Trabajo fin de máster

Máster en Ingeniería de Sistemas e Informática

Co-diseño Hardware/Software para Criptografía de Curva Elíptica sobre plataformas en chip heterogéneas

Director: Jesús Javier Resano Ezcaray

Autor: David Ken Vallejo Miguel

Departamento de Informática e Ingeniería de Sistemas

Grupo de Arquitectura de Computadores de Zaragoza

Escuela de Ingeniería y Arquitectura

Programa oficial de posgrado en Ingeniería Informática

Curso 2015-2016

Diciembre 2015

Co-diseño Hardware/Software para Criptografía de Curva Elíptica sobre plataformas en chip heterogéneas

Resumen

Recientemente ha aparecido en el mercado un nuevo tipo de sistemas en chip heterogéneos que incluyen un multiprocesador basado en procesadores ARM y una FPGA (hardware programable al que se pueden asignar aceleradores en tiempo de ejecución). El objetivo de este trabajo ha sido el analizar cómo sacar partido a estas plataformas en el campo de la criptografía asimétrica de curva elíptica analizando las distintas posibilidades de codiseño hardware/software y sus compromisos entre coste y eficiencia.

Se han utilizado dos de los algoritmos criptográficos más representativos y eficientes en entornos embebidos: la multiplicación de Montgomery sobre coordenadas proyectivas y la multiplicación de Frobenius sobre curvas Koblitz. Posteriormente se ha analizado el software para determinar las partes más adecuadas para ser sustituidas por un acelerador hardware implementado en la FPGA. Resultando las operaciones más costosas las de aritmética sobre cuerpos finitos (Multiplicación, división e inversión).

Se ha demostrado posteriormente la escalabilidad de nuestro desarrollo implementando los algoritmos tanto sobre cuerpos GF(2^{163}) como GF(2^{233}). Cuerpos recomendados por el NIST (National Institute of Standards an Technology) [11,12] y el SECG [6] para aplicaciones en criptosistemas de curva elíptica.

Se han desarrollado los aceleradores hardware en la parte de la lógica programable proporcionada por la plataforma en forma de dispositivos con registros accesibles y direccionables desde el software. La aritmética modular en hardware es de sobra conocida y en este trabajo se han desarrollado e integrado componentes ampliamente utilizados.

Se han conseguido aceleraciones muy importantes, mientas que el consumo medio se ha mantenido, incluso disminuyéndose ligeramente, con lo que el ahorro energético se multiplica. Siendo un aspecto crítico en los dispositivos embebidos y con restricciones tales como tarjetas inteligentes y dispositivos móviles.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. DAVID-KEN VALLEJO MIGUEL,

con nº de DNI 72971851-G en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) EN INGENIERÍA INFORMÁTICA, (Título del Trabajo) CO-DISEÑO HARDWARE/SOFTWARE PARA CRIPTOGRAFÍA DE CURVA ELÍPTICA SOBRE PLATAFORMAS EN CHIP HETEROGÉNEAS

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 DE NOVIEMBRE DE 2015

Índice

Índice de algoritmos

Índice de figuras

Índice de tablas

1. Introducción

El *objetivo* de este trabajo es analizar cómo sacar partido a las nuevas plataformas heterogéneas, que combinan en un solo chip un sistema basado en procesadores ARM (como el que se puede encontrar en la mayor parte de dispositivos móviles o empotrados de altas prestaciones) con hardware reprogramable (FPGA), en el campo de la criptografía asimétrica de curva elíptica analizando las distintas posibilidades de codiseño hardware/software y sus compromisos entre coste y eficiencia.

La criptografía asimétrica de curvas elípticas constituye un nuevo caso de uso de estas plataformas en el *contexto* de nuestro grupo de investigación, ya que hasta la fecha se había trabajado más en la línea de la inteligencia artificial y los juegos como el Reversi [9]. Es un caso de uso de gran interés y aplicación debido a la ubiquidad de los dispositivos móviles y al incremento de la demanda de las comunicaciones y de la seguridad.

En *líneas generales*, partiremos de la implementación software de algoritmos representativos que usan distintos enfoques para aumentar la eficiencia, a continuación se analizará el software para determinar las partes más adecuadas para ser sustituidas por un acelerador hardware y procederemos a implementarlas en la FPGA. Por último se tomarán medidas y se compararán los distintos desarrollos entre sí y desarrollos publicados por otros autores.

El resto de la memoria sigue la estructura de contenidos que se indica a continuación:

La sección 2 describe los algoritmos software implementados en el procesador y su análisis.

La sección 3 detalla la implementación hardware de las diferentes operaciones que se ha llevado a cabo.

La sección 4 muestra los resultados en términos de rendimiento y consumo.

La sección 5 expone las conclusiones obtenidas a raíz del trabajo realizado.

La sección 6 contempla las posibles líneas de trabajo futuro.

La sección 7 contiene las referencias más relevantes utilizadas en este trabajo.

Finalmente, el anexo I contiene el un resumen de los conceptos matemáticos necesarios para el desarrollo.

1.1 El chip, arquitectura heterogénea

La plataforma sobre la que se realiza el trabajo es la ZedBoard (figura 1), una placa heterogénea basada en el sistema en chip Zynq completamente programable (All-

programmable SoC [10]) de la empresa Xilinx compuesta principalmente por un procesador estándar ARM Cortex-A9 dual-core [1] - procesador a nivel de aplicaciones capaz de ejecutar un sistema operativo completo como Linux- y lógica programable basada en las FPGAs Artix®-7 y Kintex®-7 [5] y [7].

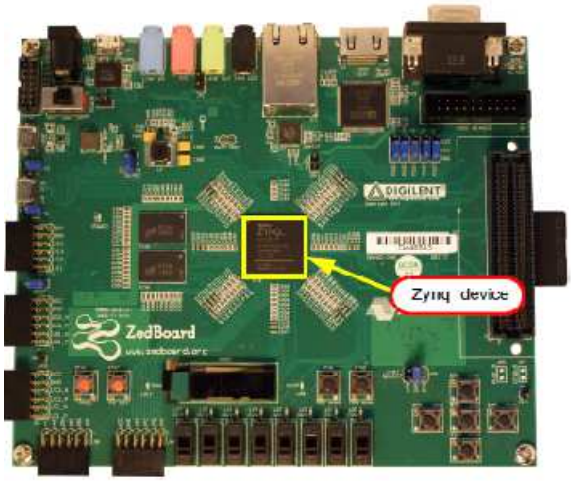

Fig. 1. Vista superior de la ZedBoard

Para la interconexión de ambas partes se cuenta con un interfaz AXI que permite aumentar el ancho de banda con conexiones de baja latencia [8] (figura 2). Esta organización da lugar a un sistema que permite aprovechar al máximo la especialización de las dos partes sin pagar un precio muy alto por la comunicación (*overhead*).

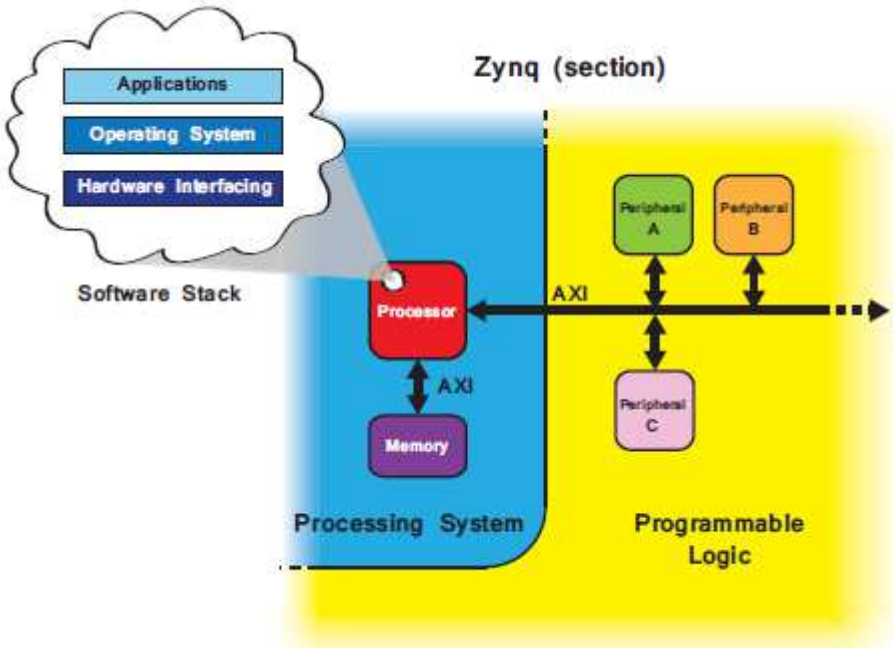

Fig. 2. Relación entre el procesador, hardware y arquitectura Zynq

Una FPGA (Field ProgrammableGateArray) es un circuito integrado que contiene bloques de lógica, elementos de memoria e interconexiones, todos ellos programables, así como bloques específicos de E/S. La configuración de la FPGA mediante la interconexión de los

bloques lógicos y la funcionalidad de los mismos, permite generar el sistema lógico deseado. Esta configuración se puede realizar tantas veces como se desee, incluso con el sistema funcionando, por lo que los mismos recursos hardware pueden utilizarse para tareas distintas según sean las necesidades del sistema. Debido a su flexibilidad las FPGAs no son tan eficientes como los circuito integrado para aplicaciones específicas (o ASICs, por sus siglas en inglés), pero a cambio presentan menores tiempos de desarrollo y menores costes de fabricación para tiradas bajas o medias.

1.2 Caso de uso. La criptografía asimétrica implementada con Curvas Elípticas

En 1985, Miller [23] y Koblitz [22], propusieron independientemente un criptosistema de clave pública análogo a los esquemas de ElGamal [24] en el que, el grupo multiplicativo de enteros módulo p, denotado por Z_p^*, se sustituye por el grupo de puntos de una curva elíptica definida sobre un cuerpo finito. El mejor algoritmo conocido para resolver el problema matemático subyacente es computacionalmente muy difícil, el problema del *logaritmo discreto de curva elíptica* (ECDLP de sus siglas en inglés), requiere tiempo totalmente exponencial. Mientras que los mejores algoritmos matemáticos conocidos para resolver los problemas subyacentes en RSA (factorización de enteros) y DSA (el logaritmo discreto) toman tiempo sub-exponencial. Por tanto los parámetros son significativamente más pequeños en criptografía de curva elíptica (ECC) que en otros sistemas, como RSA y DSA. Por ejemplo, una clave ECC de 163 bits tiene un nivel de seguridad comparable al RSA y DSA con módulos de 1024 bits. Esto quiere decir que mediante el uso de ECC se puede alcanzar el mismo nivel de seguridad con menor potencia de procesamiento, espacio de almacenamiento, ancho de banda y energía eléctrica, lo cual hace especialmente interesantes estos criptosistemas para aplicaciones en dispositivos con restricciones tales como tarjetas inteligentes, teléfonos y dispositivos móviles.

