
**UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE**

Dpto. Física y Arquitectura De Computadores

# "Estudio escalabilidad de la aritmética de cuerpos finitos en hardware reconfigurable y aplicaciones criptográficas"

Memoria de tesis para optar al grado de Doctor

*Federico García Crespí*

Director de tesis:

*Carlos Pastor Antón*

UNIVERSITAS
Miguel
Hernández

UNIVERSITAS Miguel Hernández

Dr. D. *Carlos Pastor Antón* Catedrático de Universidad del Departamento de Física y Arquitectura de Computadores de la Universidad Miguel Hernández de Elche.

CERTIFICA:

Que la tesis titulada "*Estudio escalabilidad de la aritmética de cuerpos finitos en hardware reconfigurable y aplicaciones criptográficas*" de la que es autor Federico García Crespí, ha sido realizada bajo mi dirección.

Y tras valorar el trabajo realizado por el aspirante al Titulo de Doctor,

AUTORIZO

Su presentación y defensa ante el Tribunal correspondiente.

Y para que conste a los efectos oportunos, firmo el presente certificado en Elche a 11 de marzo de 2015.

Fdo: Prof. Carlos Pastor Antón

UNIVERSITAS
Miguel
Hernández

UNIVERSITAS Miguel Hernández

Gabriel Ruíz Ruiz, Director del Departamento de Física y Arquitectura de Computadores Miguel Hernández de Elche.

CERTIFICA:

Que, Federico García Crespí ha realizado bajo la coordinación de este Departamento su memoria de tesis doctoral titulada "*Estudio escalabilidad de la aritmética de cuerpos finitos en hardware reconfigurable y aplicaciones criptográficas*" cumpliendo todos los objetivos previstos, finalizando su trabajo de forma satisfactoria para su defensa pública y capacitándole para optar al grado de Doctor.

Lo que certifico en San Juan de Alicante a 11 de marzo de 2015.

Fdo: Gabriel Ruiz Ruiz

UNIVERSITAS
Miguel
Hernández

UNIVERSITAS
Miguel
Hernández

UNIVERSITAS
Miguel
Hernández

# Índice de tablas

UNIVERSITAS
Miguel
Hernández

# Índice de figuras

UNIVERSITAS
Miguel
Hernández

# Índice

UNIVERSITAS
Miguel
Hernández

# Acrónimos

**RSA** Rivest, Shamir y Adleman, es un sistema criptográfico de clave pública desarrollado en 1977
**AES** Advanced Encryption Standard
**CBC** Cipher-block chaining
**SSL** Secure Socket Layer
**ANSI** American National Standards Institute
**FSF** Free Software Foundation
**NIST** Instituto Nacional de Estándares y Tecnología
**CCE** Criptografía Curvas Elípticas o Criptosistemas de Clave Pública
**PKI** Public Key Infrastructure
**GBB** Ghost Based Bit
**ONB** Optimal Normal Base
**DES** Data Encryption Standard
**EFF** Electronic Frontier Foundation
**NSA** National Security Agency
**NIST** National Institute of Standards and Technology
**MARS** Cifrador en bloque de IBM
**RC6** Algoritmo de cifrado en bloque derivado del RC5
**PFE** Problema de Factorización de Enteros
**PLD** Problema del Logaritmo Discreto
**PLDE** Problema del Logaritmo Discreto en Curvas Elípticas
**DSA** Digital Signature Algorithm
**DHE** Intercambio Diffie-Hellman Elíptico
**ANSI X9** American National Standards Institute X9
**VLSI** Very-large-scale integration

**FPGA** Field Programmable Gate array
**MSB** Most Significant Bit
**LSB** Least Significant Bit
**MSD** Most Significant Digit
**LSD** Least Significant Digit
**ASIC** Application Specific Integrated Circuit
**RISC** Reduced Instruction set Computer
**EDK** Xilinx Embedded Development Kit
**ISA** Instruction Set Architecture
**FSL** Fast Simplex Link
**LMB** Local Memory Bus
**OPB** IBM CoreConnect On-Chip Peripheral Bus $v2{,}0$
**SRAM** Static Random Access Memory
**SDRAM** Synchronous Dynamic Random-Access Memory
**DDR SDRAM** Double Data Rate SDRAM o SDRAM de tasa de datos doble
**MDM** Xilinx Micro Debug Mode
**MAC** Media Access Control
**OCM** On Chip Memory
**FPU** Floating Point Unit
**UART** Universal Asynchronous Receiver-Transmitter
**FIFO** First in, first out
**SoC** System on Chip
**SoPC** System on Programmable Chip
**CSoC** Configurable System on Chip
**MCSoC** Multiprocessor-configurable-System-on-Chip
**ARM** Advanced RISC Machine,
**SCP** Soft-Core Processors
**MMU** Memory Management Unit
**LEON** Mmicroprocesador 32 bits
**SSL** Secure Socket Layer
**IPSEC** Internet Protocol Security
**IDEA** International Data Encryption Algorithm
**TSL** Transport Layer Security
**VHDL** Very High Description Language, Lenguage de descripción hardware

| | |
|---|---|
| **LFSR** | Linear Feedback Shift Register |
| **BRAM** | Block Random Access Memory, bloque de memoria de acceso aleatorio |
| **SLICE** | Bloque de componentes básico en una FPGA |
| **PPC** | Procesador Power PC |
| **LUT** | Look-Up Tables |
| **IOB** | Puerto Salida Xilinx |
| **CPU** | Unidad de Control |
| **DE** | Rotación a la izquierda |
| **TDES** | Triple DES |
| **RW** | Sistema cifrado publico Robin-Williams |

UNIVERSITAS
Miguel
Hernández

# Capítulo 1

# Introducción

Aunque históricamente la técnica dominante para el intercambio de información ha sido la llamada comunicación analógica, durante la ultima parte del siglo 20, su homóloga, la comunicación digital, se ha convertido en el tipo de comunicación predominante. Además, todas las predicciones indican que esta tendencia continuará en el futuro. De hecho, durante los últimos años hemos visto avances significativos en tecnología digital y móvil, tales como la comunicación inalámbrica, una Internet más rápida, etc.. La inmensa mayoría de la información digital producida por estas tecnologías se almacenan y procesan en computadoras. La información digital se transfiere entre computadores vía fibra óptica, satélite y/o Internet. En todos estos nuevos escenarios, la transmisión y almacenamiento de la información de forma segura es de suma importancia en las nuevas infraestructuras emergentes a nivel internacional, como el comercio electrónico y el almacenamiento de la información de forma remota en servidores.

Las técnicas que se usan para el manejo seguro de la información las provee la criptografía, la cual ha sido definida como el estudio de las técnicas que aseguran una comunicación segura en un entorno hostil. Durante siglos, a la criptografía se le había dado un uso militar y diplomático. Sin embargo, durante estos últimos años y debido a los rápidos avances de la tecnología, la investigación relacionada con la criptografía se tiene que enfrentar con nuevos retos, desde el acceso no autorizado a instalaciones privadas, hasta el seguimiento de compras con la tarjeta de crédito. Entre algunas de las aplicaciones más importantes de la criptografía, podemos mencionar el cifrado de datos, uso de dinero electrónico, voto por Internet, autenticación en una red de ordenadores, firma digital y DNI electrónico.

La criptografía es la rama de las matemáticas que trata del cifrado y descifrado

de mensajes. El esquema es sencillo, Alice desea transmitir algo a Bob, sin que lo intercepte Eva. Para ello, usando una clave que ella y Bob han acordado, Alice cifra el mensaje y se lo envía a Bob, quien lo puede descifrar porque conoce la clave. Por su parte, Eva consigue hacerse con una copia del mensaje cifrado, pero sigue teniendo un problema: no tiene la clave. ¿Qué puede hacer Eva para conocer el mensaje? Criptoanálisis. El criptoanálisis es el estudio de métodos para obtener la información de un mensaje cifrado sin conocer la información requerida para descifrarlo (la clave). Estudiando el mensaje cifrado, Eva trata de descifrarlo empleando distintos métodos.

A lo largo de la historia se desarrollaron distintos algoritmos cuyo objetivo era imposibilitar el descifrado de un mensaje sin conocer la clave, paralelamente el criptoanálisis desarrollaba métodos para romper dichos algoritmos, como el análisis de frecuencias, fundamentalmente basados en matemáticas. Con la llegada de los ordenadores, la potencia de cálculo creció exponencialmente y se pudieron emplear algoritmos mucho más complejos para cifrar los mensajes aunque también evolucionaron los métodos de criptoanálisis.

Uno de los sistemas mas utilizados antiguamente era el cifrado de sustitución. Un cifrado por sustitución usa una clave aleatoria tan larga como el texto que se desea cifrar. En 1949, Shannon publicaba su libro "Communication Theory of Secrecy System" [85], en el que demostraba que si la clave era realmente aleatoria, este cifrado era imposible de romper. La razón: el texto cifrado no da ninguna información acerca del contenido del mensaje, Shannon denominó a esta condición "Secreto perfecto".

A pesar de su fuerza, este sistema tiene problemas a la hora de aplicarse: la generación de una clave completamente aleatoria, la distribución de esa clave entre los comunicantes y la necesidad de generar una clave y transmitirla para cada mensaje. Nadie se imagina tener que quedar con un librero para intercambiar claves y así poder comprarle un libro a través de su página web. Para resolver el problema del intercambio de claves, podemos utilizar funciones matemáticas sencillas de evaluar pero para las que la evaluación de su inversa supone un gran problema. Uno de los métodos más empleados es el Protocolo de Diffie-Hellman [24]. A partir de este momento apareció la llamada criptografía de clave pública.

## 1.1 Criptografía de clave pública

Los sistemas de criptografía pública se basan en algoritmos de cifrado asimétricos: la clave para cifrar un mensaje no es la misma que la clave para descifrarlo. La clave para cifrar un mensaje es pública y la clave para descifrarlo, privada.

Si Alice quiere recibir un mensaje seguro de Bob, puede enviarle un cofre en el que Bob pueda guardar lo que quiera, la llave de este cofre es única y sólo la tiene Alice, por lo que aunque Eva intercepte el cofre no puede acceder a su contenido. El cofre seria la clave pública, que Bob utilizaría para cifrar los mensajes, la llave es la clave privada que sólo posee Alice.

Ejemplos de este sistema son ElGamal [27], que está basado en el sistema de intercambio de claves Diffie-Hellman [25] el cual a su vez se basa, para su seguridad, en el Problema del Logaritmo Discreto y el algoritmo Rivest, Shamir y Adleman, sistema criptográfico de clave pública desarrollado en 1977 (RSA) [81] (muy usado en comercio electrónico), que se basa en la factorización de números primos de gran longitud.

Los métodos computacionales que resuelven el problema de la factorización de números grandes o el problema del logaritmo discreto, tienen un tiempo de resolución que crece exponencialmente cuando crece el número de bits de los números utilizados para representar la clave en estos sistemas. Cuanto mayor es el número de bits utilizados, más difícil es de romper el sistema criptográfico. Sin embargo, hay métodos recientes de criptoanalisis que resuelven el problema en un tiempo subexponencial que unido a la rapidez con que los computadores aumentan su capacidad de computo, hacen peligrar a la seguridad de las comunicaciones. La aparición de este método de criptoanálisis, ha hecho necesario utilizar números primos más grandes para proveer el mismo nivel de seguridad, aun a costa de tener que aumentar también la capacidad de computo del sistema que ejecuta el sistema criptográfico, con el consiguiente aumento del coste.

Este era el panorama de la criptografía hasta 1985, cuando Koblitz y Miller [65], propusieron el uso de las curvas elípticas. Las curvas elípticas son una estructura algebraica que se llevan estudiando desde el final del siglo $XIX$ aunque no fue después de 1985, cuando empezaron a utilizarse para desarrollar algoritmos de factorización de enteros, test de primalidad y en sistemas de criptografía de clave publica.

La seguridad de los sistemas criptográficos, tanto públicos como privados dependen de la complejidad en computar una solución al problema matemático en que se basa el sistema. La seguridad de RSA se basa en el problema de factorización de números grandes, mientras que Criptografía Curvas Elípticas o Criptosistemas de Clave Pública (CCE) garantiza su seguridad en la dificultad de resolver el Problema del Logaritmo Discreto en Curvas Elípticas (PLDE) para curvas elípticas. En concreto, la seguridad de los criptosistemas de clave publica construidos con curvas elípticas, se basa en la intratabilidad de encontrar el valor $m$ en la operación de multiplicación $mP$, siendo $P$ un punto de la curva elíptica y $m$ un entero largo. Este problema se denomina PLDE y la operación $mP$, multiplicación escalar en curvas elípticas. No existe ningún método que resuelva el PLDE en un tiempo sub-exponencial, por lo tanto, la CCE se puede utilizar en criptografía de clave pública para asegurar un alto grado de seguridad, con la ventaja de que los CCE, utilizan un menor tamaño de clave.

El problema del logaritmo discreto en este grupo es uno de los mas difíciles de computar y gracias a esta dificultad, se puede obtener la misma seguridad que otros criptosistemas pero utilizando un cuerpo finito más pequeño lo que implica un menor tamaño de clave y un menor tamaño de clave implica menor ancho de banda en las comunicaciones y menos requerimientos de memoria en los computadores.

Como se describe en [81], un criptosistema RSA de 1024 bits proporciona más o menos el mismo nivel de seguridad que un CCE de 200 bits. En los CCE, el grupo sobre el que se construye la curva elíptica, tiene $p^m$ elementos siendo el valor entero positivo $m$, el número de bits de referencia, un número entre 160 y 256.

Los criptosistemas de curvas elípticas tienen a la multiplicación modular en cuerpos finitos como la operación aritmética mas importante. El que un sistema criptográfico tenga un elevado rendimiento o no, depende en gran medida del diseño de las operaciones aritméticas del cuerpo finito subyacente, ya que estas operaciones son las que demandan un mayor número de recursos, computacionalmente hablando.

## 1.2 Planteamiento del problema

Para obtener un mejor rendimiento en la ejecución de las operaciones aritméticas de los CCE, es necesario que estas se implementen en hardware. El uso de software

proporciona mas flexibilidad pero un peor rendimiento. Aun así, las implementaciones hardware presentan varios inconvenientes:

**Reusabilidad. Cambios de algoritmo**

Hoy en día se utilizan diferentes algoritmos de clave publica y en un mundo tan globalizado como el nuestro, para asegurar la compatibilidad de las diferentes aplicaciones criptográficas, estas tienen que soportar todo tipo de algoritmos. Mientras que las implementaciones software son fáciles de actualizar y adaptar a nuevos algoritmos o a cambios en la longitud de la clave, no podemos decir los mismo de las implementaciones hardware. Cuando se construye un criptosistema, hay que tener en cuenta los posibles ataques al criptosistema de manera que podamos cambiar el tamaño de la clave sin cambiar la estructura hardware del criptosistema. Esta habilidad, denominada agilidad algorítmica, es un concepto muy importante en criptografía. No se puede estar completamente seguro de que no exista un método que rompa los actuales criptosistemas. Recientemente, un método propuesto en [12], ha convertido el RSA de 1500 bits en un criptosistema inseguro.

