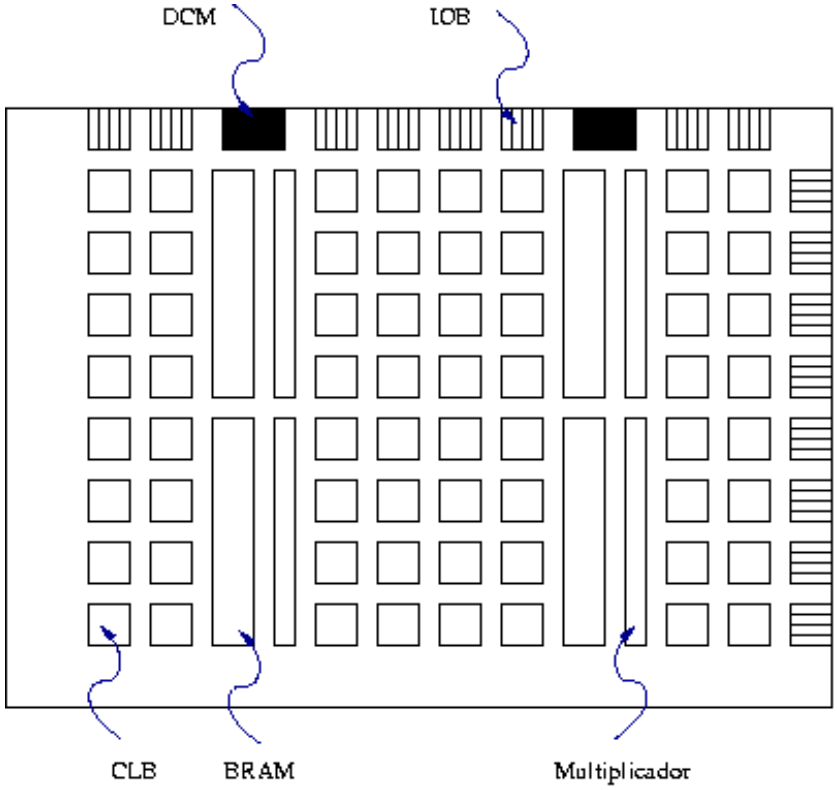


Figura A.1: Arquitectura Virtex II

- Bloques de multiplicadores dedicados de 18x18 bits.
- Administradores de Reloj Digital, DCMs (*Digital Clock Manager*), los cuales proveen soluciones de autocalibración para compensación de retraso por distribución de reloj, multiplicación y división de reloj y corrimiento de fase de reloj en grano fino y grueso.

Existen diversos fabricantes dedicados a la elaboración de dispositivos FPGA. Cada fabricante ofrece diversas familias, cada una de ellas con diferentes funcionalidades y dispositivos especializados. Como ejemplo de estos dispositivos se pueden encontrar multiplicadores dedicados, memorias RAM reconfigurables e incluso microprocesadores integrados en el FPGA. Entre las familias más renombradas y sus respectivos fabricantes encontramos: Virtex, Spartan (Xilinx), FLEX, APLEX (Altera), ACT (Actel), pASIC (Quick Logic), LCA (Logic Cell Array) y ORCA (Lucent) como se muestra en la Tabla

Fabricante	Familias FPGA
Xilinx	Virtex, Spartan
Altera	FLEX, APLEX
Actel	ACT
Quick Logic	pASIC
Logic Cell Array	LCA
Lucent	ORCA

Cuadro A.1: Fabricantes y respectivas familias de dispositivos FPGA

A.1.

Arquitectura Virtex-II

La familia Virtex-II de Xilinx presenta un conjunto de elementos básicos especializados entre los que podemos encontrar multiplicadores de 18x18 bits, memorias BRAM y buffers de tres estados. La familia cuenta con bloques CLB e IOB para la configuración de los circuitos. En nuestro trabajo, esta familia de hardware reconfigurable presenta elementos especializados que pueden ser aplicados de manera directa. Se seleccionó el dispositivo XC2V4000 para ser la plataforma de síntesis de nuestro diseño. A continuación se presenta un análisis de la arquitectura interna presente en la familia Virtex-II de Xilinx.