El rendimiento de ECC depende principalmente de la eficiencia de las operaciones sobre cuerpos finitos y de la definición de algoritmos rápidos para multiplicaciones escalares elípticas (apartado 2). La selección de los cuerpos finitos o las curvas subyacentes también puede incrementar el rendimiento (anexos).

1.3 Trabajo previo y relacionado

Existen numerosos trabajos en el campo de la criptografía de curva elíptica (ECC de sus siglas en inglés) tanto desarrollos en plataformas ASIC, como en plataformas reprogramables o FPGAs. Los desarrollos comparten la filosofía de la transferencia de toda la responsabilidad de las operaciones al hardware. En la tabla comparamos algunas de estas implementaciones de ECC. En [15] se presenta una arquitectura para ECC sobre cuerpos binarios, escalable en área y velocidad y adaptable para distintas curvas y cuerpos. Una arquitectura que consta de 3 componentes: Controlador principal, Unidad Aritmética (UA) y el Controlador de la UA. Implementan el algoritmo de Montgomery en coordenadas proyectivas (uno de los más

eficientes y más utilizados, que nosotros empleamos también en este trabajo). Consiguen realizar la multiplicación escalar en GF(2^{167}) en 0,21 ms.

En [11] también presentan un acelerador hardware para ECC en cuerpos binarios, escalable hasta polinomios de grado m=255, con los principales tipos de curvas cableadas y especial atención al multiplicador (con ejecución paralela y separación de las rutas de control y datos) consiguen una multiplicación en GF(2^{163}) de solo 0,14 ms.

plataforma	nombre	Cuerpo y método de multiplicación	tm [ms]
FPGA	Xilinx XCV400E @76.7MHz [15]	GF(2^{167}) Montgomery (Digit Size D=16)	0.21
	Xilinx XCV2000E-FG680-7 @66.4MHz [18]	GF(2^{163}) Montgomery	0.14
ASIC	Coprocessor VLSI @40MHz [19]	GF(2^{155})	3.90
	CE71 0,25¹m 165k gates @66MHz [20]	GF(2^{163}) Random: modified SSM multiplier Koblitz: modified SSM multiplier	 1.10 0.65

Tabla1. Tiempos de multiplicación escalar en ECC en distintas plataformas

En artículos más recientes, se usa la técnica del codiseño hardware-software (HSC de sus siglas en inglés) como en [16 y 17]. Con esta aproximación se reduce la complejidad del hardware eliminando mucha lógica de control, disminuyendo el tiempo de desarrollo y reduciendo también el área y coste. En [16] presentan una plataforma parecida a la nuestra compuesta por un procesador software PicoBlaze y una FPGA de Xilinx. Su arquitectura trabaja con distintas tamaños de palabra (8-bit, 16-bit y 32-bit) en la ruta de datos y es escalable tanto en software como en hardware, siendo capaz de trabajar con distintos cuerpos de polinomios. Pero, aunque su control reside en el procesador software PicoBlaze, su aproximación implementa un segundo procesador ECC implementado en la FPGA que contiene memoria, una ALU y multiplexores). La única diferencia entre las aproximaciones de [16] y [17] son el tipo de curvas y la implementación elegida. En el primero usan Montgomery con coordenadas proyectivas y en el segundo curvas Koblitz con la multiplicación o mapa de Frobenius. Ambos algoritmos se han usado en este desarrollo y pueden consultarse en la sección 2 y en los Anexos. En la tabla 9 se pueden ver los resultados de estos artículos junto con otros arquitecturas, todos con anchura de 32 bits, tamaño de palabra utilizado en este trabajo. Otro trabajo más reciente todavía es el [21] en el cual desarrollan un procesador ECC completo que implementa el algoritmo de multiplicación Montgomery López-Dahab. Esta arquitectura es completamente escalable y presenta unos resultados solo por debajo de [18] que no tiene la característica de escalabilidad.

Estos trabajos previos demuestran el interés por optimizar las computaciones con curvas elípticas utilizando FPGAs. Consideramos que nuestro trabajo complementa a estos trabajos previos al estudiar las posibilidades de codiseño con curvas elípticas en una arquitectura nueva que ha tenido una enorme acogida tanto en industria como en investigación.

1.4 Objetivos de este trabajo

El objetivo principal es estudiar las posibilidades que ofrece el codiseño hardware software en el campo de la criptografía elíptica al utilizar las nuevas plataformas en chip heterogéneas ARM/FPGA. Para ello:

- Se han identificado los algoritmos más representativos
- Se ha implementado en C y se ha realizado un análisis (profiling) de la demanda computacional de cada uno de sus componentes
- Se han seleccionado las funcionalidades más críticas y se han implementado tres aceleradores hardware de forma que el procesador puede solicitar a los aceleradores que realicen un cálculo dado haciendo una llamada a una función.
- Se han analizado los resultados obtenidos evaluando las ganancias derivadas de la inclusión de estos aceleradores

2. Aplicación software

En esta sección se van a describir los algoritmos utilizados y su implementación. En el primer apartado se va a describir la operación de multiplicación de puntos de la curva en la cual radica la seguridad criptográfica. En el segundo y tercer apartado se describen los algoritmos utilizados en este trabajo y en el último apartado se presenta el análisis del comportamiento del software usado para determinar las funciones candidatas a ser aceleradas en el hardware de la FPGA.

2.1 Multiplicación escalar de puntos en curvas elípticas

La multiplicación de puntos de la curva constituye la operación básica de la criptografía de curva elíptica (ECC en inglés) y en la cual radica la seguridad criptográfica, ya que la operación inversa, dados el punto resultado y el punto usado como generador, obtener el número usado como operando de la multiplicación es computacionalmente inviable. Es decir dados $M = k \cdot P$, con M y P puntos pertenecientes a la curva y k un número natural, a partir de M y P es muy difícil la obtención de k.

Las operaciones definidas sobre el cuerpo que forman los puntos de la curva son la suma de puntos (S = Q + P) y el duplicado de un punto (D = Q + Q = 2·Q) estas operaciones se definen de manera sencilla geométricamente (ver Anexo).

Básicamente la operación de multiplicación se define como k sumas de puntos:

$$kP = P + P + \ldots + P \text{ (k veces)} \quad \forall k > 0 \text{ and } 0 \cdot P = \infty$$

Asumiendo que el número de puntos de la curva (#E(L)) elegida pueda ser factorizado como *#E(L) = nh.* Siendo *n* primo y *h* (el cofactor) pequeño, para que *n* sea aproximadamente igual al orden del cuerpo (número de elementos).

Algoritmo básico. Sumar y duplicar

Dada la representación binaria de k (k_{t-1}, k_{t-2}, . . . , k_0), es decir

$$k= k_{t-1} \cdot 2^{t-1} + k_{t-2} \cdot 2^{t-2} + \ldots + k_0 \cdot 2^0 \text{cont} \cong log2n \cong log2q$$

De esta manera, la multiplicación k·P puede ser obtenida según el esquema:

$$K \cdot P = (\ldots 2(2(2\infty + k_{t-1}P) + k_{t-2}P) + \ldots) + k_0P$$

Algoritmo 1 Método Binario Izquierda-Derecha para la multiplicación escalar [13].

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P = (x, y) \in E(F2^m)$.
OUTPUT: kP.
1: $Q \leftarrow O$. - - Point At Infinite
2: **for** $i = m$ -1 downto 0 **do**
3: $Q \leftarrow 2Q$.
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + P$.
6: **end if**
7: **end for**
8: **return** Q.

El algoritmo consiste en *t* operaciones de duplicar punto y como mucho *t* operaciones de suma de puntos. El orden de magnitud de *t* es *log2q = mlog2p*si*L = GF(pm).* Ambas operaciones son complejas e incluyen divisiones. Por eso se desarrollan y se han elegido para este trabajo, los métodos descritos a continuación. Cada coordenada pertenece a la extensión del cuerpo finito, es decir a polinomios cuyos coeficientes pertenecen al cuerpo (Ver anexos).

2.2 Montgomery en coordenadas proyectivas

La representación estándar de los puntos de la curva se realiza en dos dimensiones L^2, siendo estas las denominadas coordenadas afines. Se pueden transformar estas coordenadas a tres dimensiones, llamadas coordenadas proyectivas. Dados dos naturales *c* y *d*, un subconjunto de L^3 puede ser asociado para todo elemento (x, y) de L^2 [14]:

$$(x, y) \rightarrow \{(X, Y, Z) \mid Z \neq 0, X = xZ_c, Y = yZ_d\}$$

En el sentido opuesto, un elemento (X, Y, Z) de L3, con Z ≠ 0, corresponde con el elemento (x, y) de L^2 y es definido por

$$x = X/Z_c \; y = Y/Z_d$$

Esta transformación establece una relación de equivalencia en L^3. En la que dos puntos pertenecen a la misma clase si se corresponden con el mismo punto en L^2. Esta clase de equivalencia es el denominado punto proyectivo.

Se escoge la tercera coordenada de manera que nos permite evitar las operaciones de división. Incrementando por contra las operaciones por iteración al obligarnos a trabajar con una coordenada adicional [14].

Este método de coordenadas proyectivas se puede aplicar a cualquier técnica. Se debe transformar la ecuación de Weierstrass que define la curva en coordenadas afines, a coordenadas proyectivas, y definir las operaciones de suma de puntos y doblado de puntos en función de esta nueva ecuación (eligiendo la tercera coordenada Z, para poder simplificar las operaciones) [13, 14].