Por tanto, sería interesante construir un criptosistema que nos permita aumentar o disminuir la seguridad sin cambiar la estructura de nuestro diseño hardware.

**Escalabilidad**

Es necesario que el criptosistema propuesto, provea varios niveles de seguridad, por lo que se deberían diseñar los algoritmos para que se adaptasen a cambios en el tamaño de la clave o en el tamaño de palabra del procesador. La escalabilidad proporciona la flexibilidad necesaria a los programadores para poder elegir entre diferentes requerimientos de área ocupada por el diseño hardware, requerimientos de tiempo de computo y de consumo.

El tamaño de los operadores está directamente relacionado con el nivel de seguridad del criptosistema, cuanto más grandes, mas seguro es el criptosistema. Si la seguridad se ve comprometida, se puede aumentar el tamaño de los operadores para de esa manera, aumentar el nivel de seguridad.

En esta situación, lo más favorable sería implementar las operaciones de cifrado de un CCE, en un sistema que proporcione tanto la aceleración del hardware como la flexibilidad del software, utilizando algoritmos escalables. Por tanto, ¿es posible crear una algoritmo que permita efectuar la multiplicación en cuerpos finitos en hardware y de manera escalable?.

## 1.3 Objetivos generales

Esta tesis está comprendida dentro de diversas líneas de investigación que tienen que ver con la seguridad en redes como son:

**Infraestructuras de clave pública** La tecnología Public Key Infrastructure (PKI) permite a los usuarios identificarse frente a otros usuarios, cifrar y descifrar mensajes, firmar digitalmente información, garantizar el no repudio de un envío, y otros usos. Se está desarrollando una PKI propia para potenciar y extender el uso de la firma digital en las comunicaciones digitales, así como un sistema de registro telemático que permita agilizar los trámites incorporando la máxima seguridad.

**Sistemas de comunicación seguros** Se está desarrollando un sistema integral de comunicación (voz, vídeo, datos, etc.) seguro, utilizable sobre cualquier plataforma y sistema de transporte de datos, pudiendo adaptarse a las evoluciones tecnológicas en materia de comunicaciones. Para mayor flexibilidad, el sistema puede funcionar con un servidor centralizado o de forma distribuida mediante tecnologías $P2P$ maximizando la disponibilidad y aplicabilidad del sistema. El sistema resulta muy beneficioso para un amplio espectro de aplicaciones tanto gubernamentales como empresariales: comunicaciones reservadas entre directivos dentro o fuera de la red de la empresa, equipos de desarrollo distribuidos, Fuerzas y Cuerpos de Seguridad del Estado; intranets corporativas, difusión de medios audiovisuales, etc.

**Copias de seguridad remotas** Las copias de seguridad remotas cifradas, garantizan que si alguna vez ocurre un desastre, tanto en la ubicación física de los ordenadores, como dentro de los propios ordenadores, se pueda disponer de los datos de forma inmediata. El sistema desarrollado permite realizar una copia de seguridad de un servidor, una estación de trabajo o un ordenador portátil, utilizando para ello Internet, contra un servidor situado en otra ubicación física. Todos los datos viajan cifrados con la máxima seguridad y puede ser adaptado a cualquier necesidad

operativa.

**Criptografía teórica** Además del uso de estándares criptográficos reconocidos internacionalrnente, el grupo ha desarrollado criptosistemas propios adecuados a ciertas situaciones concretas como un kernel criptográfico para plataformas ligeras o de bajo coste. Otras áreas en las que se posee tecnología propia es en las de códigos correctores y funciones bent.

Este trabajo se enmarca en la línea de investigación sobre criptografía teórica, analizando y diseñando nuevos algoritmos de multiplicación en cuerpos finitos. En primer lugar se hizo un análisis de la literatura relacionada y se detallaron las tareas de investigación encaminadas a conseguir un nuevo algoritmo. Como base teórica para el diseño de un nuevo algoritmo, se han investigado e implementado los diferentes algoritmos de multiplicación en cuerpos finitos y con base en implementaciones tanto hardware como software. Para la implementación de estos algoritmos se han utilizado lenguajes de programación software como el 'C' y lenguajes de descripción hardware para su implementación hardware y se han realizado experimentos de validación y prueba de los algoritmos propuestos.

### 1.3.1 Objetivos específicos

El principal objetivo al empezar a desarrollar esta tesis, fue desarrollar un nuevo algoritmo de multiplicación de cuerpos finitos, que tuviera unas características y un rendimiento mejores que los desarrollados hasta ese momento. Se estudiaron nuevas implementaciones sobre hardware reconfigurable y fruto de esa investigación fue el desarrollo de un algoritmo que utilizaba una representación matricial para los elementos del cuerpo finito [35], representación que no se había usado hasta ese momento en ninguna representación software ni hardware.

La investigación siguió por ese camino, buscando nuevas representaciones para los elementos del cuerpo. Se desarrollo un nuevo algoritmo utilizando la representación polinomial Ghost Bit [18] y buscando mejorar este algoritmo surgió el problema de la escalabilidad de los algoritmos de multiplicación de cuerpos finitos implementados en hardware. Este problema se convirtió en el objetivo principal de la tesis y se propone una solución creando un nuevo algoritmo escalable en hardware reconfigurable.

En esta tesis se presenta el diseño mixto hardware-software de un multiplicador

en cuerpos finitos (del tipo $GF(2^m)$). Este algoritmo soluciona el problema de la escalabilidad y por tanto, se puede utilizar con valores de hasta cierto grado $m$ como máximo y para cualquier polinomio irreducible $f(x)$. Además, es posible utilizarlo para cualquier polinomio irreducible en forma de trinomio o pentanomio sin el rediseño del algoritmo en hardware.

## 1.4 Estado del arte

A lo largo de los últimos 20 años, se han estudiado diversas técnicas para la implementación eficiente y rápida de la aritmética de los cuerpos finitos, tanto en software como en hardware. En concreto, se ha prestado toda la atención en la multiplicación y la división ya que son las operaciones más costosas. Los elementos de un cuerpo finito pueden representarse en diferentes bases. La bases más comúnes son la base normal y la polinomial. Dependiendo de la base elegida para la representación de los elementos del cuerpo, la formulación matemática de la multiplicación es distinta [87] por lo que se producen implementaciones diferentes en los sistemas de computación, aunque todas ellas son equivalentes. Actualmente, la representación en base normal véase ([5, 75, 68]) y la implementación usando la base polinomial ([57, 40]) son las más utilizadas.

Podemos ver implementaciones usando la base polinomial en [62, 98, 41, 58] y la normal en [6, 96]. Una comparación entre las dos bases puede verse en [42]. Silverman [88] propone una nueva representación para los elementos de un cuerpo binario, llamada Ghost Based Bit (GBB), que ha sido utilizada para la realización de un nuevo algoritmo de multiplicación. Para una mayor profundización en la aritmética de los cuerpos finitos puede verse [56]

Las implementaciones eficientes en base normal necesitan una Optimal Normal Base (ONB) pero no todos los cuerpos tienen una base normal optima [68] la estructura de un multiplicador en base normal utiliza matrices y su estructura hardware no es tan sencilla como la estructura de un multiplicador en base polinomial. Además, el rendimiento de un multiplicador hardware en base normal se puede comparar al de un multiplicador polinomial. La unidad hardware que realiza el cuadrado de dos elementos de un cuerpo finito en base polinomial, se puede realizar de forma eficiente cuando el polinomio irreducible es un trinomio o un pentanomio [11]. Además, desde

el punto de vista del programador, los cuerpos finitos, o cuerpos de Galois $GF(2^m)$, son interesantes debido a que sus operaciones aritméticas están libres de acarreos.

### Multiplicadores escalables de cuerpos finitos

Para conseguir escalabilidad, se han propuesto varios algoritmos, todos ellos implementados en hardware. Okada, Torii, Itoh y Takenaka [73] proponen un multiplicador basado en polinomios que realiza arbitrariamente una multiplicación entre polinomios de grado $m$. Sato y Takano [82] proponen un multiplicador dual basado en la representación de Montgomery. Gaubatz [32] propone otro multiplicador dual y escalable.

## 1.5 Estructura de la tesis

Dentro del marco teórico de este trabajo, se enmarcan los capítulos 2, 3, 4 y 5. En ellos se repasa el estado del arte de la Criptografía, los cuerpos finitos, la computación reconfigurable y los algoritmos de multiplicación de cuerpos finitos. En el capítulo 5 se detalla la plataforma sobre la que se han desarrollado y simulado los algoritmos propuestos. En el capítulo 6 se describe un nuevo algoritmo de multiplicación entre elementos de un cuerpo finito. En el capítulo 7 se desarrolla una aplicación concreta del uso de la aritmética de los cuerpos finitos en sistemas criptográficos, en concreto, el algoritmo Rijndael. Los dos últimos capítulos corresponden a las conclusiones y a la bibliografía.

UNIVERSITAS
Miguel
Hernández

Capítulo 2

# Criptografía

## 2.1 Introducción

El ser humano siempre ha tenido secretos de muy diversa índole, y ha buscado mecanismos para mantenerlos fuera del alcance de miradas indiscretas. Si suponemos que existe un emisor y un receptor que quieren intercambiar mensajes y de intrusos (que pueden ser pasivos, si sólo escuchan la comunicación, o activos, si tratan de alterar los mensajes) que quieren interferir de algún modo en la comunicación entre ambos, se hace necesaria una herramienta capaz de proporcionar comunicaciones seguras sobre canales inseguros.

La palabra Criptografía procede de los vocablos griegos *kriptós* (oculto) y *gráphein* (escritura) y se puede definir como la disciplina que estudia los principios, métodos y medios de ocultar la información contenida en un mensaje. En contraposición a la criptografía está el criptoanálisis, que se puede definir como la disciplina que estudia el modo de descifrar en un tiempo razonable el contenido de un mensaje cifrado sin conocer la clave. Los campos de la criptografía y el criptoanálisis son conocidos en su conjunto como criptología.

Los usos más primitivos de la criptografía se encuentran documentados desde la época de *Julio César*(cifrado de César) aunque hay constancia también de su uso por persas y espartanos. Los mecanismos de cifrado clásicos se basaban en técnicas de transposición y sustitución de caracteres y fundamentaban su eficacia en el *secreto del algoritmo* empleado para el cifrado.

*Julio César* empleaba un sencillo algoritmo para evitar que sus comunicaciones militares fueran interceptadas. *Leonardo Da Vinci* escribía las anotaciones sobre

sus trabajos de derecha a izquierda y con la mano zurda. Otros personajes, como *Sir Francis Bacon* o *Edgar Allan Poe* eran conocidos por su afición a los códigos criptográficos, que en muchas ocasiones constituían un apasionante divertimiento y un reto para el ingenio.

Los sistemas criptográficos considerados clásicos son todos los sistemas de cifrado anteriores a la II Guerra Mundial, o lo que es lo mismo, al nacimiento de las computadoras. Estas técnicas tienen en común que pueden ser empleadas usando simplemente lápiz y papel, y que pueden ser criptoanalizadas casi de la misma forma. De hecho, con la ayuda de las computadoras, los mensajes cifrados empleando estos códigos son fácilmente descifrables, por lo que cayeron rápidamente en desuso. La transición desde la Criptografía clásica a la moderna se da precisamente durante la II Guerra Mundial, cuando el servicio de inteligencia aliado rompe la máquina de cifrado del ejército alemán, llamada *ENIGMA*. Todos los algoritmos criptográficos clásicos son simétricos, ya que hasta mediados de los años setenta no nació la Criptografía pública.

En el mundo de la Criptografía, una clave es un valor numérico generado mediante computación que los algoritmos modernos de cifrado usan para cifrar y descifrar mensajes. Generalmente, el algoritmo de cifrado es públicamente conocido y sometido a análisis por parte de expertos y usuarios. Se acepta la denominada hipótesis de *Kerckhoffs*, que establece que la seguridad del cifrado debe residir exclusivamente en el secreto de la clave y no en el del mecanismo de cifrado.

Los principales problemas de seguridad que resuelve la criptografía son: la privacidad, la integridad, la autenticación y el no rechazo.

- La **privacidad**, se refiere a que la información sólo pueda ser leída por personas autorizadas. Ejemplos: en la comunicación por teléfono, que alguien intercepte la comunicación y escuche la conversación quiere decir que no existe privacidad. Si mandamos una carta y por alguna razón alguien rompe el sobre para leer la carta, ha violado la privacidad.
  En la comunicación por Internet es muy difícil estar seguros de la privacidad de la comunicación, ya que no se tiene control de la línea de comunicación. Por lo tanto al cifrar (esconder) la información cualquier intercepción no autorizada no podrá desvelar la información. Esto es posible si se usan técnicas criptográficas, en particular la privacidad se logra si se cifra el mensaje con un

método simétrico.

- La **integridad**, se refiere a que la información no pueda ser alterada en el transcurso de ser enviada. La integridad es muy importante en las transmisiones militares ya que un cambio de información puede causar graves problemas. En Internet las compras se pueden hacer desde dos ciudades muy distantes, la información tiene necesariamente que viajar por una línea de transmisión de la cual no se tiene control, si no existe integridad podrían cambiarse por ejemplo el número de una tarjeta de crédito, los datos del pedido, en fin, información que causaría problemas a cualquier comercio y cliente. La integridad también se puede solucionar con técnicas criptográficas, particularmente con procesos simétricos o asimétricos.
- La **autenticidad**, se refiere a que se pueda confirmar que el mensaje recibido haya sido mandado por quien dice lo mando o que el mensaje recibido es el que se esperaba. Ejemplo: cuando se quiere cobrar un cheque a nombre de alguien, quien lo cobra debe de someterse a un proceso de verificación de identidad para comprobar que en efecto es la persona quien dice ser, esto en general se lleva a cabo con una credencial que anteriormente fue certificada y acredita la identidad de la persona que la porta. La verificación se lleva a cabo comparando la persona con una foto o con la comparación de una firma convencional. Por Internet es muy fácil engañar a una persona con quien se tiene comunicación respecto a la identidad, resolver este problema es por lo tanto muy importante para efectuar una comunicación confiable.
  Las técnicas necesarias para poder verificar la autenticidad tanto de personas como de mensajes usan quizá la más conocida aplicación de la criptografía asimétrica que es la firma digital, de algún modo ésta reemplaza a la firma autógrafa que se usa comúnmente. Para autenticar mensajes se usa criptografía simétrica.
- El **no rechazo**, se refiere a que no se pueda negar la autoría de un mensaje enviado.

Cuando se diseña un sistema de seguridad, una gran cantidad de problemas pueden ser evitados si se puede comprobar la autenticidad, garantizar privacidad, asegurar integridad y el no-rechazo de un mensaje.