Bloques de Entrada Salida

Los bloques de entrada salida, IOBs, son completamente programables y se pueden clasificar como [26]:

- Bloques de entrada con un registro opcional SDR (*single data rate*) o DDR(*double data rate*).
- Bloques de salida con un registro opcional SDR o DDR y un buffer de 3 estados opcional.
- Bloque bidireccional.

Los registros son flip-flops tipo D o latches como aparece en la Figura A.2. Necesitan de un conjunto de dos relojes desfasados 180 grados entre ellos.

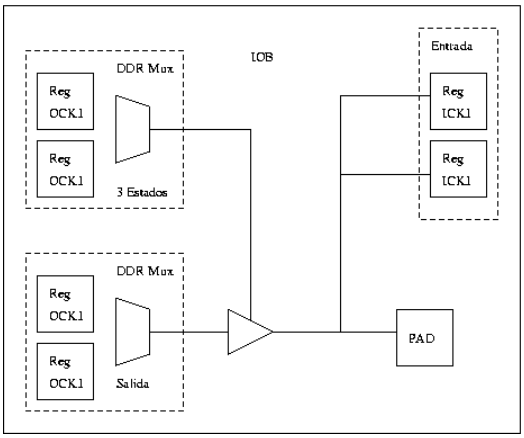


Figura A.2: Arquitectura Virtex II

Bloques Lógicos Configurables

Los bloques configurables del Virtex-II están organizados en un arreglo y su función es construir diseños de lógica combinatorial y síncrona. Cada elemento CLB se encuentra atado a una matriz de interruptores a partir del cuál accesan a la matriz de ruteo. Se componen de 4 unidades básicas conocidas como *slices* como se ilustra en la Figura A.3.

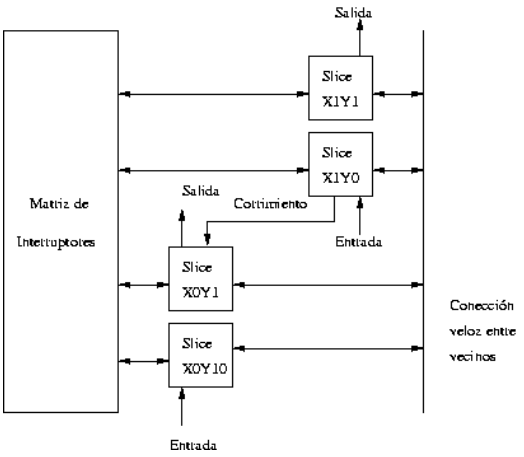


Figura A.3: Arquitectura Virtex II

Cada *slice* incluye dos generadores de funciones de 4 entradas, compuertas de lógica de acarreo, compuertas de lógica aritmética, multiplexores y dos

16K x 1 bit	2K x 9 bits
8K x 2 bits	1K x 18 bits
4K x 4 bits	512 x 36 bits

Cuadro A.2: Configuraciones de Puerto Simple y Doble.

elementos de almacenamiento. Los generadores de funciones pueden programarse como LUTs (tablas de consulta, por sus siglas en inglés *look-up table*) de 4 entradas, 16 bits de memoria RAM distribuida o como un registro de corrimiento de 16 bits.

Bloques de 18k de RAM

Los dispositivos Virtex-II incorporan varios bloques de 18K de memoria RAM para complementar la memoria RAM configurable dentro de un CLB. Los bloques de RAM incorporados de esta forma son conocidos como BRAM. Dichos bloques soportan diversos modos de configuración, incluyendo RAM de puerto simple o doble y diferentes relaciones de tamaño de datos/direccionamiento como se muestra en la Tabla A.2.

Multiplicadores 18x18 bits

Los dispositivos Virtex-II tienen varios bloques multiplicadores. Los multiplicadores implementados son de 18 bits por 18 bits con signo por complemento a 2. El bloque multiplicador se muestra en la Figura A.4.