Esta técnica es especialmente útil combinada con la multiplicación de Mongomery, que permite reducir las operaciones en cada iteración, al trabajar o iterar con una coordenada menos. Considerando de nuevo la representación binaria de k, $k = k_{t-1}2^{t-1} + k_{t-2}2^{t-2} + \ldots + k_1 2^1 + k_0 2^0$, se definen las sumas parciales

$s_0 = 0$
$s_1 = k_{t-1}2^0$
$s_2 = k_{t-1}2^1 + k_{t-2}2^0$
. . .
$s_t = k_{t-1}2^{t-1} + k_{t-2}2^{t-2} + \ldots + k_1 2^1 + k_0 2^0 = k$

Así $s_j = 2s_{j-1} + k_{t-j} \; \forall j = 1, 2, \ldots, t$

El algoritmo consiste en calcular en cada paso $A = s_j P$ y $B = (s_j+1)P$ en función de $s_{j-1}P$ and $(s_{j-1}+1)P$ (A y B en la iteración anterior) [13]:

Si $k_{t-j} = 0$ **entonces**
  $s_j P = 2(s_{j-1}P)$, $(s_j+1)P = (2s_{j-1}+1)P = s_{j-1}P + (s_{j-1}+1)P$ // A = 2·A y B = A + B
sino // $kt - j = 1$
  $s_j P = (2s_{j-1}+1)P = s_{j-1}P + (s_{j-1}+1)P$ // A = A + B
  $(s_j+1)P = (2s_{j-1}+2)P = 2(s_{j-1}+1)P$ // B = 2·B
fin

Este algoritmo, aplicado sobre curvas elípticas no-supersingulares (sobre cuerpos binarios) es debido a López y Dahab [13]. Con $A = (x_A, y_A) \neq \infty$ y $B = (x_B, y_B) \neq \infty$ dos puntos diferentes de la curva y si $A \neq -B$, las x-coordenadas x_{A+B} y x_{A-B} de A + B y A − B están relacionadas por la siguiente ecuación $x_{A+B} = x_{A-B} + x_B(x_A + x_B)^{-1} + (x_B(x_A + x_B)^{-1})^2$. Como además (ver Montgomery más arriba) $A = s_j P$ and $B = (s_j + 1)P$ para todo j, entonces A - B = - P, luego en curvas no-supersingulares, las coordenadas x coinciden $x_{A-B} = x_P$ (ver anexos), siendo P el punto original o generador y por tanto

$$x_{A+B} = x_P + x_B(x_A + x_B)^{-1} + (x_B(x_A + x_B)^{-1})^2$$

De la misma manera, para el doblado de puntos (ecuaciones de curvas no-supersingulares, ver anexos), se tiene

$$x_{A+A} = x_A^2 + b/x_A^2 \text{ Si } x_A \neq 0 \text{ y } (0, y_A) + (0, y_A) = (0, y_A) - (0, y_A) = \infty$$

Como puede verse se puede operar sin la coordenada y. Al final del proceso iterativo, ésta debe calcularse. De las propiedades de Montgomery, se tiene que $P = (x_P, y_P)$, donde $x_P \neq 0$, $kP = (x_A, y_A)$ y $(k+1)P = (x_B, y_B)$, por tanto

$$y_A = x_P^{-1}(x_A + x_P)[(x_A + x_P)(x_B + x_P) + x_P^2 + y_P] + y_P$$

Pasando a coordenadas proyectivas con c = 1 y d = 2 (coordenadas López-Dahab) pasamos a iterar con las coordenadas X y Z en cada iteración y finalmente calculamos Y deshacemos el cambio de coordenadas pasando de nuevo a afines (evitando las operaciones de división en la iteración), quedando el algoritmo implementado de la siguiente manera:

Algoritmo 2: Multiplicación de puntos de Montgomery de López-Dahab sobre GF(2^m) [13].

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, con $k_{t-1} = 1$ $P = (x, y) \in E(F2^m)$.
OUTPUT: kP.
1. $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$. {Calcula $(P, 2P)$}
2. **For** I from $t-2$ downto 0 **do**
2.1 **If** $k_i = 1$ **then**
$T \leftarrow Z_1$, $Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_1 \leftarrow x Z_1 + X_1 X_2 T$ Z_2.
$T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + b Z_2^4$, $Z_2 \leftarrow T^2 Z_2^2$.
2.2 **Else**
$T \leftarrow Z_2$, $Z_2 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_2 \leftarrow x Z_2 + X_1 X_2 Z_1 T$.
$T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + b Z_1^4$, $Z_1 \leftarrow T\, {}_2 Z_1^2$.
3. $x_3 \leftarrow X_1 / Z_1$.
4. $y_3 \leftarrow (x + X_1/Z_1)[(X_1 + x Z_1)(X_2 + x Z_2) + (x^2 + y)(Z_1 Z_2)](x Z_1 Z_2)^{-1} + y$.
5. **Return**(x_3, y_3).

Este algoritmo realiza las mismas operaciones en cada iteración, por lo tanto incrementa su resistencia a ataques de tiempo y de análisis de consumo. Siendo un claro candidato a ser implementado tanto en software como hardware.

2.3 Curvas Koblitz y mapa de Frobenius

Las curvas Koblitz son un tipo especial de curvas elípticas no-supersingulares sobre GF(2^m) [13, 14]:

$$E_0: y^2 + xy = x^3 + 1$$

$$E_1: y^2 + xy = x^3 + x^2 + 1$$

Es decir curvas no-supersingulares E_c con coeficientes $b = 1$ y $a = 0$ (E_0) o $a = 1$ (E_1).

Estas curvas permiten simplificar las operaciones de Duplicado de puntos en su totalidad, sustituyéndolas por operaciones de *squarer* de polinomios, muy rápidas y eficientes en cuerpos binarios [13, 14]. Para sustituir las operaciones de doblado se utiliza el mapa o aplicación de Frobenius τ de *Ec(GF(2^m))* a *Ec(GF(2^m))* con

$$\tau(\infty) = (\infty) \quad y \quad \tau(x, y) = (x^2, y^2)$$

Se puede demostrar que

$$2P = -\tau^2(P) + \mu\tau(P) \text{ con } \mu = 1 \text{ si } a = 1 \text{ y } \mu = -1 \text{ if } a = 0$$

Como τ es una aplicación, cualquier combinación lineal será otra aplicación en *Ec(GF(2^m))*. Por tanto el resultado anterior puede generalizarse para cualquier aplicación *α* de *Ec(GF(2^m))* a *Ec(GF(2^m))* en la que, dados dos enteros *a* y *b*, *α(P) = aP + bτ(P)*.

Se puede realizar una especie de división entre las aplicaciones *α* y *τ* de tal manera que α = α'τ + r con r ∈ {−1, 0, 1}. Las sucesivas divisiones, de manera parecida a las divisiones euclídeas en el cuerpo finito de los enteros Z_p, proporcionarán una serie de restos que permiten descomponer la función inicial α de manera que

$$\alpha = r_0 + r_1\tau + \ldots + r_{t-1}\tau^{t-1} + \alpha_t\tau^t$$

Se puede demostrar que $\alpha_t = 0$ después de un número finito de pasos. Así, considerando el caso particular de la función α con coeficientes a = k y b = 0, es decir, α(P) =kP, tendremos

$$kP = r_{t-1}\tau^{t-1}(P) + r_{t-2}\tau^{t-2}(P) + \ldots + r_1\tau(P) + r_0P \text{ con } r_i \in \{-1, 0, 1\}$$

El objetivo de las sucesivas divisiones es que Ki sea cero el mayor número de veces posible. Así si k no es divisible por *τ* en alguna iteración, escogemos $r_i \in \{-1, 1\}$, de manera que en *(κ −r)/τ* sea divisible por *τ.* Así en la siguiente iteración, el siguiente r_i sea 0. De esta manera se puede definir la multiplicación de puntos en la curva como sigue

---

Algoritmo 3: Multiplicación de puntos Koblitz sobre GF(2^m) [13].

INPUT: k ∈ [1 .. n-1] y P = *(x, y) ∈ E(F2^m) y de orden n*.
OUTPUT: *kP*.
1. *Q ← ∞; A ← K; B ←0;*
2. **While** ((A /= 0) or (B /= 0)) **do**
2.1 **if** A mod 2 = 0 **then** R_I ← 0**;**
2.2 **elsif** 2 - ((A - 2*B) mod 4) = 1 **then**
R_I ← 1;
if *Q* = ∞ **then** Q ← P**; else** Q ← Q + P; **endif**;
2.2 **else**
R_I ← - 1;
if *Q* = ∞ **then** Q ← - P**; else** Q ← Q – P; **endif**;
2.3 **endif**;
2.4. P ← τ (P); --Elevar al cuadrado cada coordenada
2.5 T ← A; A ← B + *μ* (T - R_I)/2; B ← (R_I - T)/2;

3. **Return***(Q)*.

2.4 Profiling. Análisis del software

Entre las principales herramientas se ha contado con el Vivado IDE de Xilinx para la programación hardware sobre la placa y el IDE Xilinx SDK para el desarrollo software y la integración con el hardware.

Con la segunda herramienta se ha llevado a cabo la implementación de los algoritmos descritos en los apartados anteriores completamente en lenguaje C y se ha compilado con el compilador de Xilinx para ARM con el comando *arm-xilinx-eabi-gcc*. El entorno integrado contiene una utilidad para el análisis del código o profiling de modo intrusivo que está basada en la herramienta *GNU gprof* [26]. La herramienta nos proporciona dos tipos de información para poder optimizar el software: un histograma con los tiempos de ejecución para cada función y un grafo de llamadas a función que indica quien llama a quien y cuántas veces.

Al compilar con la opción '-pg' para el análisis cada vez que se llama a una función se invoca a la función *mcount* para registrar las funciones invocante e invocada (profiling intrusivo). Así aparece en los resultados consumiendo el 50% del tiempo de ejecución total, como puede verse en la tabla 2.

Quitando la función mencionada, y que sólo debe ejecutarse al realizar el proceso de análisis, las siguientes funciones que más tiempo consumen son las de acceso a los vectores de bits que representan los polinomios, elementos del cuerpo extendido y puntos de la curva. Las funciones bitv_assign y bitv_get representan el 14,22% y el 11,8%. Ambas funciones son invocadas principalmente, como puede verse en los *parents* de cada una por las funciones de *Product* y *Multiply_By_X*. Siendo la que llama a ambas *Product_Mod_F*. Ésta es la función candidata a ser acelerada por el hardware. Función que realiza la multiplicación de polinomios (coordenadas de los puntos de la curva).

Las operaciones de división de polinomios sólo se invocan al deshacer el cambio de coordenadas al finalizar el proceso iterativo, así que tienen menor relevancia que la multiplicación (tabla 2). Sin embargo puede verse que las dos operaciones de división invocan a *bitv_get* y a *Product*, e indirectamente a bitv_assign. Funciones que tienen mucha relevancia en el rendimiento (figura 3). Por tanto finalmente se considera que la función *Divider_Mod_F* también es una buena candidata para ser acelerada por el hardware. Se decide implementarla en una segunda versión de nuestro desarrollo.