## 2.2 Criptosistemas

Definiremos un criptosistema como una quíntupla $(M; C; K; E; D)$, donde:

- $M$ representa el conjunto de todos los mensajes sin cifrar (lo que se denomina texto claro, o plaintext) que pueden ser enviados.
- $C$ representa el conjunto de todos los posibles mensajes cifrados, o criptogramas.
- $K$ representa el conjunto de claves que se pueden emplear en el criptosistema.
- $E$ es el conjunto de transformaciones de cifrado o familia de funciones (algoritmos) que se aplica a cada elemento de $M$ para obtener un elemento de $C$. Existe una transformación diferente $E_k$ para cada valor posible de la clave $k$.
- $D$ es el conjunto de transformaciones de descifrado, análogo a $E$.

Todo criptosistema ha de cumplir la siguiente condición: $D_k(E_k(m)) = m$ es decir, que si tenemos un mensaje $m$, lo ciframos empleando la clave $k$ y luego lo desciframos empleando la misma clave, obtenemos de nuevo el mensaje original $m$.

Existen dos tipos de criptosistemas, el criptosistema de clave privada o simétrico y el criptosistema de clave publica o asimétrico. En la práctica se emplea una combinación de estos dos tipos de criptosistemas, puesto que los segundos presentan el inconveniente de ser computacionalmente mucho más costosos que los primeros y los primeros tienen el inconveniente del intercambio de las claves de forma segura. En el mundo real se codifican los mensajes (largos) mediante algoritmos simétricos, que suelen ser muy eficientes, y luego se hace uso de la criptografía asimétrica para codificar y enviar a su destinatario las claves simétricas (cortas).

## 2.3 Criptografía Simétrica

La criptografía simétrica se refiere al conjunto de métodos que permiten tener comunicación segura entre las partes siempre y cuando anteriormente se hayan intercambiado la clave correspondiente que llamaremos clave simétrica. La simetría se refiere a que las partes tienen la misma clave tanto para cifrar como para descifrar. Este tipo de criptografía se conoce también como criptografía de clave privada.

La criptografía de clave privada fue el primero de los métodos criptográficos basados en clave. La clave debe ser compartida por los diferentes participantes de

la operación, de modo que en el origen se puedan cifrar los datos mediante la clave propuesta, y en el destino se puedan descifrar los datos con la misma clave. El problema de este método es que si la clave es conocida por otras personas, entonces tendrán acceso a todos los datos. Las técnicas de clave privada o simétrica tienen fundamentos de complejidad diversa, pero todas usan una clave $k$, que es conocida por el emisor de los mensajes y por el receptor, y mediante el uso de un algoritmo que usa esa clave se cifra y descifra el mensaje que se quiere proteger.

La principal ventaja que presenta la utilización de clave privada es la existencia de algoritmos muy rápidos y eficientes, especialmente si se implementan en hardware. Si la clave es lo bastante larga (típicamente se usan valores de 56 a 1024 bits) es imposible romperlas en un tiempo razonable usando ataques por fuerza bruta. El principal inconveniente estriba en la necesidad de que todas las partes conozcan la clave, que debe ser distribuida mediante una transacción separada y diferente a la transmisión del mensaje cifrado. Este es precisamente el punto más vulnerable del mecanismo: la distribución de la clave; ya que si esta fuese interceptada se pondría en peligro todo el mecanismo. La clave, por tanto, debe ser transmitida por un canal seguro para poder asegurar la eficacia del sistema criptográfico.

La gran mayoría de los algoritmos de cifrado simétricos se apoyan en los conceptos de confusión y difusión inicialmente propuestos por Shannon, que se combinan para dar lugar a los denominados cifrados de producto. Recordemos que la confusión consiste en tratar de ocultar la relación que existe entre el texto plano, el texto cifrado y la clave. Un buen mecanismo de confusión hará demasiado complicado extraer relaciones estadísticas entre las tres cosas.

Por su parte, la difusión trata de repartir la influencia de cada bit del mensaje original lo mas posible entre el mensaje cifrado. Hemos de hacer notar que la confusión por si sola sería suficiente, ya que si establecemos una tabla de sustitución completamente diferente para cada clave con todos los textos planos posibles tendremos un sistema extremadamente seguro. Sin embargo, dichas tablas ocuparían cantidades astronómicas de memoria, por lo que en la práctica serían inviables. Por ejemplo, un algoritmo que codificara bloques de 128 bits empleando una clave de 80 bits necesitaría una tabla de aproximadamente 1063 entradas.

Lo que en realidad se hace para conseguir algoritmos fuertes sin necesidad de almacenar tablas enormes es intercalar la confusión (sustituciones simples, con tablas

pequeñas) y la difusión (permutaciones). Esta combinación se conoce como cifrado de producto. La mayoría de los algoritmos se basan en diferentes capas de sustituciones y permutaciones, estructura que denominaremos Red de sustitución-Permutación. En muchos casos el criptosistema no es mas que un paso simple de sustitución-permutación repetido $n$ veces, como ocurre con Data Encryption Standard (DES).

Existe una clasificación de este tipo de criptografía en tres familias, la criptografía simétrica de bloques (*block cipher*), la criptografía simétrica de flujo (*stream cipher*) y la criptografía simétrica de resumen (*hash functions*). Aunque con ligeras modificaciones un sistema de criptografía simétrica de bloques puede modificarse para convertirse en alguna de las otras dos formas, sin embargo es importante verlas por separado dado que se usan en diferentes aplicaciones.

La criptografía simétrica ha sido la más usada en toda la historia, ésta ha podido ser implementada en diferentes dispositivos, manuales, mecánicos, eléctricos, hasta los algoritmos actuales que son programables en cualquier computadora. La idea general es aplicar diferentes funciones al mensaje que se quiere cifrar de tal modo que solo conociendo una clave pueda aplicarse de forma inversa para poder así descifrar.

Aunque no existe un tipo de diseño estándar, quizá el más popular es el de *Feistel*, que consiste esencialmente en aplicar un número finito de interacciones de cierta forma, que finalmente, da como resultado el mensaje cifrado.

### 2.3.1 Ejemplos de algoritmos simétricos

El algoritmo simétrico mas extendido mundialmente es DES [20]. Data de mediados de los setenta, cuando fue adoptado como estándar para las comunicaciones seguras por el Gobierno de los EE.UU.

A mediados de 1998, se demostró que un ataque por la fuerza bruta a DES era viable, debido a la escasa longitud que emplea en su clave. Una empresa sin animo de lucro, llamada Electronic Frontier Foundation (EFF), logro fabricar una maquina capaz de descifrar un mensaje DES en menos de tres días. Curiosamente, pocas semanas antes, un alto cargo de la National Security Agency (NSA) había declarado que dicho algoritmo seguía siendo seguro, y que descifrar un mensaje resultaba aun excesivamente costoso, incluso para organizaciones gubernamentales. No obstante, el algoritmo aun no ha demostrado ninguna debilidad grave desde el punto de vista

teórico, por lo que su estudio sigue siendo plenamente interesante.

El algoritmo DES codifica bloques de 64 bits empleando claves de 56 bits. Es una Red de Feistel de 16 rondas, más dos permutaciones, una que se aplica al principio $P_i$ y otra que se aplica al final $P_f$, tales que $P_i = P_f^{-1}$. Para descifrar basta con usar el mismo algoritmo $P_i = (P_f^{-1})$. Una de las variantes de DES consiste en aplicar varias veces el algoritmo DES con diferentes claves al mensaje original, a este algoritmo se le conoce como Triple DES o Triple DES (TDES). El funcionamiento de TDES consiste en aplicar 3 veces DES.

En los últimos 20 años se han diseñado una gran cantidad de sistemas criptográficos simétricos [28], sin embargo no han tenido el alcance de DES, a pesar de que algunos de ellos tienen mejores propiedades.

Podemos afirmar que el objetivo primordial actual de la criptografía simétrica es la búsqueda de un nuevo sistema que pueda reemplazar a DES en la mayor parte de aplicaciones. Es así como se ha optado por convocar a un concurso de sistemas criptográficos simétricos y que este decida quien será el nuevo estándar al menos para los próximos 20 años.

El National Institute of Standards and Technology (NIST) [16] convocó a un concurso para poder tener un sistema simétrico que sea seguro y pueda usarse al menos en los próximos 20 años como estándar. En la mitad del año de 1998 se aceptaron 15 candidatos, estos se han sometido a pruebas públicas por parte del NIST. Los finalistas fueron: Cifrador en bloque de IBM (MARS), Algoritmo de cifrado en bloque derivado del RC5 (RC6), Rijndael, Serpent, y Twofish, y finalmente el que gano fue Rijndael.

Las principales características que se pide al algoritmo ganador del concurso, llamado Advanced Encryption Standard (AES), son que al menos sea tan seguro y rápido como TDES, es decir, que al menos evite los ataques conocidos. Además de que pueda ser implementado en una gran parte de aplicaciones. Una vez designado el ganador AES, este podrá ser usado tanto como cifrador de bloques (*block cipher*), como cifrador de flujo (*stream cipher*), como función resumen (*hash function*), y como generador de números pseudo aleatorios.

#### 2.3.1.1 Rijndael

En criptografía, el estandar de encriptación AES, que actualmente es el algoritmo Rijndael [23], es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Se espera que sea usado en el mundo entero y analizado exhaustivamente, como fue el caso de su predecesor, DES. Al contrario que DES, Rijndael es una red de sustitución-permutación, no una red de Feistel. El AES es rápido tanto en software como en hardware, es relativamente fácil de implementar y requiere poca memoria. Como nuevo estándar de cifrado, se está utilizando actualmente a gran escala.

Estrictamente hablando, AES no es precisamente Rijndael (aunque en la práctica se los llama de manera indistinta) ya que Rijndael permite un mayor rango de tamaño de bloques y longitud de claves; AES tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 ó 256 bits, mientras que Rijndael puede ser configurado con una clave que sea múltiplo de 32 bits, con un mínimo de 128 bits y un máximo de 256 bits.La clave se expande usando el esquema de claves de Rijndael. La mayoría de los cálculos del algoritmo AES se hacen en un cuerpo finito determinado.

Hasta 2006, no se ha encontrado ningún ataque exitoso contra el AES. Entre los ataques más potentes a la criptografía simétrica están el criptoanálisis diferencial [95] y lineal [7], sin embargo no han podido ser muy eficientes en la práctica dado que la longitud de claves utilizada, al ser muy grande, les obliga a realizar cálculos durante siglos enteros.

## 2.4 Criptografía Asimétrica o de clave Pública

La solución al problema de inseguridad que supone la distribución de la clave privada a través de un canal inseguro (como es el caso de Internet) apareció en 1976. En ese año, *Whitfield Diffie* y *Martin Hellman* [25] demostraron la posibilidad de construir sistemas de cifrado que no precisan la transferencia de una clave secreta entre emisor y receptor, evitando así los problemas derivados de la búsqueda de canales seguros para tal transferencia. Estos sistemas son conocidos como de clave

pública o asimétrica.

La criptografía de clave pública usa un mecanismo diferente. También se la conoce como criptografía asimétrica porque se emplean diferentes claves para el cifrado y descifrado. Con la criptografía de clave pública, existen dos claves disponibles, la clave privada que solo el usuario conoce, y la clave pública a la que todo el mundo tiene acceso. Ambas claves son generadas al mismo tiempo empleando el mismo algoritmo. Con este sistema si alguien quiere enviar un mensaje, lo cifra con la clave pública del destinatario, y entonces solo éste es capaz de descifrar el mensaje con su clave privada. Las claves privadas nunca son intercambiadas o enviadas a través de la red. Debido a que la clave privada permite la identificación de su propietario, las firmas digitales se basan en criptografía de clave pública.

Estos criptosistemas deben cumplir además que el conocimiento de la clave pública $k_q$ no permita calcular la clave privada $k_p$. Ofrecen un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros puesto que únicamente viaja por el canal la clave pública, que solo sirve para cifrar, o para llevar a cabo autenticaciones. Todos los esquemas de criptografía de clave pública, se basan en un problema intratable o computacional mente difícil de lograr. En RSA es el problema de la factorización. El otro problema cuya intratabilidad nos permite utilizarlo en criptografía, es el llamado problema del logaritmo discreto en cuerpos finitos: dados dos elementos pertenecientes al cuerpo, $P$ y $Q$ y un número natural $k$, con $P = k.Q$, es computacionalmente difícil, averiguar $k$ conocidos $P$ y $Q$.

Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros. Otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original, bien sea porque la longitud de la clave es enorme. Se basan en general en plantear al atacante problemas matemáticos difíciles de resolver. En la práctica muy pocos algoritmos son realmente útiles. El más popular por su sencillez es RSA, que ha sobrevivido a multitud de ataques, si bien necesita una longitud de clave considerable.

Los algoritmos asimétricos emplean generalmente longitudes de clave mucho mayores que los simétricos. Por ejemplo, mientras que para algoritmos simétricos se considera segura una clave de 128 bits, para algoritmos asimétricos se recomiendan claves de al menos 1024 bits. Además, la complejidad de cálculo que comportan es-

tos últimos los hace considerablemente mas lentos que los algoritmos de cifrado por bloques. En la practica los métodos asimétricos se emplean únicamente para codificar la clave de sesión (simétrica) de cada mensaje. *Diffie* y *Hellman* [25], proponen un algoritmo para el intercambio de claves, sin embargo no fue hasta que el popular método RSA fue publicado en 1978, cuando toma forma la criptografía asimétrica.

## 2.5 Seguridad en los criptosistemas de clave pública

A diferencia de la criptografía simétrica, donde se realiza la transformación de datos mediante permutaciones y transposiciones, en la criptografía de clave pública, casi todos los métodos de transformación de los datos, se realizan mediante la ejecución de operaciones aritméticas complejas sobre un cuerpo finito. Esto se debe a la seguridad de estos criptosistemas se basa en conjeturas matemáticas muy difícil de resolver, por lo que, con el fin de romper el criptosistema, es necesario resolver el problema subyacente.

Cuanto más difícil es el problema en que se basa un sistema de cifrado, mayor será la la seguridad que ofrece el sistema. En la práctica, dos problemas se han adoptado para construir sobre ellos cifrado de clave pública: el problema de la factorización de enteros grandes y el cálculo del logaritmo discreto. La complejidad de estos dos problemas es función del cuerpo finito utilizado. Mientras que los criptosistemas basados en la factorización de enteros se consideran en la actualidad seguros utilizando operandos de 1024 bits, los criptosistemas basados en logaritmo discreto, ofrecen un nivel de seguridad equivalente utilizando operandos de 160-bits.

La criptografía asimétrica o de clave pública se divide en tres familias según el problema matemático del cual basan su seguridad:

(a) La primera de las familias es la que basa su seguridad en el Problema de Factorización de Enteros (PFE) [93], los sistemas que pertenecen a esta familia son, el sistema RSA, y el Sistema cifrado publico Robin-Williams (RW).

(b) La segunda familia es la que basa su seguridad en el Problema del Logaritmo Discreto (PLD), a esta familia pertenece el sistema de *Diffie-Hellman* Diffie-Hellman de intercambio de claves y el sistema Digital Signature Algorithm

(DSA) [93] de firma digital.

(c) La tercera familia es la que basa su seguridad en el PLDE [47, 46], como el Intercambio Diffie-Hellman Elíptico (DHE).