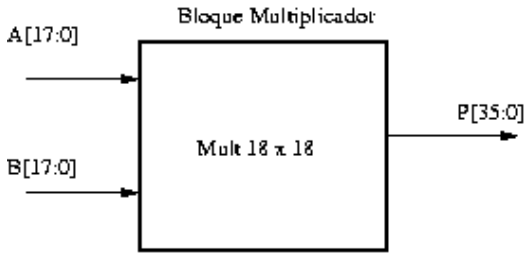


Figura A.4: Bloque Multiplicador

Dichos multiplicadores se encuentran optimizados para presentar un desempeño de velocidad alto y consumo de energía bajo comparado con multi-

XC2V40	4
XC2V80	8
XC2V250	24
XCV500	32
XCV1000	40
XCV1500	48
XCV2000	56
XCV3000	96
XCV4000	120
XCV6000	144
XCV8000	168

Cuadro A.3: Cantidad de multiplicadores en la familia Virtex-II

plicadores implementados a partir de CLBs. La cantidad de multiplicadores disponibles depende del dispositivo utilizado de acuerdo a la Tabla A.3.

Apéndice B

Longitud de la Expansión WτNAF

Solinas en sus publicaciones en el año 1997 [2] y en el año 2000 [3], presenta dos algoritmos para la generación de una expansión τNAF utilizando una ventana de precómputo de ancho ω. Los algoritmos son básicamente el mismo, la diferencia entre ellos son los representativos de clase de equivalencia mod τ^8 que utiliza.

En ambas publicaciones, Solinas demuestra que la longitud de una expansión τNAF, ℓ_{NAF}, para una curva elíptica $E_a[GF(2^m)]$ y de un equivalente modular reducido ρ de k, es siempre menor a $m+a$ donde a es el parámetro de la curva elíptica $E_a[GF(2^m)]$.

$$\ell_{NAF} \leq m + a + 3 \tag{B.1}$$

Se realizaron pruebas estadísticas de la longitud de la expansión WτNAF generadas por los Algoritmos 15 y 16 para una ventana de precómputo $w = 8$ y la curva elíptica $K-233$, $E_0[GF(2^{233})]$. La igualdad B.1 se cumplirá siempre para una expansión τNAF. Sin embargo, para expansiones WτNAF con ventanas de precómputo w mayores a 2, la longitud de la expansión puede variar aún más dependiendo de los representantes de clase de equivalencia utilizados.

La gráficas mostradas en las Figuras B.1 y B.2 presenta las diferentes longitudes ℓ_{NAF} obtenidas para 1000 enteros positivos tomados al azar. La Figura B.1 corresponde a los resultados obtenidos al aplicar el Algoritmo 15 y la Figura B.2 corresponde a las expansiones generadas por el Algoritmo 16.

	Algoritmo 15 [2]	**Algoritmo 16 [3]**
Longitud ℓ_{NAF} Mínima.	217	215
Longitud ℓ_{NAF} Máxima.	263	235
Promedio de la Longitud ℓ_{NAF}.	244	230.5
Desviación estándar de la Longitud ℓNAF.	8.29	2.62
Cantidad Promedio de elementos no-cero en la expansión.	27.7	26.4
Desviación estándar de elementos no-cero en la expansión.	1.07	0.767
Longitud de secuencias de ceros sucesivos promedio.	9.76	9.75
Desviación estándar en longitud de secuencias de ceros sucesivos.	1.0	1.0

Cuadro B.1: Tabla Comparativa para los Algoritmos 15 y 16

Observe que el segundo algoritmo no presenta expansiones mayores a 235, segun lo indicado por la Ecuación B.1.

Las gráficas mostradas en las Figuras B.3 y B.4 presentan los diferentes longitudes de sucesiones de elementos iguales a cero consecutivos. Solinas demuestra que nunca obtendremos secuencias consecutivas de ceros menores a $\omega-1$ para una ventana de precómputo de ω bits. Ambos algoritmos respetan dicha condición, y ambos presentan la misma longitud de secuencias de ceros promedio.