Name (location)	Samples	Calls	Time/Call	% Time
mcount	326607			50,25%
◢ bitv_assign	92397	119320507	77ns	14,22%
◢ parents	73177	119320507	61ns	11,26%
Product (Galois.c:-1)	49193	78856120	62ns	7,57%
Multiply_By_X (Galois.c:-1)	23340	39401175	59ns	3,59%
Shift_One (Galois.c:-1)	644	1062720	60ns	0,1%
Hex_To_Bin (Galois.c:-1)	0	492	0ns	0,0%
▷ children	0	119320507	0ns	0,0%
◢ bitv_get	76673	119324080	64ns	11,8%
◢ parents	73480	119324080	61ns	11,3%
Product (Galois.c:-1)	49193	78856120	62ns	7,57%
Multiply_By_X (Galois.c:-1)	23340	39162380	59ns	3,59%
Shift_One (Galois.c:-1)	644	1056240	60ns	0,1%
Product_Mod_F (Galois.c:-1)	284	238795	118ns	0,04%
Divide_By_X (Galois.c:188)	13	6480	200ns	0,0%
Divider_Mod_Binary_F (Galois.c:530)	6	3250	184ns	0,0%
Montgomery_Point_Multiplicationy_Pro	0	815	0ns	0,0%
▷ children	0	119324080	0ns	0,0%
◢ Product	49193	480830	10.230us	7,57%
▷ children	169241	157952500	107ns	26,04%
◢ parents	23637	480830	4.915us	3,64%
Multiply_By_X (Galois.c:254)	23340	238795	9.774us	3,59%
Product_Mod_F (Galois.c:-1)	284	238795	118ns	0,04%
Divide_By_X (Galois.c:189)	13	3240	401ns	0,0%
◢ Multiply_By_X	23340	238795	9.774us	3,59%
▷ children	223463	79520200	281ns	34,38%
◢ parents	284	238795	118ns	0,04%
Product_Mod_F (Galois.c:-1)	284	238795	118ns	0,04%

Tabla 2. Profiling del software

Las cuatro funciones que más tiempo consumen (sin contar la función intrusiva usada para el análisis *mcount*) serán cubiertas en su totalidad por los tres aceleradores hardware que se decide implementar (Multiplicador, Elevador al cuadrado y Divisor de polinomios módulo f(x)) (figura 3).

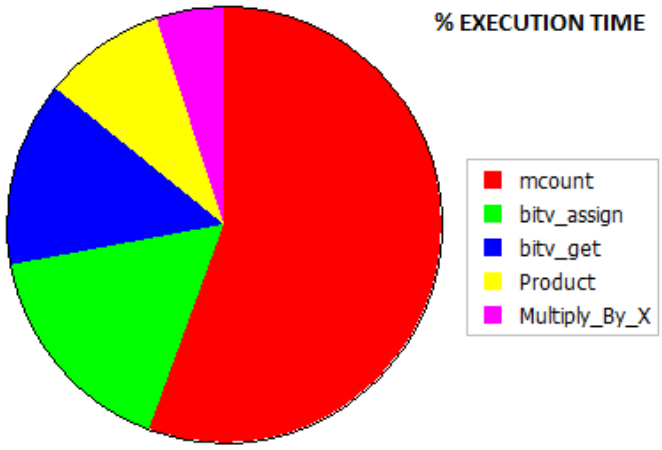

Fig. 3. Distribución del tiempo de ejecución por funciones

3. Desarrollo Hardware

En este apartado se va a detallar el desarrollo hardware llevado a cabo en el trabajo. La idea ha sido la de implementar nuevas funciones mediante aceleradores accesibles desde el software a través de escrituras y lecturas en registros direccionables por el procesador. Los objetivos han sido minimizar el tamaño y el consumo del hardware e intentar acelerar al máximo la ejecución de las operaciones más costosas detectadas durante el análisis del software. Las operaciones de suma y doblado de puntos (ver anexos) se componen de numerosas multiplicaciones. Aproximadamente la mitad son operaciones de elevar al cuadrado un polinomio, así que se ha tomado la decisión de desarrollar dos componentes: el *multiplier* y el *squarer* en la primera versión y el *divider* en la segunda.

Los componentes que implementan aritmética modular son altamente conocidos y hay muchos desarrollos disponibles en el mercado. En este trabajo se han seleccionado para cumplir los objetivos de mínima área y coste. Posteriormente se les ha dotado de un interfaz AXI, que explicamos a continuación y además se ha dotado de una Unidad de Control muy sencilla tanto al multiplicador como al divisor.

3.1 Interfaz AXI

AXI significa Advanced eXtensible Interface, su actual versión es la 4 y es parte del estándar abierto ARM AMBA 3.0. Estándar de facto para la comunicación en chip. La plataforma nos proporciona tres posibles protocolos para la comunicación, en función de las necesidades de la conexión:

- AXI4: transferencia con memoria mapeada en la que se proporciona una dirección y un tamaño de ráfaga de hasta 256 palabras.
- AXI4-Lite: transferencia de una sola palabra a una sola dirección, también mapeada en memoria
- AXI4-Stream: ráfagas de transferencia sin límite de tamaño. No tiene mecanismo de direccionamiento, es un flujo directo entre el origen y el destino. Se indica el dispositivo únicamente (memoria no mapeada).

Debido a que los polinomios ocupan pocas palabras de transferencia se ha escogido la opción AXI4-Lite que además era la más sencilla. Y de los nueve interfaces AXI que proporciona la plataforma, se ha escogido el M_AXI_GP0, de propósito general y muy apropiado para transferencias de baja a media intensidad. Interfaz directo que no incluye almacenamiento en memoria intermedia (buffering) en el que el procesador es *master* y los aceleradores integrados *slave*.

Para las operaciones sobre las curvas definidas sobre cuerpos binarios de extensión GF(2^{163}) hacen falta 163 bits, seis palabras de 32 bits. Normalmente uno o dos polinomios de entrada y uno de salida. Para los cuerpos binarios de extensión GF(2^{233}) serán necesarios 8 palabras de 32 bits por polinomio.

3.2 Multiplicador de polinomios

Los elementos del cuerpo GF(2^m) se han representado en base polinomial. Se podrían haber representado en otras bases como la normal o las bases dual o triangular, pero la polinomial es la más común. Así los elementos se representan como polinomios de hasta grado m-1. Las operaciones de suma de polinomios y resta se implementan mediante operaciones XOR muy rápidas, mientras que la multiplicación es considerada la operación más compleja e importante. Las operaciones se realizan módulo f(x), siendo f un polinomio irreducible de grado m de la siguiente forma:

$$f(x) = x^m + f_m - 1x^{m-1} + \ldots + f_1x + f_0$$

con $f_i \in$ GF(2) = {0, 1}. Siendo el conjunto {1, x, . . . x^{m-1}} la baso polinomial en GF(2^m) con la que podemos representar cualquier elemento. La multiplicación de dos elementos a(x) y b(x) en GF(2^m) se define como c(x) = a(x)b(x) mod f(x). Esta operación implica dos pasos, la multiplicación y la reducción módulo f(x). Los polinomios irreducibles utilizados son los recomendados por el NIST para curvas no-supersingulares[11, 12]:

$$f(x) = x^{163} + x^7 + x^6 + x^3 + 1 \text{ en } GF(2^{163}) \quad y \quad f(x) = x^{233} + x^{74} + 1 \text{ en } GF(2^{233}).$$

Entre los multiplicadores se contemplaron distintas opciones. El multiplicador combinacional clásico, el Kartsuba-Ofman que utiliza un método recursivo muy eficiente, así como el multiplicador de Mastrotivio basado en operaciones matriz-vector [14]. O el multiplicador de Montgomery que permite versiones tanto combinacionales como secuenciales. El que se ha escogido finalmente es el Multiplicador entrelazado de tipo secuencial y muy sencillo. Más lento que las opciones combinacionales (Classic y Mastrovito) pero con un coste y un área mucho menor. En la tabla pueden verse las comparaciones entre las diferentes implementaciones, tanto en tamaño como en eficiencia [14].

M	Type	FFs	LUTs	Slices	Period	Cycles	Total time
163	Classic	–	22,356	15,171	–	–	39
	Interleaved	509	511	271	4.5	163	815
	Mastrovito	–	22,347	15,201	–	–	36
	Montgomery	344	347	184	7.4	163	1,206
233	Interleaved	763	723	417	6.4	223	1427
	Montgomery	484	489	255	7.5	233	1,748

Tabla 3. Implementación sobre FPGA de multiplicación en GF(2^M)

De las dos opciones de implementación que permite el multiplicador entrelazado: Bit Más Significativo Primero (MSB-First) o el Menos significativo primero (LSB-first), se ha escogido la segunda porque presenta menos dependencias en las operaciones a iterar y por tanto un camino crítico más corto en la ruta de datos [14]. La implementación del componente se ha tomado de [14] y presenta la siguiente interfaz:

```
entity interleaved_mult is
port (
A, B: in std_logic_vector (M-1 downto 0);
clk, reset, start: in std_logic;
Z: out std_logic_vector (M-1 downto 0);
done: outstd_logic
);
end interleaved_mult;
```

3.2.1 Interfaz AXI del Multiplicador

Al componente selecicionado se le han añadido los registros, y señales necesarios para implementar su interfaz AXI. El componente va a tener un bus de escritura de 32 bits y otro de lectura de 32 bits. El bus de direccionamiento de 7 bits servirá para poder direccionar cada uno de los registros del componente (figura 3)

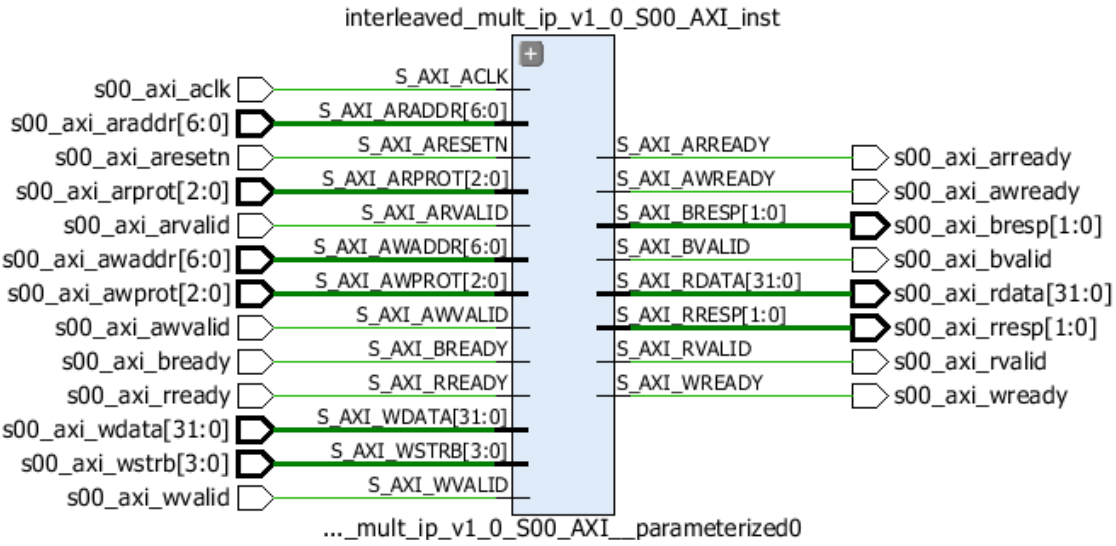

Fig. 4. Interfaz AXI del multiplicador

El funcionamiento de la comunicación es la siguiente. En primer lugar se pondrá en el bus la dirección del registro a escribir (*AWADDR*). En segundo lugar se pondrá el dato en el bus de datos de escritura (*WDATA*) y a continuación se usará la señal de *WSTRB* para realizar el *WRITESTROBE* (la escritura consecutiva de un 0 y un 1) para iniciar la escritura del dato. De forma parecida se utilizan los buses de datos (*RDATA*) y dirección (*ARADDR*) para la lectura.