UNIVERSITAS
Miguel
Hernández

Capítulo 3

# Conceptos matemáticos

## 3.1 Introduccion

El objetivo de este capítulo es proporcionar la información necesaria para entender los cuerpos finitos. Una descripción más detallada puede encontrarse en [63, 86].

Los cuerpos finitos son cada vez más importantes en diversas áreas como la informática, la estadística, la teoría de la información, y la ingeniería. También, muchos algoritmos criptográficos realizan operaciones aritméticas sobre estos cuerpos [90]. Veremos también algunas definiciones matemáticas previas a la definición de un cuerpo finito.

Igualmente, la implementación eficiente de la aritmética de los cuerpos finitos, es un requisito importante en los sistemas criptográficos y en los códigos correctores de errores, porque muchas operaciones de estos sistemas se realizan utilizando operaciones aritméticas en el cuerpo finito subyacente. En concreto, los cuerpos finitos son necesarios para implementar criptosistemas de curvas elípticas.

## 3.2 Grupos

**Definición 3.1:**

Un *grupo* $\langle G, * \rangle$ es un conjunto finito de elementos $G$ con una operación binaria $*$ definida en parejas de elementos en $G$

$$* : G \times G \leftarrow G, (a, b) \mapsto a * b \tag{3.1}$$

que tiene las siguientes propiedades:

- Clausura: $\forall\ x, y \in S, x * y \in S$
- Asociativa: $\forall\ x, y, z \in S, (x * y) * z = x * (y * z)$
- Identidad: $\exists\ I \in S$ tal que $x * I = x$
- Inverso: $\forall\ x \in S$, existe un único $y \in S$ tal que $x * y = y * x = I$

**Ejemplo 3.1:**

El conjunto de los enteros $Z$ con la operación $+$ forman un grupo, con el 0 como elemento identidad.

El conjunto $Z_n$ con la operación de multiplicación modulo $n$ no es un grupo, dado que no todo los elementos tienen inverso para la multiplicación.

El conjunto $Z_{n^*}$ con la operación de multiplicación modulo $n$ forma un grupo, con el 1 como elemento identidad.. ■

**Definición 3.2:** El orden de un elemento $x$ de un grupo finito $G$ es el menor entero positivo que cumple:

$$x^t = x * x * \ldots x* = 1 \tag{3.2}$$

(a) Si el orden de un elemento $x$ es igual al número de elementos del grupo, $n$, entonces $x$ es el generador de $G$.
(b) Si $G$ tiene generador, se dice que $G$ es cíclico.

**Definición 3.3:** Si $\forall x, y \in S$, $x * y = y * x \in S$ a **G** se le llama grupo abeliano.

## 3.3 Anillos

**Definición 3.4:** Un anillo $(R, *, +)$ consiste en un conjunto $R$ con dos operaciones binarias $+$ y $*$ que satisfacen los siguientes axiomas:

(a) $(R, +)$ es un grupo conmutativo con el 0 como elemento identidad

(b) $x * y * z = (x * y) * z, \forall x, y, z \in R$ *propiedad asociativa.*
(c) Existe un elemento llamado identidad multiplicativa, 1, con $1 \neq 0$, tal que $x * 1 = 1 * x = x, \forall x \in R$.
(d) $x * (y + z) = (x * y) + (x * z)$ y $((x + y) * z) = (x * z) + (y * z)$ $\forall x, y, z \in R$ *propiedad distributiva*
(e) y además, si $x * y = (y * x)$ $\forall x, y, z \in R$.

**Ejemplo 3.2:** El conjunto $Z$ de enteros con las operaciones usuales de suma y multiplicación es un anillo conmutativo. El conjunto $Z_m$ de enteros con las operaciones usuales de suma y multiplicación módulo $m$ es un anillo conmutativo. ■

## 3.4 Cuerpos

**Definición 3.5:** Un cuerpo $(F, +, *)$ consiste en un conjunto $F$ con dos operaciones binarias $+$ y $*$, con elemento identidad para la suma, 0 y con elemento identidad para la multiplicación, 1 que satisfacen los siguientes axiomas:

(a) $(F, +, *)$ es un anillo conmutativo.
(b) Todos los elementos de $F$ distintos de cero tienen inverso multiplicativo.

**Definición 3.6:** La característica de un cuerpo es el menor entero positivo $m$ tal que $\sum_{i=1}^{m} 1 = 0$.

**Ejemplo 3.3:** Alguno de los cuerpos más conocidos son:

- El conjunto de los números reales $R$ forman un cuerpo de característica 0 con sus operaciones usuales.
- El conjunto de los enteros $Z$ con las operaciones de suma y multiplicación no es un cuerpo porque el único elemento distinto de cero con inverso multiplicativo son el 1 y el $-1$.
- El conjunto $Z_p$ con las operaciones de suma y multiplicación modulo $p$ es un

cuerpo si y solo si $p$ es primo. Si $p$ es primo, entonces $Z_p$ tiene característica $p$. Sea el cuerpo $Z_5$. La tabla de la suma módulo 5 es la siguiente:

| + | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 3 | 4 | 0 |
| 2 | 2 | 3 | 4 | 0 | 1 |
| 3 | 3 | 4 | 0 | 1 | 2 |
| 4 | 4 | 0 | 1 | 2 | 3 |

■

**Tabla 3.1:** *Suma en* $Z_5$

**Definición 3.7:** Un subconjunto $E$ de un cuerpo $F$ es un subcuerpo de $F$ si $E$ a la vez un cuerpo con respecto a las operaciones de $F$. En tal caso, decimos que $F$ es un cuerpo extendido de $E$. Si $E \neq F$, decimos que $E$ es un subcuerpo propio de $F$. Un cuerpo que no contenga subcuerpos propios se denomina *cuerpo primo*.

## 3.5 Polinomios

**Definición 3.8:** Si R es un anillo conmutativo, entonces un polinomio en la indeterminada $x$ sobre R es una expresión de la forma:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \tag{3.3}$$

donde $a_i \in R, \forall i \in 0, 1, \ldots, n$. Al elemento $a_i$ se le llama *coeficiente* de $x^i \in f(x)$.

(a) El mayor entero $m$ tal que $a \neq 0$ es el grado de $f(x)$. Se denota $deg(f)$ y $a_m$ es el coeficiente lider. Si todos los coeficientes de $f(x)$ son iguales a 0 entonces $f(x)$ se llama polinomio cero. A los polinomios cero también se le llaman polinomios constantes.

(b) Un polinomio mónico es un polinomio cuyo coeficiente líder es igual a 1.

(c) El anillo de polinomios $R[x]$ es el anillo formado por el conjunto de todos los polinomios en la indeterminada $x$ y con coeficientes en $R$. Las dos operaciones son la suma y la multiplicación estándar de polinomios, con la aritmética entre coeficientes realizadas en $R$. El elemento identidad 0 de la suma es el polinomio cero y el elemento identidad 1 de la multiplicación es el polinomio mónico constante.

**Ejemplo 3.4:** Sea $f(x) = x^4 + 3x^3 + 2x + 4$ y $g(x) = 4x^3 + 3x + 4$ elementos de $Z_5[x]$. La suma y multiplicación de $f(x)$ y $g(x)$ es:

$$f(x) + g(x) = x^4 + 2x * 3 + 3 \tag{3.4}$$

$$f(x)g(x) = 4x^7 + 2x^6 + 3x^5 + x^4 + 3x^3 + x^2 + 1 \tag{3.5}$$

■

A partir de ahora, trataremos exclusivamente con polinomios sobre un cuerpo arbitrario $F$.

**Definición 3.9:** Gracias al hecho de que $F$ es un cuerpo, todos los coeficientes distintos de cero tienen inverso y se puede realizar la operación división de polinomios. Por tanto, si $g(x)$ y $h(x) \neq 0$ son polinomios en $F$, entonces existen dos polinomios $q(x)$ (el cociente) y $r(x)$ (el resto) en $F[x]$ tal que:

$$g(x) = q(x)h(x) + r(x) \quad donde\ deg(r) < deg(h) \tag{3.6}$$

y se denota: $r(x) = g(x) \bmod h(x)$

# 3.6 Congruencias de polinomios

**Definición 3.10:** Dados 3 polinomios $g(x), h(x)$,y$f(x) \in F[x]$, $g(x)$ es congruente a $h(x)$ modulo $f(x)$ si $f(x)$ divide a $g(x) - h(x)$ y se denota $g(x) \equiv h(x) \bmod f(x)$

PROPIEDADES:

(a) $g(x) \equiv h(x) \bmod f(x)$ si y solo si $g(x) \bmod f(x) = h(x) \bmod f(x)$
(b) La relación $g(x) \equiv h(x) \bmod f(x)$ es una relación de equivalencia. □

De las propiedades anteriores puede verse que las relaciones de congruencia dividen $F[x]$ en clases de equivalencia o colección de elementos con propiedades comunes. Si $n$ es el grado de $f(x)$ entonces cada clase de equivalencia contiene exactamente un polinomio de grado $d < n$. Además, si $F$ es un cuerpo finito, entonces el número de clases de equivalencia es igual a $|F|^n$ donde $|F|$ es el número de elementos de $F$. El conjunto de clases de equivalencia se escribe como $F[x]/f(x)$.

PROPIEDADES:

(a) $F[x]/f(x)$ es un anillo conmutativo.
(b) Si $f(x)$ es irreducible, entonces $F[x]/f(x)$ es un cuerpo. □

## 3.7 Cuerpos finitos

Un cuerpo finito es un cuerpo $F$ que contiene un número finito de elementos. El orden de un cuerpo finito es el número de elementos en el cuerpo.

**Definición 3.11:** Sea $p$ un número primo, $F = Z_p = F_p$ y $f(x)$ un polinomio irreducible de grado $n$ sobre $F_p$. El cuerpo $F_p[x]/f(x)$ contiene $q = p^n$ elementos y se denomina $F_q$ o $GF(q)$ (cuerpo de Galois). Un cuerpo finito $F$ de orden $q$, $F_q$, solamente puede existir si $q$ es una potencia de un número primo $q = p^n$. El número primo $p$ es llamado la característica de $GF$. Si $n = 1$ el cuerpo es llamado un cuerpo primo. Si $n \geq 2$, $F_p$ es llamado un cuerpo extendido. Si $p = 2$ el cuerpo se denomina $GF(2^m)$ o cuerpo binario. Dos cuerpos son isomorficos si tienen la misma estructura aunque difieran en la representación de sus elementos. Si $n = 1$, el correspondiente cuerpo $F_q$ es isomórfico a $F_p$, $F_q$ puede considerarse una extensión de $F_p$ de grado $n$. El conjunto $F_{q^*}$ es un cuerpo cíclico.

Una manera de construir un cuerpo finito, por ejemplo $F_p^m$ es la siguiente: primero debemos encontrar un polinomio $f(x) \in F_p[x]$ de grado $m$ irreducible sobre $F_p$

(llamado cuerpo base). Sea $F_p[x]/f(x)$. Si consideramos $\alpha = x \bmod f(x)$ entonces $f(\alpha) = 0$. Con esto el cuerpo finito $F_p^m$ lo identificamos con el conjunto,

$$F_p^m = a - 0, a_1\alpha + \ldots + a_{m-1}\alpha^{m-1} : a_i \in F_p \tag{3.7}$$

en donde las operaciones de suma y multiplicación están determinadas por el polinomio $f(x)$ utilizado.

Podemos usar el cuerpo $F_{p^m}$ como cuerpo base para construir otro cuerpo finito, en concreto $F_{q^n}$ donde $q = p^m$.

**Ejemplo 3.5:** Consideramos como cuerpo base a $F_2 = 0, 1$ y construimos el cuerpo $F_{p^3}$ usando el polinomio $f(x) = x^3 + x + 1$. El cuerpo finito resultante es:

$$F_{2^3} = a_0 + a_1\alpha + a_2\alpha^2 : a_i \in F_2 \; y \; \alpha^3 + \alpha + 1 = 0 \tag{3.8}$$

■

Sea $F_{q^n}$ el cuerpo finito definido por un polinomio irreducible $f(x)$ de grado $n$, $Fq^n$ es isomorfo como espacio vectorial a ${F_q}^n$. A través de:

$$\theta : F_{q^n} \mapsto {F_q}^n, ) = (a_0, a_1, a_2, a_3, \ldots, a_{n-1}) \tag{3.9}$$

donde $\alpha \equiv \bmod f(x)$.

La representación vectorial es muy útil para almacenar y simplificar el manejo de los elementos de un cuerpo finito utilizando herramientas informáticas como veremos más adelante.

## 3.7.1 Bases de un cuerpo finito

Si se considera una extensión finita $F = F_{q^m}$ del cuerpo finito $E = F_q$ como un espacio vectorial sobre $E$ , entonces $F$ tiene dimensión $m$ sobre $E$ y si $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ es una base de $F$ sobre $E$, entonces cada elemento $\alpha \in F$ se puede representar de forma única como: $\alpha = a_1\alpha_1 + a_2\alpha_2 + \ldots + a_m\alpha_m$, donde $a_i \in E$ , para cada $1 <= i <= m$. Existen diferentes bases de un cuerpo $F$ sobre $E$, pero las más importantes son la base polinomial y la base normal [63, 86].

**Definición 3.12:** Si $\alpha \in F = F_{q^m}$ y $E = F'_q$ entonces la traza de $\alpha$ sobre $E$, $Tr_{F/E}$ se define como:

$$Tr(\alpha) = \alpha + \alpha^q + \ldots + \alpha^q m - 1 \tag{3.10}$$

**Definición 3.13:** Sea $\beta \in F_{q^n}$, si $BP = 1, \beta, \beta^2, \beta, \ldots, \beta^{n-1}$ son linealmente independientes, decimos que $BP$ es una base polinomial de $F_q^n$ sobre $F_q$.

Recordando la construcción de un cuerpo finito descrita en la sección anterior y considerando $\alpha$ tal que $f(\alpha) = 0$ (donde $f(x)$ es irreducible), en la definición de cuerpo:

$$F_q^n = \sum_{i=0}^{n-1} a_i \alpha^i : a_i \in F_q \ y \ f(\alpha) = 0 \tag{3.11}$$

el conjunto $BP = 1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$, es una base polinomial de $F_q^n$ sobre $F_q$.

Para realizar operaciones aritméticas como la suma y la multiplicación sobre $F_q^n$, debemos escoger una base y después realizar los algoritmos en términos de la base elegida. Veamos como se realizaría la suma y la multiplicación en un cuerpo finito eligiendo una base polinomial.