Para el caso del algoritmo 15 presentado por Solinas en [2], los representantes de equivalencia utilizados para una ventana de precómputo ω fueron los números $u = 1, 3, \ldots, 2^{w-1} - 1$. Las longitud máxima ℓ_{NAF} encontrada entre 1000 elementos generados al azar es de 263, la cuál no cumple con la restricción B.1.

Para el caso del algoritmo 16 presentado por Solinas en [3], utiliza como representantes de equivalencia los números $\alpha_u = u \bmod \tau^8$ para $u =$

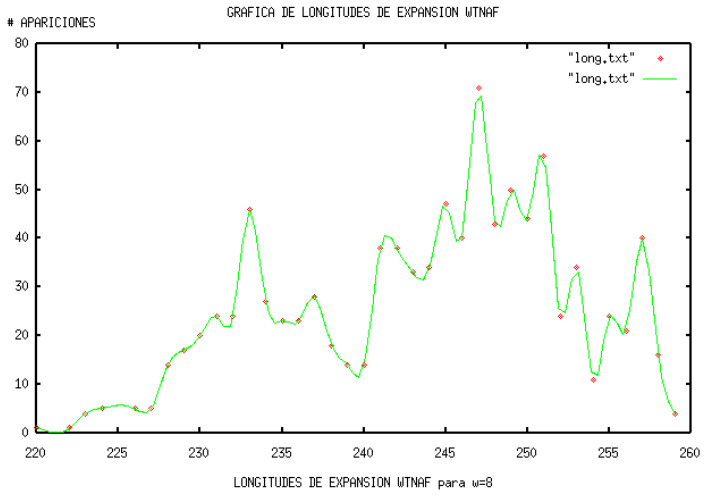


Figura B.1: Gráfica presentando las diversas longitudes de expansión WτNAF generadas por el Algoritmo 15

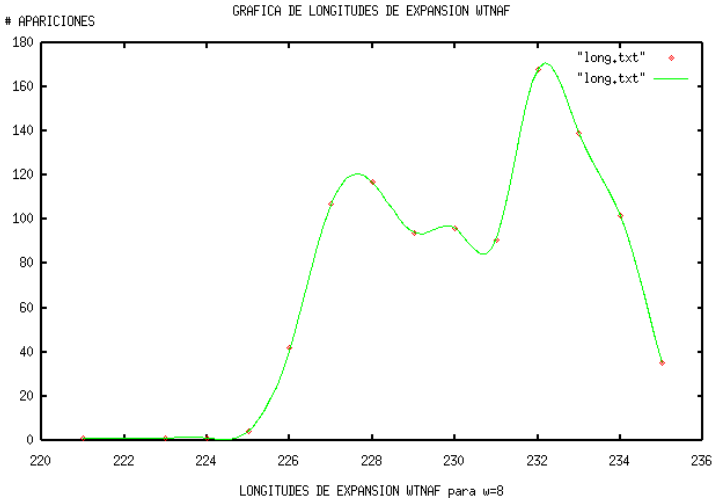


Figura B.2: Gráfica presentando las diversas longitudes de expansión WτNAF generadas por el Algoritmo 16

$1, 3, \ldots, 2^{w-1} - 1$. Las longitud máxima ℓ_{NAF} encontrada entre 1000 elementos generados al azar es de 235, la cuál cumple con la restricción B.1.