Internamente se controla que las escrituras sólo se habiliten (*WRITE_ENABLE*) cuando los datos, la dirección se hayan establecido y sean válidas y se termine de hacer el WSTRB.

En la figura 4 puede verse una sección de la conexión entre el componente seleccionado (multiplicador entrelazado de polinomios de 163 bits) y los registros del dispositivo o controlador:

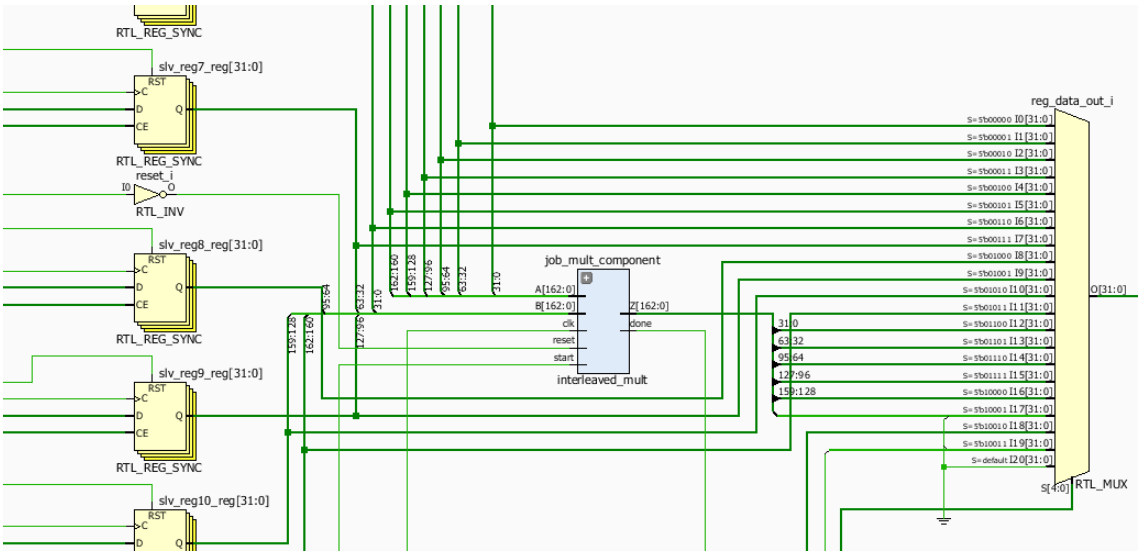

Fig. 5. Sección del esquema RTL del multiplicador

Se han desarrollado por último las funciones software para que el procesador pueda leer y escribir de los registros del dispositivo como en los de cualquier otro periférico

void INTERLEAVED_MULT_IP_mWriteReg(u32 BaseAddress, unsigned RegOffset, u32 Data)

u32 INTERLEAVED_MULT_IP_mReadReg(u32 BaseAddress, unsigned RegOffset)

Con *BaseAddrress* la dirección base (mapeada en memoria) donde se encuentra nuestro multiplicador. *RegOffset* es el desplazamiento del registro que queremos leer o escribir y los datos que se pasan como parámetro en caso de escritur o se devuelven al leerse del registro especificado.

De la misma manera se ha implementado la función software que sustituye a la función software *Product_Mod_F* (identificada en la sección 2.4 de profiling) en la que los cálculos se sustituyen por la escritura de los operadnos y la lectura del resultado en los registros correspondientes del acelerador hardware una vez que se detecte (leyendo un resgitro de ocntrol) que los cálculos han terminado.

Polynomial Interleaved_Mul_Mod_F(Polynomial A, Polynomial B, Polynomial C)

3.2.2 Unidad de control del Multiplicador

Al componente se le ha añadido unidad de control que implementa una máquina estados de tipo Moore muy sencilla que puede verse en la figura 6.

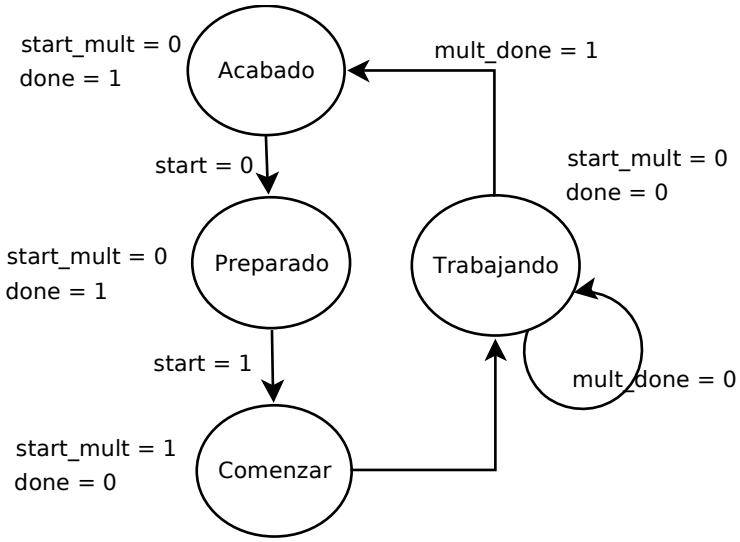

Fig.6. Máquina de estados del multiplicador

Las señales de *start* y *done* se transmiten igualmente por el interfaz AXI entre el procesador principal y el componente. Para empezar a trabajar el componente cliente debe mandar consecutivamente las ordenes de start = 0 y start= 1 (clásica técnica de *strobe* o palanca) pasando por dos estados en los cuales se resetea el componente y se le da la orden de comienzo. Finalmente entramos en un estado de espera mientras se calcula la multiplicación antes de pasar al estado acabado en el que el componente cliente obtiene la salida done = 1 indicando que ya puede leer el resultado.

3.3 Cuadrado de polinomios

Una manera directa de implementar la operación de elevar al cuadrado modular sería utilizar un multiplicador como los del apartado anterior con un solo operando de entrada para poder realizar $c(x) = a(x)a(x) \bmod f(x) = a(x)^2 \bmod f(x)$. Pero la operación de elevar al cuadrado puede optimizarse ya que en GF(2^m) se trata de una operación lineal, es decir,

$$c(x) = a(x)^2 \bmod f(x) = (a_{m-1}x^{2(m-1)} + a_{m-2}x^{2(m-2)} + \ldots + a_1 x^2 + a_0) \bmod f(x)$$

Por tanto, la primera parte de la operación, la de multiplicación (que va seguida de la posterior reducción modular) puede calcularse como $d(x) = a(x)a(x)$, siendo $d(x)$ un polinomio de grado $2m - 2$ de la forma

$$d(x) = a_{m-1}x^{2(m-1)} + a_{m-2}x^{2(m-2)} + \ldots + a_1 x^2 + a_0 = (a_{m-1}, 0, a_{m-2}, 0, \ldots, 0, a_1, 0, a_0)$$

Se han comparado distintas opciones de implementación. En la tabla 4 puede verse que las opciones combinacionales son más rápidas y ocupan menos espacio que sus contrapartidas secuenciales, así que se ha optado por las primeras.

Type	m	Ffs	LUTs	Slices	Period	Cycles	Totaltime
LSB-First	163	464	510	306	5.3	82	435
	233	659	723	436	6.0	117	702

Seq. Montgomery	163	361	341	199	4.3	163	701
	233	542	484	309	4.4	233	1,025
Classic	163	-	165	86	-	-	3
	233	-	153	99	-	-	3
Montgomery	163	-	267	147	-	-	20
	233	-	117	74	-	-	7

Tabla 4. Implementaciones sobre FPGA de squarer en GF(2^m)

En concreto hemos optado por el método clásico [14] que es además muy sencillo de implementar y presenta la siguiente interfaz:

```
entity classic_squarer is
port (
a: in std_logic_vector(M-1 downto 0);
c: out std_logic_vector(M-1 downto 0)
);
end classic_squarer;
```

En este caso no ha sido necesaria la adicción de una unidad de control al tratarse de un circuito combinacional y solamente se ha añadido el interfaz AXI correspondiente. De la misma manera al definido en la sección 3.2.1 del multiplicador.