## 3.7.2 Suma

Sean $\gamma_1 \gamma_1 \in F_q^n$, cuyas representaciones en términos de una base polinomial $BP$ sobre $F_q^n$ son:

$$\gamma_1 = \sum_{i=0}^{n-1} a_i \alpha^i, \ \gamma_2 = \sum_{i=0}^{n-1} b_i \alpha^i, \ \text{con } a_i, b_i \in F_q \tag{3.12}$$

La suma de estos dos elementos es la adición usual en vectores, es decir:

$$\gamma_1 + \gamma_2 = \sum_{i=0}^{n-1} (a_i + b_i) \alpha^i, \tag{3.13}$$

**Ejemplo 3.6:** Sean $\alpha^5 + 2\alpha^2 + 2\alpha + 1$, $\alpha^5 \alpha^5 + 2\alpha^4 + \alpha^3 + 2 \in F_3^6$. Entonces su suma es: $(\alpha^5 + 2\alpha^2 + 2\alpha + 1) + (\alpha^5 \alpha^5 + 2\alpha^4 + \alpha^3 + 2) = (1+1)\alpha^5 + (0+2)\alpha^4 + (0+1)\alpha^3 +$

$(2+0)\alpha^2+(2+0)\alpha+(1+2)=2\alpha^5+2\alpha^4+\alpha^3+2\alpha^2+2\alpha^5$ Si lo representamos en términos de sus coordenadas obtenemos: $(100221)+(121002)=(221220)$ ■

### 3.7.3 Multiplicación

Sean $\gamma_1\gamma_1 \in F_q^n$ como en la sección anterior. La multiplicación de estos elementos es:

$$\gamma_1+\gamma_2=(\sum_{i=0}^{n-1} a_i\alpha^i)*\sum_{i=0}^{n-1} b_i\alpha^i)=\sum_{k=0}^{2n-2} c_k\alpha^k) \text{ donde} c_k=\sum_{k=i+j} a_i*b_j) \tag{3.14}$$

Como se puede observar, necesitamos expresar las potencias $\alpha^k$ para $n <= k <= 2n-2$ en terminos de la base $BP_{F_q^n}$ para finalmente realizar sumas. Esta operación es similar a la multiplicación de polinomios. Este algoritmo tiene complejidad $O(n^2)$, ya que se realizan $2n-2$ sumas de $c_k$ que a su vez lleva a lo mas a $n$ multiplicaciones, siendo $O(2n^-2n)=O(n^2)$.

## 3.8 Cuerpos finitos $GF(2^m)$

Los Cuerpos finitos $GF(2^m)=F_{2^m}$ son una extensión del cuerpo $GF(2)=F_2=Z_2$. Los cuerpos finitos de orden $2^m$ son cuerpos finitos de característica 2, también conocidos como cuerpos binarios. Los cuerpos finitos $GF(2^m)$ tienen interés debido a su amplio número de áreas técnicas en las que se utilizan, como corrección de errores, criptografía, generación de números aleatorios o procesado digital de señal.

Los elementos de un cuerpo finito $GF(2^m)$ son polinomios

$$0,1,\alpha,\alpha^2,\ldots,\alpha^{m-1},\alpha^{m-2},\ldots,\alpha+1 \tag{3.15}$$

donde $\alpha$ es una raíz de un polinomio irreducible $f(x)$ sobre $GF(2)$, $f(\alpha)=0$ y en donde los coeficientes del polinomio pertenecen a $GF(2)=0,1$.

**Ejemplo 3.7:** Sea $\alpha \in GF(2^3)=F_2^3$ una raiz del polinomio irreducible $f(x)=x^3+1 \in GF(2)[x]$. Los elementos de $GF(2^3)$ representados en la base polinomial $\alpha^2,\alpha,1$ son los de la tabla 3.2. La representación binaria del elemento se realiza escribiendo los coeficientes del polinomio en forma de vector

| Elementos en $GF(2^3)$ | Polinomio | Representación binaria |
|---|---|---|
| $0$ | $0$ | $(0,0,0)$ |
| $\alpha$ | $\alpha$ | $(0,1,0)$ |
| $\alpha^2$ | $\alpha^2$ | $(1,0,0)$ |
| $\alpha^3$ | $\alpha^2+1$ | $(1,0,1)$ |
| $\alpha^4$ | $\alpha^2+\alpha$ | $(1,1,0)$ |
| $\alpha^5$ | $\alpha+1$ | $(0,1,1)$ |
| $\alpha^6$ | $1$ | $(0,0,1)$ |
| $\alpha^7$ | $\alpha^2+\alpha+1$ | $(1,1,1)$ |

**Tabla 3.2:** *Elementos de $GF(2^3)$*

## 3.9 Aplicaciones criptográficas de los cuerpos finitos

### Rijndael

Uno de los algoritmos implementados en nuestra plataforma de implementación es el algoritmo de clave simétrica Rijndael. En 1996 el National Institut of Standard and Technology [70], inició un concurso para elegir un nuevo algoritmo que reemplazara al DES y convertirlo en el nuevo estandar en cifrado simétrico, AES anunciando en Octubre del 2000 al ganador, el algoritmo Rijndael, diseñado por los belgas J.Daemen y V. Rijmen [23].

El algoritmo Rijndael es un sistema simétrico de cifrado por bloques, cuya entrada pueden ser bloques de tamaño variable, y cuya llave también es de tamaño variable. Utiliza la misma clave para el proceso de cifrado como para el proceso de descifrado. Su diseño permite la utilización de claves de sistema con longitud variable siempre que sea múltiplo de 4 bytes. La longitud de las claves utilizadas por defecto son 128 (AES-128), 192 (AES-192) y 256 (AES-256) bits. De la misma manera el algoritmo permite la utilización de bloques de información con un tamaño variable siempre que sea múltiplo de 4 bytes, siendo el tamaño mínimo recomendado de 128

bits, y el tamaño mínimo de 16 bytes.

El algoritmo Rijndael opera a nivel de bytes, interpretando estos como elementos de un cuerpo de Galois $GF(2^8)$, y a nivel de registros de 32 bits, considerándolos como polinomios de grado menor que 4 con coeficientes que son a su vez polinomios en $GF(2^8)$.

En este algoritmo todos los bytes se interpretan como elementos de un cuerpo finito. Como se ha visto anteriormente, en los cuerpos del tipo $GF(2^8)$ los coeficientes serán los restos del módulo 2, es decir, 0 y 1, lo que permite una representación binaria. Por lo tanto, cada elemento del cuerpo se representa con $m$ bits y el número de elementos será $2^m$. Por ejemplo, para el cuerpo $GF(2^3)$sus elementos son: 0, 1, $x$, $x+1$, $x^2$ $(x^2+1)$, $(x^2+x)$, $(x^2+x+1)$ que son precisamente todos los restos de un polinomio de grado $m-1$. En el caso del algoritmo Rijndael, se definen operaciones a nivel de byte, encontrándonos en el cuerpo $GF(2^8)$. Un byte $B$, se compone de los bits $(b7\ b6\ b5\ b4\ b3\ b2\ b1\ b0)$, si lo consideramos como un polinomio con coeficientes en $0, 1$ tenemos el polinomio: $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$.

Por ejemplo, un byte con el valor hexadecimal 57, en binario 01010111, se corresponde con el polinomio $x^6 + x^4 + x^2 + x + 1$.

## Criptosistema de Curvas elípticas

Los CCE [14] es una de las disciplinas más prometedoras en el campo de los cifradores asimétricos. Las curvas elípticas constituyen un formalismo matemático conocido y estudiado desde hace más de 150 años, y presentan una serie de propiedades que da lugar a problemas cuya solución es costosa de encontrar, en términos computacionales, debido a su dificultad, análogos a los que presentaba la aritmética modular, lo cual las hace válidas para aplicar algunos de los algoritmos asimétricos más conocidos. Si bien su estructura algebraica es algo compleja, su implementación suele resultar tanto o más eficiente que la aritmética modular, y además con claves mucho más cortas se puede alcanzar el mismo nivel de seguridad que con otras técnicas. Las primeras propuestas de uso de las curvas elípticas en Criptografía fueron hechas por *Neal Koblitz* y *Victor Miller* [65] en 1985.

Este criptosistema se basa en el hecho de que el conjunto de puntos racionales de una curva elíptica, es un grupo conmutativo.

Precisamente, el principal argumento que esgrimen los detractores de estas técnicas es que, si bien las curvas elípticas han sido objeto de estudio y análisis durante más de un siglo, las propiedades que pueden estar directamente relacionadas con su calidad como base para un sistema criptográfico, apenas llevan quince años siendo consideradas. Para introducir el concepto de Curva Elíptica, vamos a establecer un paralelismo con otro formalismo mucho más cercano e intuitivo: los números enteros. Los números enteros constituyen un conjunto para el que podíamos definir una serie de operaciones, y éstas tenían unas propiedades concretas. Estos conjuntos y operaciones mostraban una estructura que hacía surgir problemas computacionalmente difíciles de tratar.

Las curvas elípticas se pueden formar sobre diferentes estructuras algebraicas como los números reales o los cuerpos finitos.

**Definición 3.14:** Una curva elíptica sobre $\mathbb{R}$ es el conjunto de puntos del plano $(x, y)$ que cumplen la siguiente ecuación:

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

**Curvas Elípticas en $\mathbb{R}$** Los coeficientes $a$ y $b$ caracterizan unívocamente cada curva. Si $x^3+ax+b$ no tiene raíces múltiples, lo cual es equivalente a que $4a^3+27b^2 \neq 0$, entonces la curva correspondiente, en conjunción con un punto especial $O$, llamado punto en el infinito, más la operación suma que definiremos más adelante, es lo que vamos a denominar grupo de curva elíptica $E(\mathbb{R})$. Hay que recalcar que $O$ es un punto imaginario situado por encima del eje de abscisas a una distancia infinita, y que por lo tanto no tiene un valor concreto.

**Suma en E($\mathbb{R}$)** Ya tenemos un conjunto sobre el que trabajar. Nuestro siguiente paso será definir una ley de composición interna que, dados dos elementos cualesquiera, nos devuelva otro que también pertenezca al conjunto. Denominaremos suma a esta operación y la representaremos mediante el signo $+$. Sean los puntos $r = (r_x, r_y); s = (s_x, s_y); p = (p_x, p_y); t = (t_x, t_y) \in E(\mathbb{R})$, la operación suma se define de la siguiente forma:

- $r+O = O+r = r$, sea cual sea el valor de $r$. Esto quiere decir que O desempeña

el papel de elemento neutro para la suma.

- Si $r_x = s_x$ y $r_y = -s_y$ decimos que r es el opuesto de s,escribimos$r = -s$, y además $r_y = r + s = s + r = O$ por definición.
- Si $r \neq s_x$ y si $r \neq s$, entonces para sumarlos, se traza la línea recta que une $r$ con $s$. Dicha recta cortará a la curva en un punto. La suma $t$ de $r$ y $s$ será el opuesto de dicho punto.
- Para sumar un punto $p$ consigo mismo, se emplea la tangente a la curva en $p$. Si $p_y \neq 0$, dicha tangente cortará a la curva en un único punto. La suma $t = p + p$, será el opuesto a dicho punto.
- Para sumar un punto $p$ consigo mismo, cuando $p_y = 0$, la tangente a la curva será perpendicular al eje de abscisas, por lo que podemos considerar que corta a la curva en el infinito. Por lo tanto, $p + p = O$ si $p_y = 0$.

Por razones de simplicidad en la notación diremos que sumar un punto $p$ consigo mismo $k$ veces, es como multiplicar dicho punto por el escalar $k$, y lo notaremos $k_p$.

Nótese que, cuando se suma $r$ y $-r$, la recta que los une resulta perpendicular al eje de abscisas, por lo que podemos considerar que corta a la curva en el infinito, dando como resultado $O$. Compruebese además, que cuando $r_y = 0$, se cumple:

$$2r = r + r = O$$
$$3r = 2r + r = O + r = r$$
$$4r = 3r + r = r + r = O$$
$$\ldots$$

Algebraicamente, la suma de curvas elípticas se define de la siguiente forma: sea $r = (r_x, r_y)$ y $s = (s_x, s_y)$, donde $r \neq -s$, entonces $r + s = t$ donde:

$$d = \frac{r_y - s_y}{r_x - s_x}; t_x = d^2 - r_x - s_x; t_y = -r_y + d(r_x - t_x);$$

y cuando queremos sumar un punto consigo mismo, tenemos que $2p = t$ donde:

$$d = \frac{3p_y^2}{2p_y}; t_x = d^2 - 2p_x; t_y = -p_y + d(p_x - t_x);$$

Si nos fijamos un poco, podremos observar que $d$ representa a la pendiente de la recta que une $r$ y $s$, o bien a la tangente en el punto $p$.

**Curvas Elípticas en** $GF(p)$ Recordemos que un cuerpo de Galois $GF(p)$ es el grupo finito generado por $p$, siendo $p$ un número primo. En dicho conjunto todos los elementos menos el cero tienen inversa, por lo que podemos sumar, restar, multiplicar

y dividir exactamente de la misma forma que en $\mathbb{R}$, por lo que nada nos impide ver qué puntos cumplen la ecuación:

$$y^2 = x^3 + ax + b \mod p$$

definiendo de esta forma el conjunto $E(GF(p))$.

**Curvas Elípticas en** $GF(2^m)$ Vamos a dar un paso más. Los elementos de $GF(p^m)$ y las operaciones entre ellos presentan unas propiedades análogas a las de los elementos de $GF(p)$, con la característica añadida de que, cuando $p = 2$, la implementación de los algoritmos correspondientes es más sencilla y rápida. Nada nos impediría, pues, definir el conjunto $E(GF(2^m))$. En $GF(2^m)$, debido a su especial estructura, la ecuación de curva elíptica que será útil para nuestros propósitos es la siguiente:

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

y la única condición necesaria para que genere un grupo es que $b \neq 0$.

Dentro de $GF(2^m)$, los puntos de nuestra curva van a ser polinomios de grado $m-1$ con coeficientes binarios, 0,1 por lo que podrán representarse mediante cadenas de bits.

## Uso en criptografía

En el uso criptográfico, se elige un punto base $G$ específico y publicado para utilizar con la curva $E(q)$. Se escoge un número entero aleatorio $k$ como clave privada, y entonces el valor $P = k * G$ se da a conocer como clave pública (nótese que la supuesta dificultad del PLDE implica que $k$ es difícil de deducir a partir de $P$). Si María y Pedro tienen las claves privadas $kA$y $kB$, y las claves públicas $PA$ y $PB$, entonces María podría calcular $kA * PB = (kA * kB) * G$; y Pedro puede obtener el mismo valor dado que $kB * PA = (kB * kA) * G$.

Esto permite establecer un valor *secreto* que tanto María como Pedro pueden calcular fácilmente, pero que es muy complicado de derivar para una tercera persona. Además, Pedro no consigue averiguar nada nuevo sobre $kA$ durante ésta transacción, de forma que la clave de María sigue siendo privada.

Los métodos utilizados en la práctica para cifrar mensajes basándose en este

valor secreto consisten en adaptaciones de antiguos criptosistemas de logaritmos discretos originalmente diseñados para ser usados en otros grupos. Entre ellos se podrían incluir Diffie-Hellman, ElGamal y DSA.

La realización de las operaciones necesarias para ejecutar este sistema es más lenta que para un sistema de factorización o de logaritmo discreto módulo entero del mismo tamaño. De todas maneras, los autores de sistemas de CCE creen que el PLDE es significativamente más complicado que los problemas de factorización y así se puede obtener la misma seguridad mediante longitudes de clave mucho más cortas utilizando CCE, hasta el punto de que puede resultar más rápido que, por ejemplo, RSA. Los resultados publicados hasta la fecha tienden a confirmar esto, aunque algunos expertos se mantienen escépticos.