El primer algoritmo 15 no genera una expansión WτNAF óptima, sin

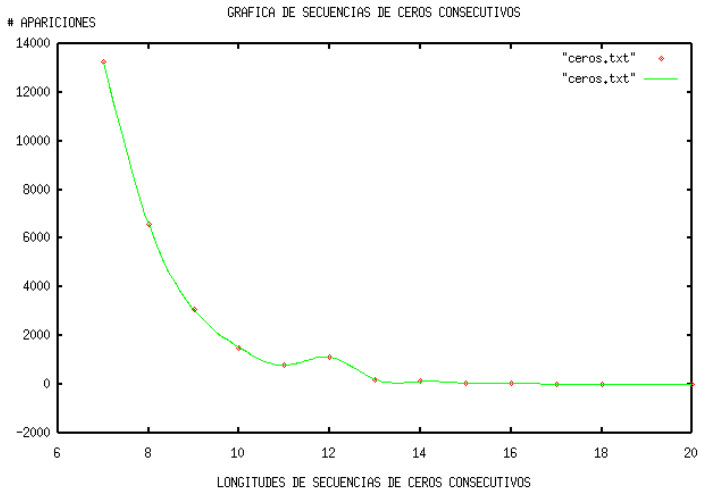


Figura B.3: Gráfica presentando las diversas longitudes de secuencias de elementos ceros en una expansión WτNAF generadas por el Algoritmo 15

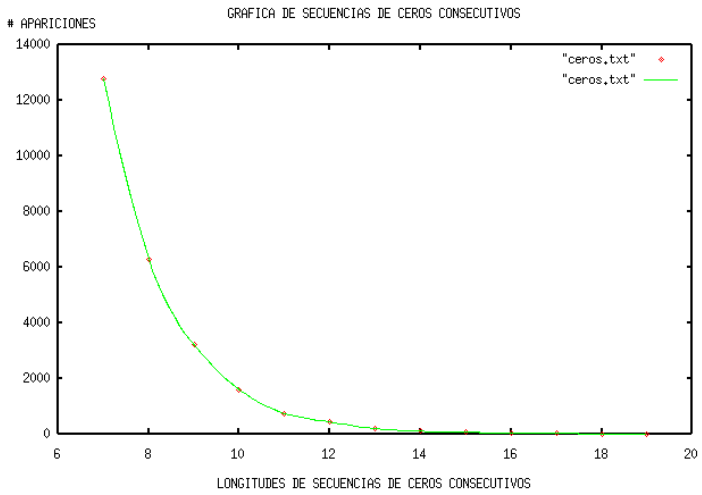


Figura B.4: Gráfica presentando las diversas longitudes de secuencias de elementos ceros en una expansión WτNAF generadas por el Algoritmo 16

embargo la ausencia de componentes complejas τ en los representantes de equivalencia permite una algoritmo de expansión simple.

El segundo algoritmo genera una expansión WτNAF óptima que incluso

cumple con la restricción B.1, sin embargo el algoritmo de generación de expansión 16 requiere de una adición extra y un proceso de indexado en las constantes α_u que lo complican.

Apéndice C

Representantes de Clase de Congruencia $mod\ \tau^8$

En el capítulo 2 se ha expuesto la posibilidad de representar a un escalar positivo k como un polinomio con coeficientes enteros y base compleja τ. $\mathbb{Z}[\tau]$ representa el anillo formado por todos los polinomios de base τ con coeficientes enteros. Dado que $\tau^2 = \mu\tau - 2$, todos los polinomios $\alpha \in \mathbb{Z}[\tau]$ pueden ser expresados en una forma canónica $\alpha = a_0 + a_1\tau$.

Dado que es posible encontrar una analogía directa entre los métodos de expansión NAF (representación polinómica base 2 con coeficientes enteros 1, 0 y -1) y de expansión τNAF (representación polinómica base τ con coeficientes enteros 1, 0 y -1), dicha analogía puede ser aplicada para proponer un método de ventana de precómputo WτNAF análogo al método existente WNAF.

En la literatura encontramos dos algoritmos WτNAF publicados por Solinas. En 1997, Solinas observa [2] que los números:

$$\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)$$

son incongruentes módulo τ^w. Por lo tanto propone calcular y almacenar los múltiplos uP para toda u impar en $2 < u < 2^{w-1}$.

En el año 200, Solinas observa [3] que los números:

$$\pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}$$

tal que $\alpha_u = u \bmod \tau^w$, también son incongruentes módulo τ^w. Proponiendo esta vez, calcular y almacenar los puntos $\alpha_u P$.