3.4 Divisor de polinomios

La división sobre un conjunto de polinomios, más concretamente sobre la estructura de anillo de polinomios $Z_p[x]/f(x)$ con p primo y con $f(x)$ polinomio de grado $m > 0$ solo es posible si$f(x)$ es irreducible (es decir que no tiene factores de grado mayor o igual a uno). De esta manera todo $h(x)$ perteneciente a $Z_p[x]/f(x)$ tiene un inverso $h(x)^{-1}$tal que $h(x)h^{-1}(x) \bmod f(x) = 1$. En este caso la estructura de anillo se pasa a denominarse cuerpo finito. $Z_p[x]/f(x)$ pasa a ser un cuerpo extendido del cuerpo finito Z_p, también denominada *Galois Field o GF(p^m)*. En cuerpos binarios p = 2. La operación bajo estudio es la siguiente: dados $g(x)$ y $h(x)$ en$Z_p[x]/f(x)$, donde $h(x)$ es un polinomio no nulo, calcular $z(x)$ tal que $g(x) = h(x)z(x) \bmod f(x)$, es decir,

$$z(x) = g(x)h^{-1}(x) \bmod f(x)$$

Hay dos tipos de algoritmos como en el caso de Z_p. El primer tipo de algoritmos son los que permiten representar el mcd (máximo común divisor) de dos polinomios a(x) y b(x) sobre Z_p en la forma α(x)a(x) + β(x)b(x) (gracias a la Identidad de Bezout) con α (x) y β (x) polinomios sobre Z_p. En este caso para hallar $h^{-1}(x)$, con $f(x)$ irreducible y grado de $h(x)$ menor a m (grado del irreducible) tenemos que su mcd = 1 y, aplicando Bezout, existen dos polinomios $\alpha(x)$ y $\beta(x)$ tales que $\alpha(x)f(x) + \beta(x)h(x) = 1$ y $\beta(x)h(x) \bmod f(x) = 1$, es decir

$$h^{-1}(x) = \beta(x) \bmod f(x)$$

A este primer grupo pertenecen el Algoritmo de Euclides Extendido y el Algoritmo Binario (elegido por nosotros). Un segundo método consistiría en sustituir la inversión por la exponenciación basándonos en la idea del carácter cíclico del cuerpo, que no hemos contemplado en este trabajo.

Nuestro trabajo ha sido el elegir el algoritmo de división. Para ello se han comparado los principales algoritmos comentados, tanto en área y recursos como en rendimiento (Ver [14])

Algoritmo	Ffs	LUTs	Slices	Mult	RAM	Periodo	Ciclos	Tiempo
PseudoEuclidean	871	3,923	2,272	39	1	36	147	5,292
Binary	623	3,235	2,001	37	–	56	37	2,072
Reduction to multiplications (MSE)	562	2,607	1,594	34	1	25	7,602	190,05
Reduction to multiplications (LSE)	672	2,794	1,754	35	1	19	4,202	79,838
Optimalextensionfield (MSE)	603	2,873	1,609	34	1	25	235	5,875
Optimalextensionfield (LSE)	715	3,268	1,894	35	1	19	133	2,527

Tabla 5. Implementaciones sobre FPGA de la división en GF(2^m)

Esta tabla se ha extraído de [14] donde se comparan distintas implementaciones de Divisores sobre GF(239^{17}) implementados en una FPGA Spartan3 (speed-5) de Xilinx. Los tiempos están expresados en ns, los FFs, LUTs, Mults y RAM, representan el número de flip-flops, el número de tablas Look-Up , el número de multiplicadores de 18 bit-by-18-bit y el número de bloques RAM usados en la implementación. Hemos escogido el método binario ya que es el único que no usa RAM y que proporciona el mejor rendimiento con un coste o tamaño muy aceptable.

La implementación se ha tomado de [14]. Al componente se le ha añadido un interfaz AXI parecido al detallado en la sección 3.2.1 junto con una máquina de estados, igualmente muy parecida a la del Multiplicador de la sección 3.2.2.

4. Resultados

Durante el desarrollo se verificó la corrección de cada una de las versiones, comparando los resultados obtenidos fueran exactos a los resultados publicados por el NIST[11 y 12] así como implementaciones de otros autores [14].

Para la evaluación del consumo utilizamos como instrumento de medida un vatímetro Yokogawa WT210 [27]. El vatímetro viene acompañado de un software que procesa el muestreo de las medidas y ofrece gráficas de una gran variedad de magnitudes de medida. En la tabla 6 se puede observar que el consumo medio de las 2 versiones es muy parecido, la

versión software y la versión completa con los tres aceleradores hardware (Multiplicador y Elevador al cuadrado y el divisor).

m	Algoritmo Mutiplicación puntos ECC	Potencia (W)
163	Montgomery Lopez-Dahab (Versión Software)	4,475
233		4,357
163	Montgomery Lopez-Dahab (3ops)	4,340
233		4,395

Tabla 6. Comparación del consumo de potencia en los 3 desarrollos

Los consumos llegan a ser algo inferiores en la versión hardware, suponemos que por el menor uso del procesador software, que puede quedar suspendido en espera de la finalización de las operaciones hardware. El principal ahorro energético vendrá por la reducción de los tiempos de ejecución que podemos ver más abajo. En la tabla 7 aparecen los recursos utilizados por las distintas versiones de código desarrolladlas, con 2 y 3 operaciones, siendo el divisor la tercera operación, añadida en la segunda versión. En estos desarrollos hay que tener en cuenta que se deben incluir también los componentes del interfaz AXI. Los FFs, LUTs, representan el número de flip-flops y el número de tablas Look-Up empleadas.

Desarrollo (Versión)	Recurso	Utilizados
M = 163 (2ops)	FF	1902
	LUT	2008
M = 163 (3ops)	FF	3041
	LUT	3020
M = 233(2ops)	FF	2318
	LUT	2393
M = 233 (3ops)	FF	3877
	LUT	3724
[17]	FF	913
	LUT	2028

Tabla 7. Recursos de implementación de las operaciones hardware

En [21] se proporcionan el número de Slices utilizados, para su implementación en la plataforma XC3S400 utilizan 2418 Slices y en la XC4VFX12 utilizan 2648 Slices. Nuestra frecuencia de trabajo para realizar las mediciones ha sido de 100MHz. Las utilizadas en [17 y 16] son de 68.3 MHz y las de [17] estaban en 79.6 y 142.5 MHz. Los trabajos [2, 3 y 4] son anteriores y trabajan con rutas de datos con una anchura de 8 bits presentando rendimientos inferiores.

Las primeras mediciones se han realizado para comprobar si efectivamente las nuevas operaciones hardware aceleraban la ejecución del algoritmo implementado en software. En efecto, como puede verse en la tabla 8, las aceleraciones con las dos versiones desarrolladas son altísimas. Esto es debido a que la aceleración se produce sobre las funciones que consumen casi la totalidad del tiempo de ejecución.

m	Algoritmo Mutiplicación puntos ECC	Rendimiento [S]	Aceleración
163	Montgomery Lopez-Dahab (Versión Software)	28,398	1
233		70,147	1
163	Montgomery Lopez-Dahab (2ops)	0,0733	387,4
233		0,1455	482,1
163	Montgomery Lopez-Dahab (3ops)	0,0091	3120,6
233		0,0141	4974,9

Tabla 8. Comparación de resultados de las distintas versiones desarrolladas

Trabajo	m	Algoritmo Mutiplicación puntos ECC	Plataforma	Rendimiento [S]	Escalable
[3]	163	Double-Add	Dalton 8051 ISS-8bit	3.97	No
[4]	163	Montgomery Lopez-Dahab	AVR-AT94K - 8 bit	0.113	
[2]	163		AVR – 8 bit	0.290	
	233		ATmega128 - 8bit	0.810	
[16]	163	MixedCoordinates	PicoBlaze 32bit	0.0155	Sí
	283			0.0451	
[17]	163	Montgomery Lopez-Dahab	PicoBlaze 32bit 68.3MHz	0.038	Sí
	233			0.0734	
[21]	163	Montgomery Lopez-Dahab	XC3S400 32bit79.637MHz	0.864 ms	Sí
	283			1.957 ms	
	163		XC4VFX12 32bit142.53MHz	0.483 ms	Sí
	233			1.093 ms	
Este Trabajo	163	Montgomery Lopez-Dahab (2ops)	ZedBoard 32bit 100MHz	0.073	Sí
	233			0.145	
	163	Montgomery Lopez-Dahab (3ops)		0.009	
	233			0.014	
	163	Koblitz (2ops)		3.7241	
	233			7.7863	
	163	Koblitz (3ops)		0.0707	
	233			0.1202	

Tabla 9. Comparación de resultados con otras arquitecturas

Como se puede ver en la tabla 9 los tiempos conseguidos en este trabajo son algo mejores que los obtenidos en las aproximaciones de codiseño [17 y 16], sobre todo si comparamos con nuestra versión de 3 operaciones aritméticas (multiplicación, cuadrado y división). La arquitectura desarrollada en [21] claramente supera a nuestro desarrollo, siendo una de las implementaciones (sobre FPGA) de multiplicación de puntos más rápida, aunque todavía por debajo de [18].Pero hay que recordar que [21] presenta una solución totalmente implementadas en HW, lo que implica más recursos, y más tiempo de desarrollo. Nuestro trabajo no pretende competir con soluciones de este tipo sino evaluar la utilidad del codiseño en la que la mayor parte de la funcionalidad puede ir en SW acelerando el tiempo de desarrollo. Una ventaja adicional del trabajo desarrollado es que presentamos una

arquitectura escalable (al igual que en [21]). Con los recursos necesarios podría trabajarse en cualquier tamaño y no sólo en los dos tamaños presentados. Comparando nuestros desarrollos, vemos que nuestra versión Koblitz, que hace un mayor uso de la división tiene peor rendimiento, desventaja muy acusada en las versiones con sólo dos operaciones.

5. Conclusiones

5.1 Resumen del trabajo realizado

El objetivo de este trabajo es estudiar las posibilidades de codiseño que ofrecen los nuevos chips con procesadores ARM y FPGA. El estudio se ha centrado en criptografía con curvas elípticas, un campo especialmente interesante para sistemas móviles y empotrados en el que existe interés por la utilización de FPGAs desde hace tiempo. En este trabajo se ha implementado un algoritmo puramente en software para la multiplicación de puntos en curvas elípticas. Se ha analizado el software y se han sustituido las operaciones más costosas (multiplicación, elevar al cuadrado y división) por aceleradores hardware.

Se han conseguido aceleraciones muy importantes con aceleradores genéricos y fácilmente reutilizables, manteniendo los niveles de consumo, con lo que el ahorro energético se multiplica. Siendo un aspecto crítico en los dispositivos embebidos y con grandes restricciones.

5.2 Conclusiones sobre el trabajo desarrollado

Con los resultados obtenidos parece evidente que las nuevas arquitecturas heterogéneas ARM/FPGA son una oportunidad para conseguir lo mejor de dos mundos que hasta ahora eran independientes. Por un lado el diseñador tiene a su disposición los entornos de desarrollo para ARM con herramientas de compilación, depuración, profiling… así como la posibilidad de usar todo tipo de librerías desarrolladas para estos sistemas. Por otro lado las FPGAs permiten desarrollar aceleradores a medida, que además se pueden cambiar en tiempo de ejecución [25] de forma que los mismos recursos que se utilizan para acelerar la computación con curvas elípticas se pueden usar para otras aplicaciones en otro momento. Además las FPGAs de Xilinx y en concreto la plataforma ZedBoard son muy eficientes para la implementación de operaciones aritméticas, gracias a recursos especiales que incorporan como los DSP48E1[10].