La CCE ha sido ampliamente reconocida como el algoritmo más fuerte, para una longitud de clave más corta que la utilizada por los demás algoritmos de clave pública, por lo que podría resultar útil sobre sistemas que tengan requisitos muy limitados de ancho de banda.

El NIST y el American National Standards Institute X9 (ANSI X9) han establecido unos requisitos mínimos de tamaño de clave de 1024 bits para RSA y DSA y de 160 bits para ECC, correspondientes a un bloque simétrico de clave de 80 bits. El NIST ha publicado una lista de curvas elípticas recomendadas de 5 tamaños distintos de claves $(80, 112, 128, 192, 256)$.

UNIVERSITAS
Miguel
Hernández

Capítulo 4

# Algoritmos de multiplicación en Cuerpos finitos

## 4.1 Aritmética en cuerpos finitos

Para definir las operaciones aritméticas de curvas elípticas se deben primero definir las operaciones aritméticas sobre cuerpos finitos como: la suma, la resta, la multiplicación, el cuadrado, la raíz cuadrada, el inverso multiplicativo, la división y la exponenciación.

Hay varias maneras de realizar eficientemente la aritmética de los cuerpos finitos dependiendo de la base elegida para representarlo. Las dos bases más utilizadas en aplicaciones criptográficas son la normal [86] y la polinomial [77]. Cada una de estas representaciones llevan a diferentes algoritmos para la implementación de las distintas operaciones entre elementos de un cuerpo aunque si elegimos operar en una representación normal, hay que realizar conversiones entre bases para realizar las operaciones aritméticas correctamente. La base polinomial es una de las mas prometedoras ya que proporciona al programador mas libertad en la elección del polinomio irreducible. En esta tesis se utiliza la representación polinomial. En esta representación, los elementos de $GF(2^m)$ se pueden representar como polinomios de grado al menos $m-1$ de la forma: $GF(2^m) = a(\alpha)| = a_{m-1}\alpha^{m-1} + \ldots + a_1\alpha^1 + a_0 \; \forall \; a_i \in GF(2)$, donde los coeficientes $a_i$ son las coordenadas del polinomio en su representación binaria. La base polinomial también se puede representar como el conjunto $1, x, x^2, \ldots, x^{m-1}$ y por tanto

Las operaciones aritméticas en $GF(2^m)$ se realizan módulo un polinomio irre-

ducible $f(x)$ sobre $GF(2)$. La suma de polinomios se realiza módulo 2 que en la práctica es una simple operación $XOR$ entre las coordenadas de su representación binaria. La suma y la resta es la misma operación, por tanto $1-x$ es igual a $1+x$.

En esta representación, cada polinomio se representa como un vector m-dimensional $a \in GF(2^m)$ de la forma:

$$a(x) = \sum m - 1^{i=0} a_i x^i = a_i x^i + \ldots + a_1 x^1 + a_0, a_i \in GF(2) \tag{4.1}$$

Por defecto, $f(x) = 0$ por tanto $x$ es una raíz de $f(x)$. Entonces,

$$x^m = -g(x) = -\sum_{i=0}^{m-1} g_i x^i \ , \ g_i \in F_2 \tag{4.2}$$

nos ofrece una manera de realizar la reducción siempre que nos encontremos potencias de $x$ mayores que $m-1$ al realizar operaciones aritméticas en el cuerpo. Generalmente, $f(x)$ se elige con el menor número de coeficientes $g_i$, en forma de trinomios o pentanomios, para minimizar la complejidad de las operaciones aritméticas.

Aunque la ecuación 4.2 nos proporciona una manera de reducir un polinomio, también se puede reducir un polinomio $c(x)$ dividiéndolo por $f(x)$ de manera que el resto de esa división es el resultado de la reducción. A lo largo del texto, $a(x)$ mod $f(x)$ hará referencia a la reducción del polinomio $a(x)$ por $f(x)$. Con la representación polinomial, la suma y la multiplicación de dos elementos del cuerpo $GF(2^m)$, $a(x), b(x)$, $a(x) = \sum_{i=0}^{k-1} a_i x^i$, $b = \sum_{i=0}^{k-1} b_i x^i$ se realiza de la siguiente forma:

- Suma: La suma de los elementos de este cuerpo, es la suma usual de los polinomios utilizando la aritmética modulo 2 para los coeficientes. Es una operación XOR ($\oplus$) coeficiente a coeficiente, tanto en hardware como en software.

$$c(x) = a(x) + b(x) = \sum_{i=0}^{m-1} (a_i \oplus b_i) x^i = \sum_{i=0}^{m-1} c_i x^i$$

- Multiplicación: $C(x) = A(x)B(x)$ mod $f(x) = \sum_{i=0}^{m-1} (c_i) x^i$ donde la multiplicación $a(x)b(x)$ es una multiplicación de polinomios y todos los $x^t$, con $t \geq m$ se reducen con la ecuación 4.2.

Así pues, hay dos formas de calcular la multiplicación: la primera, digamos el método clásico, consiste en realizar primero la multiplicación polinomial clásica, y

después hacer la operación de reducción al resultado de ese producto. En una implementación en hardware esto implica la construcción de dos estructuras diferentes para cada una de estas dos operaciones, restando modularidad y simplicidad al circuito. La segunda, que realiza la reducción a cada uno de los productos parciales, es conocida como el método de shift-add o entrelazado, y es, entre los dos métodos, el más utilizado para implementaciones en hardware debido a que permite el desarrollo de estructuras más simples y regulares. Una discusión sobre ambos métodos se puede consultar en [39]

**Ejemplo 4.1 (Cuerpo binario $\mathbf{F_{2^4}}$):** Los elementos del cuerpo binario $\mathbf{F_{2^4}}$ son los siguientes 16 polinomios binarios, de grado menor o igual que 3:

$$\begin{array}{cccc} 0 & x^2 & x^3 & x^3+x^2 \\ 1 & x^2+1 & x^3+1 & x^3+x^2+1 \\ x & x^2+x & x^3+x & x^3+x^2+x \\ x+1 & x^2+x+1 & x^3+x+1 & x^3+x^2+x+1 \end{array}$$

ejemplos de operaciones aritméticas son, con el polinomio irreducible $f(x) = (x^4+x^2+1)$

- Suma: $(x^3+x^2+1)+(x^2+x+1) = (x^3+x)$
- Resta: $(x^3+x^2+1)-(x^2+x+1) = (x^3+x)$
- Multiplicación: $(x^3+x^2+1)(x^2+x+1) = (x^2+1)$ ya que $[(x^3+x^2+1)(x^2+x+1) = (x^5+x+1)]$ y $[(x^5+x+1) \bmod f(x) = (x^2+1)]$ ■

# 4.2 Multiplicación clásica

La multiplicación clásica en cuerpos finitos es una traslación del algoritmo clásico de multiplicación de polinomios. La multiplicación requiere de dos pasos para su cálculo: el producto de los dos polinomios, cuyo resultado será un polinomio de orden $2m-2$, y una operación de reducción módulo el polinomio irreducible, que resultará en un polinomio de orden $m-1$. Al multiplicar los polinomios $a(x)$ y $b(x)$ se tiene que $c'(x) = a(x)b(x)$ y al tomar el residuo módulo el polinomio irreducible $f(x)$, es decir $c(x) = c'(x) \bmod f(x)$ se obtendrá el resultado de la multiplicación como $C'(x)$. La elección del polinomio irreducible con pocos términos, facilitara la

operación de reducción.

Esta multiplicación se puede escribir en forma de matriz [71] de manera que los coeficientes de $d(x)$ vienen determinados por la siguiente ecuación:

$$\left\{ \sum_{i=0}^{k} a_i b_{k-i};\ k = 0, \ldots, m-1 \sum_{i=k}^{2m-2} a_{k-i+(m-1)} b_{i-m-1};\ k = m, \ldots, 2m-2 \right\} \quad (4.3)$$

En este caso la suma es una operación $XOR$ y la multiplicación una operación $AND$. El número total de puertas lógicas para la computación paralela de este producto de matrices es $m^2$ puertas $AND$ y $m-1$ puertas $XOR$ [71]. Las puertas $AND$ operan en paralelo y su retraso es $T_and$ mientras que las puertas $XOR$ están organizadas en forma de árbol con una profundidad de $[log_2 j]$ para $j$ operandos. En total, el delay de esta operación de multiplicación de polinomios tiene un retraso en su implementación hardware de $T_{and} + [log_2 m] T_{xor}$.

Después de la multiplicación de polinomios, se debe de realizar la reducción. En esta operación, el polinomio $d(x)$ con grado $2m-2$, se reduce a un polinomio $c(x)$ de grado $deg(c(x)) \leq m-1$:

$$\begin{aligned} c(x) = d(x) \bmod f(x) = \\ (d_{2m-2} x^{2m-2} + \ldots + d_1 x + d_0) \bmod f(x) = \\ c_{m-1} x^{m-1} + \ldots + c_1 x + c_0 \end{aligned} \quad (4.4)$$

.

Esta operación de reducción se puede ver como un mapeo línear entre los $2m-1$ coeficientes de $d(x)$ y los $m$ coeficientes de $c(x)$. Este mapeo se representa en notación matricial [77] de la siguiente manera:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots & 0 & r_{0,0} & \ldots & r_{0,m-2} \\ 0 & 1 & \ldots & 0 & r_{1,0} & \ldots & r_{1,m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & r_{m-1,0} & \ldots & r_{m-1,m-2} \end{bmatrix} \begin{bmatrix} d_0 \\ \vdots \\ d_{m-1} \\ d_m \\ \vdots \\ d_{2m-2} \end{bmatrix}$$

La matriz $R$ con los coeficientes $R_i$ es una función única que depende de los coeficientes de $f(x)$. Eligiendo apropiadamente el polinomio $f(x)$ podemos reducir la complejidad de esta operación.

### 4.2.1 Algoritmo Karatsuba

El algoritmo Karatsuba-Ofman ([97]) es un método recursivo para multiplicar polinomios. Se sabe que dos polinomios arbitrarios de una variable de grado menor o igual a $m-1$ con coeficientes en $GF(2^m)$, se pueden multiplicar con no mas de $m^2$ multiplicaciones en $GF(2^m)$ y $(m-1)^2$ sumas. El algoritmo de Karatsuba reduce el número de multiplicaciones y sumas, para valores de $m$ grandes. El algoritmo de Karatsuba restringido a polinomios con $m = 2^t$ con $t$ un entero puede verse en [77].

## 4.3 Implementación software de la aritmética de cuerpos finitos

El cuerpo $GF(2^m)$ contiene $2^m$ elementos donde $m$ es un entero positivo. La adición en $GF(2^m)$ es la suma de polinomios la cual se implementa en la práctica con la operación XOR entre sus coeficientes. Una multiplicación en el cuerpo consiste en la multiplicación de dos polinomios y la reducción posterior mediante un polinomio irreducible $f(x)$ de grado $m$ con coeficientes en $GF(2)$. Un polinomio de grado $m$ se dice que es irreducible sobre $GF(2)$ si es mónico y no se puede factorizar como producto de polinomios de grado estrictamente menor que $m$ y coeficientes en $GF(2)$.

Un elemento en $GF(2^m)$ se puede representar mediante un polinomio de grado menor o igual que $m$. Para su uso en computación, un polinomio $a(x) = (a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0)$ se representa mediante el array de $m$ bits, $a(x) = (a_{m-1}, a_{m-2}, \ldots, a_0)$. En un procesador con un tamaño de palabra de $w$ bits, este array se divide en $d$ dígitos o palabras, donde $d = \lceil \frac{m}{w} \rceil$.

Por tanto, un elemento de $GF(2^m)$ se almacena en $d$ palabras en procesadores con palabras de $w$ bits. Si el array de palabras se representa mediante $(A_{s-1}, A_{s-2}, \ldots, A_0)$, tenemos las expresiones siguientes:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = \sum_{j=0}^{d-1} A_j(x) x^{wj}$$

donde

$$A_j(x) = \sum_{i=0}^{w-1} a_{i+wj} x^i$$

La palabra más significativa $A_{d-1}$ se rellena con ceros por la izquierda, si es necesario.

El algoritmo más simple para realizar la multiplicación es el algoritmo llamado desplazamiento y suma [64] con la reducción intercalada, algoritmo 1. Este algoritmo se basa en que $a \otimes b$ se puede reescribir $a_{m-1}x^{m-1}b + \ldots + a_1xb + a_0b$. La iteración $i$esima del algoritmo 1 computa $x^ib \bmod f(x)$ y suma el resultado al acumulador $c$ si $a_i = 1$. La operación $x^ib \bmod f(x)$ se calcula desplazando a la izquierda el polinomio $b$, seguido de una suma $b + f(x)$ si $b_m = 1$.

Este algoritmo no está optimizado para implementarlo en software debido a los múltiples desplazamientos (paso 4, algoritmo 1) que provocan continuos accesos a

**Entrada**: Polinomios A(x) y B(x) de grado menor o igual a $m-1$.
**Salida** : $C(x) = A(x) \otimes B(x) \bmod f(x)$.

```
1 if a_0 = 1 then c ← b;
2 else c ← 0;
3 for i ← 1 to m − l do
4 |   b ⟵ b · x mod f(x);
5 |   if a_i = 1 then c ← c + b;
6 |   ;
7 end
```

**Algoritmo 1:** Desplazamiento y Suma. Shift and Add

memoria. Un método más eficiente para multiplicar dos elementos de un cuerpo finito en esta base, es multiplicar primero los dos polinomios, obteniendo un polinomio de longitud $2m-1$ y luego hacer la reducción mediante $f(x)$. El método mas sencillo para multiplicar dos polinomios es el llamado 'scholbook' o 'lapiz y papel', algoritmo 2, y es parecido al algoritmo desplazamiento y suma pero sin la reducción.

**Entrada**: Polinomios A(x) y B(x) de grado menor o igual a $m-1$.
**Salida** : $C(x) = A(x)B(x)$.

```
1 for i ← 0 to m − l do
2 |   la multiplicación por x es un desplazamiento;
3 |   for j ← 0 to m − l do
4 |   |   c ⟵ c + a_j · b_i x^{i+j};
5 |   end
6 end
```

**Algoritmo 2:** Scholbook

Para multiplicar dos polinomios existen otros métodos pero los más rápidos son los llamados 'comb method' de Lopez y Dahab, [58], descritos en el algoritmo 3 y 4. Al contrario que en el algoritmo 'desplazamiento y suma', los bits de $a$ no se recorren secuencialmente sino que primero se comprueba el bit 0 de todas las palabras de $a$, desde $A[0]$ hasta $A[s-1]$, (bits $a_0, a_w, a_2w, \ldots$) y así sucesivamente. Luego, el algoritmo comprueba el bit 1 en todas las palabras, luego el bit 2, etc.. Comparado con el algoritmo suma y desplazamiento, el algoritmo 'comb' reduce el

número de desplazamientos desde $m-1$ a $w-1$.