Para implementar el método propuesto por Solinas en [3] es necesario calcular todos los representantes de clases de congruencia $\alpha_u = u \bmod \tau^w$ para $u = 1, 3, \ldots, 2^{w-1} - 1$. En nuestro caso de estudio nuestra implementación presenta una ventana de precómputo $\omega = 8$ bits para la curva $K - 233$.

En [1] se presentan tablas con los correspondientes α_u para curvas Koblitz $a = 0$ y $a = 1$ y ventanas de precómputo $\omega = 3, 4, 5, 6$. En esta sección presentamos el cálculo de los representantes de congruencia para $K - 233$ y una ventana de precómputo $\omega = 8$.

Las cantidades que necesitamos calcular son $\alpha_u \bmod \tau^8$, para $u = 3 + 0\tau, 5 + 0\tau, \ldots, 127 + 0\tau$. Para realizar la operación modular es necesario calcular la forma canónica de τ^8. Solinas demuestra [3] que, dada la secuencia de Lucas U_k:

$$U_0 = 0,\ U_1 = 1 \text{ y } U_{k+1} = \mu U_k - 2U_{k-1} \text{ para } k \geq 1 \tag{C.1}$$

entonces

$$\tau^k = U_k\tau - 2U_{k-1} \text{ para } k \geq 1 \tag{C.2}$$

de esta forma obtenemos que la forma canónica de τ^8 es $3\tau - 14$. El siguiente paso es calcular $u + 0\tau \bmod -14 + 3\tau$ para toda $3 \geq u \geq 127$ impar. Solinas publica en [3] el Algoritmo 28 que realiza la operación $c_0 + c_1\tau \bmod d_0 + d_1\tau = q_0 + q_1\tau$ garantizando una norma $N(q_0 + q_1\tau)$ particularmente pequeña.

Procedimiento 28 Reducción Módular en $\mathbb{Z}[\tau]$

Entrada: $\gamma = c_0 + c_1\tau$ y $\delta = d_0 + d_1\tau$.
Salida: $\gamma \bmod \delta : \rho = r_0 + r_1\tau$

1: $g_0 \leftarrow c_0 d_0 + \mu c_0 d_1 + 2c_1 d_1$
2: $g_1 \leftarrow c_1 d_0 - c_0 d_1$
3: $N \leftarrow d_0^2 + \mu d_0 d_1 + 2d_1^2$
4: $\lambda_0 \leftarrow g_0/N$
5: $\lambda_1 \leftarrow g_1/N$
6: $(q_0, q_1) \leftarrow \text{RoundOff}(\lambda 0, lambda1)$ Alg. 14
7: $r_0 \leftarrow c_0 - d_0 q_0 + 2d_1 q_1$
8: $r_1 \leftarrow c_1 - d_1 q_0 - d_0 q_1 - \mu d_1 q_1$

Una vez aplicado dicho algoritmo a $3 \geq u \geq 127$, obtenemos la Tabla C.1.

u	$alpha_u$	u	$alpha_u$	u	$alpha_u$	u	$alpha_u$	u	$alpha_u$
1	1	3	3	5	5	7	7	9	$-5+3\tau$
11	$-3+3\tau$	13	$-1+3\tau$	15	$1+3\tau$	17	$3+3\tau$	19	$5+3\tau$
21	$7+3\tau$	23	$9+3\tau$	25	$-3+6\tau$	27	$-1+6\tau$	29	$1+6\tau$
31	$3+6\tau$	33	$5+6\tau$	35	$7+6\tau$	37	$9+6\tau$	39	$11+6\tau$
41	$-7-8\tau$	43	$-5-8\tau$	45	$-3-8\tau$	47	$-1-8\tau$	49	$1-8\tau$
51	$-11-5\tau$	53	$-9-5\tau$	55	$-7-5\tau$	57	$-5-5\tau$	59	$-3-5\tau$
61	$-1-5\tau$	63	$1-5\tau$	65	$3-5\tau$	67	$-9-2\tau$	69	$-7-2\tau$
71	$-5-2\tau$	73	$-3-2\tau$	75	$-1-2\tau$	77	$1-2\tau$	79	$3-2\tau$
81	$5-2\tau$	83	$-7+1\tau$	85	$-5+\tau$	87	$-3+\tau$	89	$-1+\tau$
91	$1+\tau$	93	$3+\tau$	95	$5+\tau$	97	$7+\tau$	99	$9+\tau$
101	$-3+4\tau$	103	$-1+4\tau$	105	$1+4\tau$	107	$3+4\tau$	109	$5+4\tau$
111	$7+4\tau$	113	$9+4\tau$	115	$11+4\tau$	117	$-1+7\tau$	119	$1+7\tau$
121	$3+7\tau$	123	$5+7\tau$	125	$7+\tau$	127	$9+7\tau$		