Los entornos de desarrollo utilizados permiten acoplar de forma sencilla la parte software y los aceleradores hardware. Y los resultados muestran que en el caso de las curvas elípticas, la aceleración obtenida al incluir recursos hardware compensa el sobrecoste de las comunicaciones. Es importante remarcar que una vez diseñado un acelerador e integrado en el sistema, el software puede utilizarlo de la misma manera con la que interactúa con cualquier otro periférico haciendo llamadas a una serie de funciones definidas en una librería. Es decir,

no hace falta conocimientos de diseño hardware para utilizar un acelerador hardware ya diseñado. Por tanto los aceleradores podrán incluirse en librerías que permitiesen a cualquier desarrollador software acelerar sus diseños. Por ello se ha tratado de hacer diseños genéricos y reutilizables.

Lógicamente, incluir nuevos recursos heterogéneos en un chip como una FPGA no sólo conlleva ventajas. También aumenta la complejidad del diseño, y las necesidades de test y verificación. En ese sentido es interesante comentar que existe una comunidad muy activa y bastante documentación. Aun así al ser plataformas recientes a veces se encuentran problemas no muy bien documentados. En general la complejidad del proceso de diseño de un acelerador en estas plataformas y su inclusión como periférico en el sistema no me ha resultado excesivamente complejo. Las herramientas proporcionadas por el fabricante tanto para el diseño hardware como software nos han parecido muy potentes, flexibles y sencillas de utilizar. En cuanto al proceso de test y verificación se han encontrado dificultades en la selección de valores escalares suficientemente grandes y representativos, así que finalmente se tuvo que pasar a una representación vectorial del entero con el que multiplicar los puntos generadores de la curva. Además tampoco hay publicados muchos trabajos que incluyan resultados para poder comparar. Ha sido un proceso complicado, pero finalmente nos permite estar seguros de nuestras mediciones.

5.3 Grado de consecución de objetivos

Los objetivos iniciales se han cumplido en su totalidad. Se han aplicado conocimientos adquiridos tanto en las asignaturas del máster de Programación orientada a prestaciones para realizar el *profiling* como de la asignatura de Procesadores de dominios específicos.

El trabajo desarrollado puede representar otra vía de investigación en el área. Un caso de uso distinto dentro del codiseño hardware software que se realiza en el grupo, pero en el que se pueden incorporar los avances genéricos como la mejora de las comunicaciones en chip y aprovechar toda la experiencia acumulada.

En lo personal, ha supuesto un reto simultanear los estudios del máster con la vida profesional. Y ha quedado demostrado que el no trabajar de manera continuada en el tiempo en un proyecto es altamente perjudicial para su finalización. El autor quiere agradecer la colaboración de Javier Olivito en el proceso de toma de mediciones y en especial el apoyo y la ayuda del director del trabajo, Javier Resano, a lo largo de todo el desarrollo.

6. Trabajo futuro

El siguiente paso sería el desarrollo de operaciones hardware de grano más grueso o más complejas, como pueden ser las de Suma de Puntos o Doblado de Puntos de las curvas. Esta idea ha quedado a medio desarrollar y no ha podido finalizarse a fecha de hoy. El objetivo era sondear si la pérdida de generalidad merecía la pena en términos de coste y eficiencia.

Al incrementar la complejidad de las operaciones que se indica, se aumenta la cantidad de polinomios a transmitir y se podrían aplicar las técnicas de transferencia con el interfaz AXI DMA que ha sido optimizado por otros miembros del grupo de investigación, para aumentar las transmisiones.

Finamente, también ha quedado por concluir la escalabilidad de nuestro desarrollo (solo finalizado para curvas de tamaño m = 163 y m = 233) que debería extenderse a la totalidad de las curvas recomendadas por el NIST [11].

7. Referencias

[1]ARM, "The ARM Cortex-A9 Processors", White paper, v2.0, September 2009. Available: http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf

[2]H.Eberle, A.Wander, N. Gura, S.C. Shantz, "Architectural Extensions forElliptic Curve Cryptography over GF(2m)", Sun Microsystems Laboratories.2005.

[3] L. Batina, D. Hwang, A. Hodjat, K. Sakiyama, I. Verbauwhede,"Reconfigurable architectures for curve-based cryptography on embeddedmicro-controllers", International conference on Field Programmable Logic andApplications. FPL. 2006.

[4]S. S. Kumar, C. Paar, "Reconfigurable instruction set extension for enablingECC on an 8-bit processor ", In Field Programmable Logic and Application -FPL 2004, LNCS 3203, pp. 586–595. Springer Verlag, 2004.

[5]M. Santarini, "Xilinx Redefines State of the Art With New 7 Series FPGAs", Xcell Journal, Third Quarter 2010, pp. 6 - 11.Disponible: http://www.xilinx.com/publications/archives/xcell/Xcell72.pdf

[6] Certicom Research, SEC 2: Recommended Elliptic Curve Domain Parameters, v1.0, 2000

[7]Xilinx, Inc., "7 Series FPGAs Overview", Product Specification, DS180, v.1.15, February 2014.Disponible: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[8]Xilinx, Inc., "AXI Reference Guide", UG761, v14.3, November 2012.Disponible: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

[9] Olivito, J.; González, C.; Resano, J. (2010) FPGA Implementation of a Strong Reversi Player. Proceedings IEEE International Conference on Field-Progammable Technology (FPT 2010), pag: 507-510.

[10]Xilinx, Inc., "7 Series DSP48E1 Slice User Guide", UG479, v1.7, May 2014. Available: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[11] Recommended Elliptic Curves for Federal Government Use. http://csrc.nist.gov/.

[12] U.S. Department of Commerce/National Institute of Standards and Technology (NIST),Digital Signature Standard (DSS), FIPS PUB 182-2changel, 2000.

[13] D. Hankerson, A. Menezes, and S. Vanstone. Guide to Elliptic CurveCryptography. Springer, New York, 2004.

[14] J.P. Deschamps, J.L. Imaña, and G.D. Sutter.HardwareImplementationof Finite-FieldArithmetic.McGraw-Hill, 2009.

[15] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processorfor GF(2^m). In Ç. K. Koç and C. Paar, editors, Cryptographic Hardwareand Embedded Systems — CHES 2000, LNCS 1965, pages 41 –56.Springer-Verlag,2000.

[16] M. Hassan and M. Benaissa, "Low Area - Scalable Hardware/Software Co-design for Elliptic Curve Cryptography for Low-Resource Applications" in 3rd International Con-ference on New Technologies, Mobility and Security (NTMS), December2009

[17] M. Hassan and M. Benaissa, "Low Area - Scalable Hardware/Software Co-design for Elliptic Curve Cryptography on PicoBlaze Microcontroller" in 3rd International Con-ference on New Technologies, Mobility and Security (NTMS), December2009

[18] N. Gura, S. Chang, H. Eberle, G. Sumit, V. Gupta, D. Finchelstein, E. Goupy, andD. Stebila. An End-to-End Systems Approach to Elliptic Curve Cryptography. InÇ. K. Koç and C. Paar, editors, Cryptographic Hardware and Embedded Systems —CHES 2001, LNCS 1965, pages 351–366. Springer-Verlag, 2001.

[19] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An implementation of ellipticcurve cryptosystems over $F2^{155}$. IEEE Journal on Selected areas in Communications,11(5):804–813, June 1993.

[20] S. Okada, N. Torii, K. Itoh, and M. Takenaka. Implementation of Elliptic CurveCryptographic Coprocessor over GF(2^m) on an FPGA. In Çetin K. Koç andChristof Paar, editors, Proceedings of the Second Workshop on Cryptographic Hardwareand Embedded Systems — CHES 2000, pages 25–52, Berlin, Germany, 2000.Springer-Verlag

[21] K. C. CinnatiLoi and Seok-Bum Ko. High Performance Scalable Elliptic CurveCryptosystem Processor in GF(2^m). In Circuits and Systems (ISCAS), 2013 IEEE International Symposium

[22] N. Koblitz. Elliptic Curve Cryptosystems.Mathematics of Computation, 48:203–209, 1987

[23] V. Miller. Uses of Elliptic Curves in Cryptography. In H. C. Williams, editor, Advancesin Cryptology — CRYPTO '85, LNCS 218, pages 417–426, Berlin, Germany,1986. Springer-Verlag.

[24] T. ElGamal. A public-key cryptosystem and a signature scheme based on discretelogarithms. IEEE Transactions on Information Theory, IT-31(4):469–472, 1985.

[25] Xilinx, Inc., "Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices", v1.0 2013. Disponible: http://www.xilinx.com/support/documentation/application_notes/xapp1159-partial-reconfig-hw-accelerator-zynq-7000.pdf

[26] GNU gprof. Disponible: http://sourceware.org/binutils/docs-2.18/gprof/index.html

[27] Yokogawa Electric Corporation Website, http://www.yokogawa.com/

8. Anexos. Conceptos y base matemática

En este capítulo se ofrece una breve introducción a la formación matemática en Criptosistemas Curva Elíptica (ECC del inglés, en adelante). Sólo damos una breve introducción que cubre todos los aspectos que son relevantes para esta tesis. Las partes 2.1 y 2.2 se han tomado principalmente de [2]. Para una introducción más detallada a ECC recomendamos la siguiente bibliografía [1, 3 y 4]

8.1 Introducción a cuerpos finitos

Un cuerpo finito consiste en un conjunto finito de elementos F, dos operaciones binarias, adición y multiplicación y los inversos multiplicativos y aditivo de cada elemento. Las operaciones binarias satisfacen ciertas propiedades aritméticas. El número de elementos en el cuerpo se denomina *orden*. Existe un cuerpo finito de orden *q* si y sólo si q es una potencia de un primo. Esencialmente, sólo hay un cuerpo finito de orden q denotado por F_q. Si $q = p^m$ donde *p* es un número primo y *m* es un entero positivo, entonces *p* se llama *característica* de F_q y *m* se llama el grado de extensión de F_q. Los cuerpos finitos también se denominan Cuerpos de Galois en honor a Évariste Galois (1811-1832) o GF de sus siglas en inglés.

En lo que sigue, se describen brevemente los dos tipos más importantes de cuerpos finitos aplicados en la práctica, el cuerpo primo GF(p) y el cuerpo binario $GF(2^m)$.