El algoritmo left-to-right comb es parecido al right-to-left comb, pero testea los bits de $a$ de izquierda a derecha. Es más lento que el right-to-left comb porque los desplazamientos los realiza sobre el acumulador $c$ y no sobre $b$.

**Entrada**: Polinomios A(x) y B(x) de grado menor o igual a $m-1$.
**Salida** : $C(x) = A(x) \times B(x)$.

```
1 C ← 0;
2 for i ← 0 to w − l do
3 |   for j ← 0 to s − l do
4 |   |   C ← (b_{wi+j} · SHIFT(A << w · i)) + C;
5 |   |   A ← SHIFT(A << 1);
6 |   end
7 end
```

**Algoritmo 3:** Algoritmo Right-to-Left comb para la multiplicación de polinomios

La operación SHIFT$(A << k) = \sum_{i=0}^{m-1} a_i x^{i+k}$, realiza una operación de desplazamiento de k bits en $A$. La operación SHIFT$(A << w \cdot i))$ en donde $w$ es el tamaño de palabra del procesador, actúa sobre el mismo conjunto de bits y y el desplazamiento software se sustituye por una nueva referencia del puntero que estamos utilizando para referenciar las palabras de $A$.

El algoritmo Left-to-Right es similar al algoritmo 3 pero los bits se recorren de izquierda a derecha, es decir, desde el bit más significativo hasta el menos significativo. Este método es más lento porque realiza los desplazamientos sobre el producto parcial $C$ y no sobre $A$ aunque se puede mejorar su rendimiento reduciendo el número de desplazamientos, utilizando una técnica de ventana y a costa de aumentar el espacio de almacenamiento para guardar la tabla de multiplicación de todos los polinomios $a(z)$ de grado menor que $w$ con $b(z)$ [41], (hay que almacenar los resultados de las multiplicaciones entre $B(x)$ y todos los polinomios de grado menor o igual que $w$ en una tabla). De esta manera, se procesan $W$ bits de $A[j]$ a la vez, siendo $W$ el tamaño de la ventana.

## 4.4 Implementación Hardware

Las implementaciones en hardware de las operaciones aritméticas sobre cuerpos finitos $GF(2^m)$, se presentan como una alternativa a las implementaciones en software debido a la mayor seguridad y rendimiento que se obtiene con ellas. Tales implementaciones en hardware usan como elemento básico a los circuitos integrados (CIs o chips), cuyas frecuencias de operación permiten cumplir con los requerimientos de alta velocidad que se desean. En aquellas situaciones en las que se necesite un mayor rendimiento por parte de las aplicaciones criptográficas, las implementaciones de la aritmética en software se deben de sustituir por implementaciones hardware.

¿Cuales son los criterios que hay que tener en cuenta para decidirse por una implementación hardware u otra?

- Coste. Es necesario evaluar el coste final de la plataforma de implementación.
- Rendimiento. Un sistema criptográfico para un servidor requiere ejecutar cientos o miles de operaciones por segundo mientras que un portátil o un ordenador personal requerirán sólo unas pocas operaciones.
- Complejidad del diseño. Si el circuito a implementar es muy complejo, eso se traslada a una mayor necesidad de área en los dispositivos hardware y por tanto aumentará el coste final.
- Flexibilidad. Habilidad de las implementaciones para realizar operaciones aritméticas en diferentes cuerpos.
- Flexibilidad de algoritmos. Muchos protocolos criptográficos requieren la negociación del algoritmo antes de realizar operaciones, es decir, necesitan la capacidad de decidir entre varios algoritmos.
- Plataforma hardware. ¿Un circuito dedicado fabricado en Very-large-scale integration (VLSI) o una algoritmo diseñado en una Field Programmable Gate array (FPGA)?. La relación entre rendimiento y coste es mucho menor en las FPGA. Las implementaciones VLSI son mucho más caras. Además, con las FPGA se pueden construir muchos prototipos antes de decidirse por el diseño final, mientras que con VLSI, una vez creado el circuito impreso, cuesta mucho modificarlo.
- Escalabilidad. Es recomendable que el algoritmo a implementar nos de la oportunidad de cambiar el nivel de seguridad, cambiando el tamaño del cuerpo sin rediseñar el algoritmo.

La elección de uno u otro criterio dependerá la aplicación final. No es lo mismo un sistema criptográfico para un servidor de gama alta que para un dispositivo de telefonía móvil.

La eficiencia de la aritmética de cuerpos finitos se caracteriza mediante las siguientes dos medidas:

(a) La complejidad de espacio, definida por el número de puertas lógicas requeridas para la construcción del circuito (área ocupada en la FPGA).
(b) La complejidad de tiempo, que se mide por el número de retardos de las puertas que forman la ruta crítica de datos o datapath.

La operación más importante dentro de la aritmética de los cuerpos finitos es la multiplicación. En hardware, los algoritmos de multiplicación de cuerpos finitos, se dividen en tres categorías dependiendo del número de bits del resultado que generan:

(a) bit-serial Se genera un bit del resultado final cada ciclo de reloj
(b) paralelo Se generan todos los bits del resultado final en un ciclo de reloj
(c) digit-serial Se genera un número de bits determinado, llamado dígito cada ciclo de reloj

La elección de uno u otro dependerá de la relación entre área y rendimiento que necesite el sistema final. La investigación sobre arquitecturas para multiplicadores hardware, primero se centro en los multiplicadores en paralelo [54, 101] y serial [52]. Más adelante, aparecieron los multiplicadores digit-serial [89].

La tabla 4.1 muestra las características de los tres tipos de multiplicadores hardware.

| | Bit-Serial | Digit-Serial | Paralelo |
|---|---|---|---|
| Ciclos de reloj | $O(m)$ | $O(m/D)$ | $O(1)$ |
| Área | $O(m)$ | $O(mD)$ | $O(m^2)$ |

**Tabla 4.1:** *Características Multiplicadores*

El uso de las arquitecturas paralelas en criptografía no es aconsejable, debido a que los cuerpos finitos utilizados son muy grandes, del orden de $m$ entre 160 y 1024, y su diseño eh hardware ocuparía mucha área en el chip, por tanto, voy a describir dos versiones de cada uno de los multiplicadores restantes, bit-serial y digit-serial: Bit-serial MSB (most significant bit, bit más significativo), bit-serial

LSB (least significant bit, bit más significativo), digit-serial MSD (most significant digit, digito más significativo), digit-serial LSD (least significant digit, dígito menos significativo). Las arquitecturas MSB/MSD realizan el producto recorriendo el multiplicando empezando en el bit/dígito más significativo y terminando en el bit/dígito menos significativo. Las arquitecturas LSB/LSD realizan el producto recorriendo el multiplicando empezando en el bit/dígito menos significativo y terminando en el bit/dígito mas significativo.

### 4.4.1 Multiplicadores serial

Los algoritmos bit-serial se utilizan en sistemas con recursos limitados y en donde un alto rendimiento no sea necesario, como por ejemplo las tablets y smartphones. Pueden verse ejemplos en [13, 11, 55]. Otra característica de los multiplicadores bit-serial es que se pueden diseñar para aceptar polinomios de reducción de grado variable, [33].

Los multiplicadores serial procesan todos los coeficientes del multiplicando en paralelo en el primer paso, mientras los coeficientes del multiplicador son procesados uno a uno. Para calcular una operación de multiplicación requieren de $O(m)$ pulsos de reloj y utilizan $O(m)$ puertas lógicas para su construcción, lo que define sus complejidades de tiempo y espacio respectivamente. Como puede verse, no son tan rápidos como los paralelos, pero requieren de poco espacio físico para su implementación en hardware. Estos multiplicadores pueden ser diseñados en dos versiones diferentes, dependiendo del orden en que son tratados los coeficientes del multiplicador: LSB y MSB

**Multiplicador LSB**

Sean $A(\alpha) = \sum_{i=0}^{k-1} a_i \alpha^i$ y $B(\alpha) = \sum_{i=0}^{k-1} b_i \alpha^i$ y $C(\alpha) = \sum_{i=0}^{k-1} c_i \alpha^i$ su producto. En este tipo de multiplicador, la multiplicación puede ser determinada de manera polinomial:

$$
\begin{aligned}
C(x) = A(x)B(x) \bmod F(x) \\
&= A(x)\left(\sum_{i=0}^{m-1} b_i x^i\right) \\
&= \sum_{i=0}^{m-1} b_i (x^i A(x) \bmod F(x) \\
&= b_0 A + b_1(xA(x) \bmod F(x)) + b_2(x^2 A(x) \bmod F(x)) + \ldots + \\
& b_{m-1}(x^{m-1} A(x) \bmod F(x))
\end{aligned}
\tag{4.5}
$$

**Entrada**: Polinomios A(x) y B(x) de grado menor o igual a $m-1$ $\in GF(2^m)$ .
**Salida** : $C(x) = A(x) \otimes B(x) \bmod f(x) \in GF(2^m)$.

1 $C \longleftarrow 0$;
2 **for** $i \leftarrow 0$ **to** $m-1$ **do**
3 $\quad C \longleftarrow b_1 A + C$;
4 $\quad A \longleftarrow Ax^i \bmod F(x)$;
5 **end**

**Algoritmo 4:** LSB

El algoritmo 4 muestra el algoritmo para esta multiplicación. Se podrá observar que se requieren $m$ ciclos de reloj para obtener el cálculo de una multiplicación. Por otro lado, en cada iteración, se realizan las siguientes operaciones:

- multiplicación de un elemento de $GF(p)$ por un elemento de $GF(p^m)$, es decir, se requiere de $m$ multiplicadores $GF(p)$ en paralelo.
- adición en $GF(p^m)$. Se requieren $m$ sumadores $GF(p)$.
- multiplicación por $x$.
- operación de reducción módulo $F(x)$.

La complejidad en cuanto al espacio es:

$AND = 2m$ $XOR = 2m - 1$

**Multiplicador MSB**

Sean $A(\alpha) = \sum_{i=0}^{k-1} a_i \alpha^i$ y $B(\alpha) = \sum_{i=0}^{k-1} b_i \alpha^i$ y $C(\alpha) = \sum_{i=0}^{k-1} c_i \alpha^i$ su producto. La multiplicación polinomial $C(x) = A(x)B(x) mod P(x)$, se realiza como sigue:

$$c_0 = b_{m-1}A(x) \bmod F(x) c_i = (xc_{i-1} + b_{m-i-1}A) \bmod F(x)$$

y entonces $C(\alpha) = c_{m-1}(\alpha)$.

**Entrada**: Polinomios A(x) y B(x) de grado menor o igual a $m-1$ $\in GF(2^m)$ .
**Salida** : $C(x) = A(x) \otimes B(x) \bmod f(x) \in GF(2^m)$.

```
1 C ⟵ 0;
2 for i ← 0 to m − 1 do
3 |   C ⟵ (b_{m−1}A + C mod F(x);
4 |
5 end
```

**Algoritmo 5:** MSB

En este multiplicador, el primer bit que se procesa del multiplicador es $b_{m-1}$, continuando en forma descendente hasta llegar a $b_0$.

El coste en área es:

$AND = 2m$ $XOR = m$

## 4.4.2 Multiplicación por dígitos, LSD

El multiplicador por dígitos, introducido en [89] presenta un equilibrio entre velocidad, área y consumo de potencia. Su operación se basa en que es capaz de procesar varios coeficientes del elemento multiplicador $A$ de manera simultanea. El número de coeficientes procesados en paralelo se define como el tamaño de los dígitos y se representa con la letra $D$.

Denotemos a $d = \lceil \frac{m}{D} \rceil$, el cual equivale al número total de dígitos en un polinomio de grado $m-1$. Si el array de palabras se representa mediante $(A_{s-1}, A_{s-2}, \ldots, A_0)$,

tenemos las expresiones siguientes:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = \sum_{j=0}^{s-1} A_j(x) x^{wj}$$

donde

$$A_j(x) = \sum_{i=0}^{w-1} a_{i+wj} x^i$$

La palabra más significativa $A_{s-1}$ se rellena con ceros por la izquierda, si es necesario.

La multiplicación entre $A(x)$ y $B(x)$ se realiza de la siguiente forma:

$$C(x) = A(x)B(x) \bmod F(x) = \sum_{i=0}^{d-1} A_i B(x) x^{Di} \bmod F(x).$$

**Entrada**: $A(x) = \sum_{i=0}^{m-1} a_i x^i, a_i \in GF(p), B(x) = \sum_{i=0}^{(m/D)-1} B_i x^{Di}$
**Salida** : $C(x) = A(x) \otimes B(x) \bmod f(x) \in GF(2^m)$.

1 $C \longleftarrow 0$;
2 **for** $i \leftarrow 0$ **to** $[m/D - 1]$ **do**
3 $\quad C \longleftarrow B_i A + C$;
4 $\quad A \longleftarrow Ax^D \bmod F(x)$;
5 $\quad C \longleftarrow [C \bmod F(x)]$
6 **end**

**Algoritmo 6:** Multiplicación por dígitos

Capítulo 5

# Plataforma de implementación

## 5.1 Alternativas de diseño para implementar algoritmos criptográficos

Tradicionalmente, para el desarrollo de sistemas computacionales se han planteado dos posibilidades bien definidas. Por un lado, la posibilidad de *ejecución software*, donde la tarea se realiza mediante la realización de un conjunto de instrucciones. Se trata de una solución muy flexible, porque permite ejecutar diferentes tareas aunque la eficiencia es menor dada la funcionalidad general del microprocesador. Por otro lado la posibilidad de *ejecución hardware*, donde se desarrolla un circuito especifico Application Specific Integrated Circuit (ASIC) para la tarea a realizar. Esta solución presenta un mejor rendimiento dada la especialidad del diseño y el posible paralelismo de la aplicación, pero reduce la flexibilidad del sistema debido a la imposibilidad de cambiar la funcionalidad una vez fabricado.

Los dispositivos reconfigurables, y por ello los sistemas reconfigurables, se presentan como una solución intermedia entre el uso de microprocesadores de propósito general y el diseño de ASIC. En una situación ideal los sistemas reconfigurables combinan lo mejor de las dos soluciones anteriores: la flexibilidad del software que se ejecuta en un microprocesador de propósito general y la velocidad del hardware específico [94]. La figura 5.1 ilustra la situación de los sistemas reconfigurables.

**Procesadores vs. Sistemas reconfigurables.** Los procesadores modernos están formados por pequeñas unidades de ejecución que presentan una fuerte multiplexación para soportar la demanda de computación. Sin embargo, en cada momento

solo una pequeña cantidad de los recursos hardware disponibles se emplean de forma activa. Además, los microprocesadores presentan una arquitectura fija, lo que implica que la decodificación secuencial de instrucciones, el acceso a la memoria y la existencia de una unidad de control fija, limitan el rendimiento que puede ser alcanzado para una aplicación. Por el contrario, en los sistemas reconfigurables las operaciones de computación se implementan mediante la ejecución paralela de varias unidades de ejecución diseñadas específicamente, en vez de ejecutarse de forma secuencial como las instrucciones en un microprocesador (ver figura 5.2).