Cuadro C.1: $\alpha_u = u \bmod \tau^8$ para la curva elíptica $K-233$ y una ventana de precómputo $\omega = 8$

Bibliografía

[1] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[2] Jerome A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.

[3] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Des. Codes Cryptography*, 19(2-3):195–249, 2000.

[4] Neal Koblitz. CM-curves with good cryptographic properties. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287. Springer, 1991.

[5] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

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

[7] IEEE P1363 Working Group. Standard specifications for public key cryptography. In *IEEE P1363a, Draft Version 13, Working Group*. Available at http://grouper.ieee.org/groups/1363/., 2000.

[8] Klaus Schmeh. *Cryptography and public key infrastructure on the internet*. Wiley, 2003.

[9] Robert J. McEliece. *Finite field for scientists and engineers*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[10] Alfred J. Menezes and Ian F. Blake. *Applications of finite fields*. Kluwer, 1993.

[11] Abbas Saqib Nazar. *Efficient Implementation of Cryptographic Algorithms on Reconfigurable Hardware Devices*. PhD thesis, Centro de Investigaciones y de Estudios Avanzados del I.P.N., September 2004.

[12] David V. Chudnovsky and Gregory V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7:385–434, 1986.

[13] Julio Lopez and Ricardo Dahab. Improved algorithms for elliptic curve arithmetic in GF(2 n). In *Selected Areas in Cryptography, LNCS*, volume 1556, pages 201–212, 1998.

[14] National Institute of Standrads and Technology. Recomended elliptic curves for federal government use. Available at http://csrc.nist.gov/csrc/fedstandards.html, 1997.

[15] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2000.

[16] Eric Bach and Jeffrey Shallit. *Algorithmic number theory*. MIT Press, Cambridge, MA, USA, 1996.

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

[18] Andrew Rushton. *VHDL for Logic Synthesis*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[19] Gerardo Orlando and Christof 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*, volume 1965, pages 41–56, London, UK, 2000. Springer-Verlag.

[20] Nazar A. Saqib, Francisco Rodríguez-Henríquez, and Arturo Díaz-Pérez. A parallel architecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$. In *18th International Parallel & Distributed Processing Symposium (RAW 2004)*, pages 144–145, SantaFe, New Mexico, 2004. IEEE Computer Society Press.

[21] J. Lutz. *High Performance Elliptic Curve cryptographic co-processor*. Master Thesis, University of Waterloo, 2004.

[22] Nam Su Chang, Chang Han Kim, Young-Ho Park, and Jongin Lim. A non-redundant and efficient architecture for karatsuba-ofman algorithm. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2005.

[23] certicom. Ecc tutorial. http://www.certicom.com/index.php?action=ecc_tutorial,home.

[24] Panos C. Lekkas Radall K.Ñichols. *Wireless Security: Models, Threats and Solutions*. McGraw-Hill TELECOM, New York, NY, USA, 2002.

[25] Rajeev Murgai, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. *Logic Synthesis for Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[26] Xilinx. *Virtex-II Platform FPGAs: Complete Data Sheet*. Product Specification, 2005.