8.1.1 El Cuerpo finito F_p o GF(p)

Llamamos cuerpo primo al cuerpo finito F_p donde *p* es un número primo. Se representa por la conjunto de números enteros {0, 1, 2,…, p – 1}. Las operaciones de adición y de multiplicación son módulo *p*. Si a es un elemento distinto de cero en F_p, decimos que el inverso de a módulo *p*, denotado por a^{-1}, es el único entero *c* perteneciente a F_q para el que $a \cdot c = 1$. En esta tesis no utilizamos el cuerpo finito F_p. Sólo usamos el cuerpo finito $GF(2^m)$, que se presenta a continuación.

8.1.2 El Cuerpo finito $GF(2^m)$

El cuerpo finito GF(2^m) puede ser visto como un espacio vectorial de dimensión m sobre el cuerpo F_2 que consta de dos elementos 0 y 1. A GF(2^m) se le conoce a menudo como cuerpo finito de *característica* dos o cuerpo finito binario. La característica es el menor número de veces que debes sumar 1 para obtener 0 (suma en el cuerpo finito). Se denomina *orden* (*q*) del cuerpo finito al número de elementos que contiene. Con $q = 2^m$en GF(2^m). Como se trata de un espacio vectorial, todos los elementos *a* de GF(2^m) pueden ser representados como una cadena de bits (a_0a_1… a_{m-1}): a =$a_0\beta_0+a_1\beta_1$ +…+$a_{m-1}\beta_{m-1}$; donde $a_i \in$ GF(2) = {0, 1}. El conjunto {β_0 β_1… β_{m-1}} se llama una base de GF(2^m) sobre GF(2). Hay muchas bases diferentes y algunas de ellos conducen a implementaciones más eficientes que otras. En esta tesis, sólo tenemos en cuenta las representaciones sobre base polinómicas, porque son muy adecuadas para microprocesadores y arquitecturas hardware. Otras bases se describen, por ejemplo, en [5], que es también nuestra principal referencia para esta sección. Un polinomio f(x) irreducible de grado m sobre F_2 o GF(2) se puede escribir como:

$$f(x) = x^m + f_m - 1x^{m-1} + \ldots + f_1x + f_0$$

con $f_i \in$ GF(2) = {0, 1}. Siendo el conjunto {1, x, . . . x^{m-1}} la base polinomial en GF(2^m). Irreducible significa que no puede ser factorizado en polinomios de grado menor que m (y mayor o igual a 1). La identidad multiplicativa se representa por el polinomio constante (grado 0) igual a 1. Y la identidad aditiva por el polinomio nulo (todos los coeficientes iguales a 0). La suma (y la resta) se implementan con operaciones XOR bit a bit. La multiplicación se obtiene tras multiplicar los coeficientes de los dos polinomios de entrada aplicando la propiedad distributiva, obteniendo un polinomio de grado hasta 2(m-1) y luego reduciendo posteriormente por el polinomio irreducible (dividiendo y obteniendo su resto).

8.2 Introducción a las curvas elípticas sobre cuerpos finitos

Dado un cuerpo finito K, una curva elíptica E se define sobre K por la ecuación de Weierstrass

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

con a_1, a_2, a_3, a_4, y a_6 pertenecientes a K y satisfaciendo algunas condiciones adicionales establecidas sobre el discriminante de la ecuación [2, Cap. 3]. El objetivo de estas condiciones es definir una ecuación y una curva regular, es decir sin vértices ni intersecciones para que las tangentes sean únicas para todo punto de la curva. Dado un cuerpo de extensión L de K, la curva elíptica correspondiente E(L) se define por la siguiente relación:

$$E(L) = \{(x,y) \in L \times L: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}$$

Siendo ∞ un punto adicional denominado punto en el infinito (identidad de la suma de puntos).

En esta tesis solo trabajamos con cuerpos finitos de característica 2 (p = 2). La expresión de la curva puede simplificarse enormemente mediante un cambio de variables. En nuestro caso la ecuación puede definirse o simplificarse de dos maneras:

a) $y^2 + cy = x^3 + ax + b$ (curva supersingular) a, b, y c ∈K, y c ≠ 0.
b) $y^2 = x^3 + ax^2 + b$ (curva no-supersingular) a y b ∈ K, a ≠ 0 y b ≠ 0.

Puede ser demostrado (Teorema de Hasse, [2]) que el número de puntos de E(L) pertenece al siguiente intervalo:

$$q + 1 - 2q^{1/2} \leq \#E(L) \leq q + 1 + 2q^{1/2}$$

donde q es el número de elementos de L. Así, para grandes valores de q, el número de elementos de E(L) es aproximadamente igual al número de elementos del cuerpo finito: #E(L) ≅ q. En esta tesis, K = Z_p con p = 2 y L = GF(2^m), las más usadas en aplicaciones prácticas.

Definición del Grupo o Ley de Grupo

Sea E una curva elíptica definida sobre L. Hay una regla de *arco-y-tangente* para sumar dos puntos en E(L) para dar un tercer punto en E(L). Junto con esta operación de suma, el conjunto de puntos de E(K) forma un grupo abeliano con ∞ como elemento identidad. Es este grupo el que se utiliza en la construcción de sistemas criptográficos de curva elíptica.

La *regla de adición* se explica mejor geométricamente. Sean P= (x_1, y_1) y Q =(x_2,y_2) dos puntos distintos sobre una curva elíptica E. La suma R, de P y Q, se define como sigue. Primero se dibuja una línea que pasa por P y Q. Esta línea intersecta la curva elíptica en un tercer punto. Entonces R será el reflejo o el opuesto de este punto respecto del eje x. La operación puede verse gráficamente en la figura 1 (a).

El *doble de P*, el punto R, se define como sigue. Primero se dibuja la recta tangente a la curva elíptica en P. Esta línea intersecta la curva elíptica en un segundo punto. R será el reflejo o el opuesto de este punto respecto del eje x. Esto se representa en la figura 1 (b).

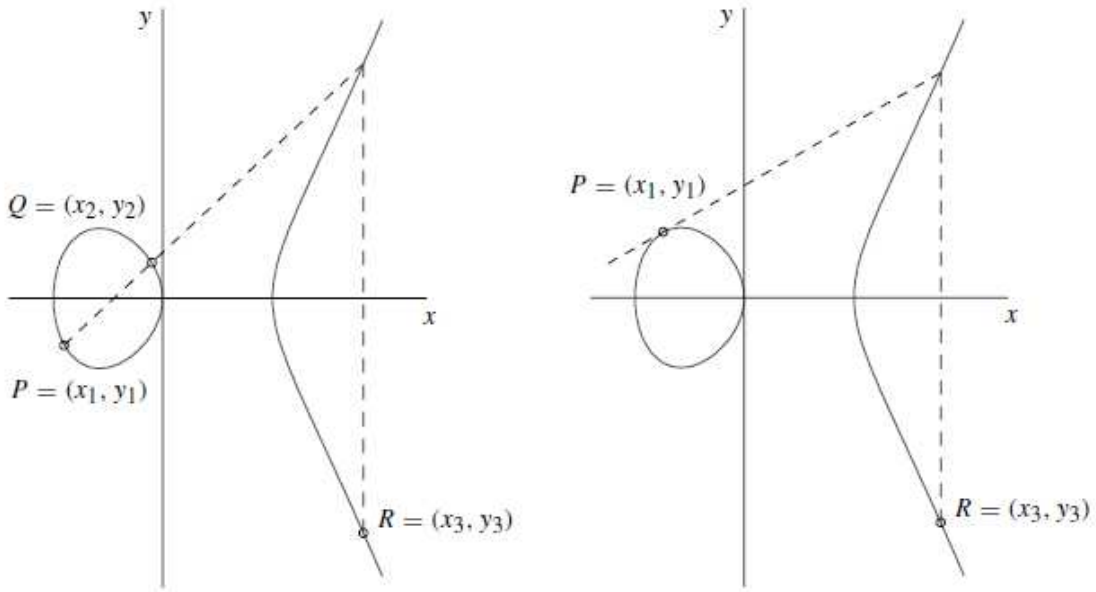

Fig.1: Suma y Duplicado geométricos de puntos de la curva elíptica

Las fórmulas algebraicas para la ley de grupo se pueden derivar de la descripción geométrica. Estas fórmulas se presentan a continuación para curvas elípticas E de la ecuación de Weierstrass simplificada (en coordenadas afines) cuando la característica del cuerpo K subyacente es 2 con curvas elípticas no-supersingulares, es decir L = GF(2^m) = F2^m.

Ley de grupo para E/F2^m no-supersingulares: y2+x y = x3+ax2+b

1. *Identidad*. P + ∞ = ∞ + P = P ∀P ∈E(F2^m).

2. *Negativos*. Si P = (x, y) ∈ E(F2^m), entonces (x, y) + (x, x + y) = ∞. El punto (x, x + y) se denota como −P y es llamado opuesto de P; notar que −P de hecho es un punto en E(F2^m). Además −∞ = ∞.

3. *Suma de Puntos*. Sea P = (x_1, y_1) ∈ E(F2^m) y Q = (x_2, y_2) ∈ E(F2^m), donde P =±Q. Entonces P + Q = (x_3, y_3), donde $x_3 = \lambda^2+\lambda+x_1+x_2 +a$ y $y_3 = \lambda(x_1 +x_3)+x_3 + y_1$ con $\lambda = (y_1 + y_2)/(x_1 +x_2)$.

4. *Duplicado de punto*. Sea P =(x_1, y_1) ∈E(F2^m), donde P =−P (es decir $x_1 \neq 0$). Entonces 2P =(x_3, y_3), donde $x_3 = \lambda^2+\lambda+a = x_1^2+ b / x_1^2$, e $y_3 = x_1^2+\lambda x_3 +x_3$ con $\lambda = x_1+ y_1/x_1$.

Estas son las operaciones implementadas en la tesis. Donde cada coordenada x e y pertenece al cuerpo de extensión de K, es decir son polinomios cuyos coeficientes pertenecen a Z_2.

9. Anexos. Referencias

[1] I. Blake, G. Seroussi, and N. Smart. Elliptic Curves in Cryptography. Cambridge University Press, London Mathematical Society Lecture Notes Series 265, 1999

[2] D. Hankerson, A. Menezes, and S. Vanstone. Guide to Elliptic CurveCryptography. Springer, New York, 2004

[3] J. H. Silverman. The Arithmetic of Elliptic Curves. Springer-Verlag, New York,USA, 1986.

[4] J. H. Silverman and J. Tate.Rational Points on Elliptic Curves.Springer-Verlag,1992.

[5] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve DigitalSignature Algorithm(ECDSA). A Certicom Whitepaper, 2001.http://www.certicom.com/resources/w_papers/w_papers.html.