**ASIC vs. Hardware reconfigurable.** En el otro lado del espectro de computación aparecen los ASIC, que son circuitos diseñados específicamente para aplicaciones determinadas y por tanto, presentan un rendimiento superior para un conjunto restringido de tareas computacionales debido a su funcionalidad fija. Por ejemplo, en ciertas aplicaciones de procesado de señal, los ASIC ofrecen un rendimiento superior a las mejores FPGA en un factor dos [67, 61]. Una desventaja clara de los ASIC es la imposibilidad de optimizaciones post-diseño y actualizaciones para mejorar el rendimiento [15]. La razón fundamental de la popularidad de los ASIC en aplicaciones comerciales es bien conocida: cualquier algoritmo del que se quiera obtener el máximo rendimiento debe ser implementado en hardware [17]. Los ASIC además, dominan el segmento del bajo consumo, donde los sistemas reconfigurables no son competitivos actualmente. La industria de semiconductores tiene contemplados tra-

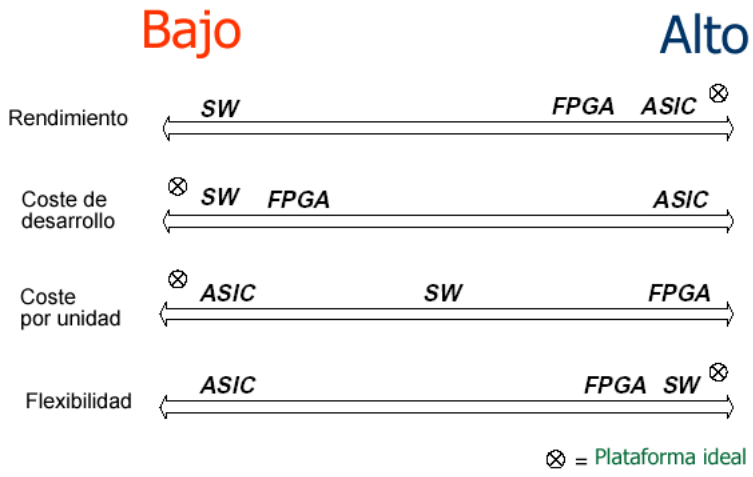


**Figura 5.1:** *Comparativa sistemas*

dicionalmente a los ASIC como la única alternativa económica para los productos comerciales en masa, y los primeros dispositivos reconfigurables no han invadido el territorio de los ASIC. Sin embargo, esta situación ha cambiado con la aparición de las modernas FPGA de millones de puertas que permiten niveles de integración sin precedentes. Los dispositivos reconfigurables deben hacer valer sus ventajas sobre los ASIC, por ejemplo en la reducción de costes, tiempo de diseño, y su flexibilidad para implementar diferentes funciones usando el mismo dispositivo [67].

## 5.2 Sistemas reconfigurables

El uso del 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 un tratamiento en tiempo real. Aplicaciones sobre redes de baja y media velocidad pueden hacer uso de dichos procesadores. Es común la im-

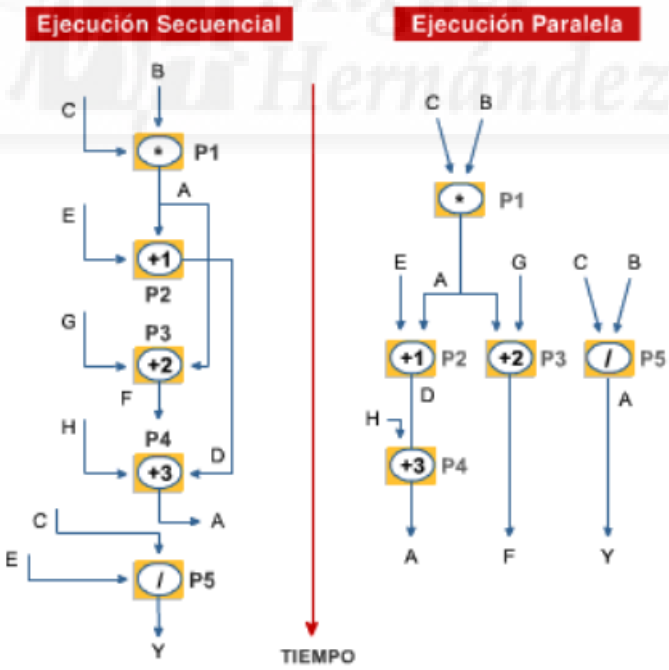


**Figura 5.2:** *Computación paralela vs. secuencial*

plementación de bibliotecas criptográficas para familias de procesadores, como lo son la familia Pentium. Dichas bibliotecas pueden ser optimizadas para el uso de los recursos 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.

El rendimiento obtenido por soluciones implementadas sobre plataformas software dependen del correcto uso de las características específicas que pudieran encontrarse en el procesador de propósito general, del rendimiento del propio procesador y de la habilidad del programador. La principal ventaja de las plataformas 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 capacidad de proceso 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 vídeo en tiempo real necesitan de un buen rendimiento en sus procesos para garantizar un límite mínimo de calidad en la transmisión. Las 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 implica costes muy altos. El mantenimiento posterior de los sistemas (como la actualización o corrección de los algoritmos implementados) no resulta viable por la dificultad 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 ofrecen la posibilidad de un ciclo de desarrollo rápido con un alto grado de flexibilidad.

El uso del hardware reconfigurable para implementar algoritmos criptográficos es recomendable por:

- Los algoritmos criptográficos utilizan operaciones aritméticas uy lógicas a nivel de bit cuya implementación cabe muy bien en la estructura de de la FPGA.
- Es posible que el programador del sistema criptográfico implemente en la misma FPGA distintos algoritmos, unos dedicados a los sistemas de clave publica y otros dedicados a los sistemas de clave privada.

Desde hace dos décadas, las FPGA han estado presentes en todo tipo de aplicaciones de red, telecomunicaciones, procesamiento de vídeo, etc. Estos dispositivos son circuitos electrónicos formados por un array de bloques lógicos programables y un conjunto de interconexiones también programables para unir esos bloques que permiten a los diseñadores de sistemas desarrollar soluciones que requieren de la velocidad del hardware a medida, manteniendo el beneficio de la flexibilidad de las soluciones basadas en microprocesadores.

Aunque en un inicio las FPGA ya eran reconfigurables, los fabricantes de FPGA añadieron a estos dispositivos la capacidad para reconfigurarse parcialmente, lo que se conoce como reconfiguración parcial [59]. Tradicionalmente, si se deseaba cambiar el diseño de la FPGA se necesitaba parar el dispositivo y cargar la nueva configuración con la consecuente pérdida del diseño anterior (esto se conoce como reconfiguración estática). Esto limita su uso si queremos que la FPGA se comporte como acelerador de algoritmos cuando la naturaleza de las tareas a ejecutar es dinámica. Además si se quiere actualizar la configuración del diseño es necesario enviar la configuración completa, lo que significa una gran cantidad de megabytes para diseños que contengan varios miles de puertas. Con la llegada de los dispositivos FPGA modernos, a partir de la familia Virtex de Xilinx [1], ya es posible reconfigurar parte del dispositivo sin tener que parar la FPGA y restablecer la configuración.

Estos dispositivos permiten nuevas posibilidades en el diseño de circuitos evolutivos, tolerancia a fallos, etc. Al mismo tiempo que se ampliaba la capacidad de reconfiguración de las FPGA también ha crecido su densidad, y del mismo modo ha aumentado el tamaño y número de funciones que es posible implementar. En la actualidad, además de la lógica programable, las FPGA integran nuevos componentes como procesadores,memorias, unidades aritméticas y mecanismos de comunicación a alta velocidad con elementos externos. Todo este avance tecnológico ha permitido

la implementación en un único dispositivo reconfigurable de sistemas complejos, que junto con un incremento del interés en el hardware reconfigurable, ha dado lugar a diferentes tipos de sistemas:

- System on Chip (SoC): Circuito integrado formado por diversos módulos VLSI con distinta funcionalidad que interconectados entre sí ofrecen una funcionalidad especí?ca para una aplicación.
- System on Programmable Chip (SoPC): Se aplica este término específicamente cuando el dispositivo utilizado para realizar el SoC es reconfigurable. En los SoPC no se utiliza la capacidad de reconfiguración dinámica que puedan disponer estos integrados, sino que únicamente las facilidades que ofrecen estos dispositivos en la fase de desarrollo y posteriores actualizaciones del sistema.
- Configurable System on Chip (CSoC) [10]: Mediante este término se definen los sistemas SoC en los que se hace uso de la capacidad de reconfiguración de los mismos para aplicaciones de computación reconfigurable. Pueden incluirse bajo la denominación CSoC tanto los sistemas que admiten diferentes configuraciones estáticas según ciertos condicionantes, como los que utilizan la reconfiguración parcial dinámica para modi?car en tiempo de ejecución una sección hardware.
- Multiprocessor-configurable-System-on-Chip (MCSoC): Se aplica esta definición a los sistemas CSoC que incluyan varias unidades procesador funcionando de forma simultánea.

### 5.2.1 Configurable System-on-Chip

A finales de los 90 los fabricantes de FPGA comenzaron a introducir microprocesadores en las FPGA para el diseño de SoC. Algunos dispositivos incluyen uno o más microprocesadores hard-core implementados directamente en el chip, junto a eficientes mecanismos para la comunicación entre el microprocesador y la lógica reconfigurable de la FPGA. Más recientemente, Altera desarrolló los dispositivos Excalibur [21] empleando un procesador ARM9 en una FPGA de un millón de puertas y Xilinx ofrece el dispositivo Virtex-II Pro [2] que incorpora uno o más procesadores PowerPC en una FPGA con decenas de millones de puertas.

Mientras las plataformas microprocesador hard-core/FPGA ofrecían un excelente encapsulado y ventajas de comunicación, se comenzó a emplear soluciones basadas

en soft-cores, es decir, microprocesadores que los diseñadores pueden implementar usando FPGA estándar. A estos procesadores se los conoce como Soft-Core Processors (SCP) y ofrecen principalmente las ventajas de la flexibilidad y bajo coste. Muchos fabricantes de FPGA ofrecen este tipo de diseños. Altera dispone del procesador NIOS y el más reciente NIOS II [22]. Xilinx, por su parte, presenta los procesadores PicoBlaze y MicroBlaze [100]. Paralelamente a las SCPs comerciales, también se encuentran disponibles algunos de libre distribución que son ampliamente utilizados, como LEON de Gaisler Research [78] y OpenRISC de OpenCores.org [91]. Los SCPs ofrecen a los diseñadores una enorme flexibilidad durante el proceso de diseño, permitiendo configurar el procesador para adaptarlo a lo que necesitan en sus sistemas y rápidamente integrar el procesador en cualquier FPGA. Por ejemplo, es posible añadir instrucciones a medida o incluir/eliminar unidades operacionales de la arquitectura del procesador. Al contrario que las FPGA que disponen de microprocesadores hard-core, los procesadores soft-core permiten incorporar un número variable de procesadores en una misma FPGA dependiendo de las necesidades de la aplicación. Mientras que algunos sistemas embebidos requieren de unos pocos procesadores, otros diseños pueden incluir hasta 64 procesadores.

Desafortunadamente, los procesadores soft-core implementados en FPGA presentan ciertas desventajas como un alto consumo y un rendimiento inferior comparado con los procesadores hard-core. Para reducir la relación entre el rendimiento y el consumo, un diseñador puede emplear el particionamiento hardware/software. Este particionamiento consiste en dividir una aplicación que inicialmente se ejecuta completamente en software, en un microprocesador, en coprocesadores hardware. Algunos trabajos realizados sobre MicroBlaze [60] han demostrado que puede ser comparable y competitivo con los microprocesadores hard-core existentes, mientras mantiene toda las ventajas de la flexibilidad asociada a los procesadores soft-core. Esta arquitectura es la que se ha utilizado para la implementación de los algoritmos en esta tesis.

## 5.3 Sistemas Criptográficos basados en CSoC

Los dispositivos reconfigurables presentan varias ventajas para la implementación de algoritmos de criptografía [99, 26]:

- Agilidad de los Algoritmos: este término se refiere al intercambio de algoritmos durante la ejecución de la aplicación. Se puede observar que la mayoría de los protocolos de seguridad, como Secure Socket Layer (SSL) [31] e Internet Protocol Security (IPSEC) [50], son independientes del algoritmo y permiten múltiples algoritmos de cifrado.
- Actualización de los Algoritmos: desde el punto de vista criptográfico la actualización de algoritmos es necesaria porque un algoritmo actual puede ser roto (por ej. DES), que expire un estándar (por ej. DES) o que se cree un nuevo estándar (por ej. AES). Suponiendo algún tipo de conexión a redes como Internet, los dispositivos equipados con FPGA pueden actualizarse con la nueva configuración. Esta posibilidad diferencia claramente este tipo de sistemas de los ASIC, en los que la actualización es prácticamente imposible.
- Eficiencia de la Arquitectura: en ciertos casos las arquitecturas hardware puede ser mucha más eficientes si son diseñadas para un conjunto específico de parámetros. Estos parámetros en los algoritmos de criptografía pueden ser por ejemplo la clave, el coeficiente empleado, el tamaño de bloque, etc. En general, cuanto más específico un algoritmo es implementado, más eficiente puede llegar a ser. Una implementación muy eficiente del algoritmo International Data Encryption Algorithm (IDEA) basada en claves fijas fue presentada en [92]. Con claves fijas, la principal operación del algoritmo IDEA se convierte en una multiplicación por constante que es mucho más eficiente que una multiplicación estándar.
- Eficiencia de Recursos: la mayoría de los protocolos de seguridad son híbridos, por ej. IPSEC, SSL, Transport Layer Security (TSL) [29]. Esto implica que un algoritmos de clave pública se emplea para la transmisión de la clave de sesión. Después de que la clave ha sido establecida se emplea un algoritmo de clave privada para el cifrado de los datos. Ya que los algoritmos no se emplean simultáneamente, el mismo dispositivo FPGA se puede emplear mediante reconfiguración en tiempo de ejecución.
- Modificación del Algoritmo: existen aplicaciones que requieren modificaciones de los algoritmos de criptografía estándar, por ej., usando S-boxes o permutaciones propietarias. Estas modificaciones son fáciles de implementar mediante hardware reconfigurable.
- Rendimiento: los procesadores de propósito general no están optimizados para

una ejecución rápida, especialmente en el caso de los algoritmos de clave pública [92, 8], porque las instrucciones para el manejo de operaciones de aritmética modular emplean operandos muy grandes. Las operaciones aritméticas modulares incluyen por ejemplo exponenciación para RSA [79] y multiplicación, raíz cuadrada, inversión, y suma para CCE [66]. Aunque típicamente son más lentas que las implementaciones en ASIC, las implementaciones en FPGA tienen el potencial de ejecutarse más rápido que las implementaciones software.

- Eficiencia de Coste: los dos factores a tener en cuenta respecto al coste cuando se analiza la eficiencia de las FPGA son el coste de desarrollo y el coste de los dispositivos. El coste para el desarrollo de un determinado algoritmo es mucho menor que el caso de los ASIC porque se puede emplear la estructura de la FPGA (por ej. look-up tables), y porque es posible probar el dispositivo tantas veces como sea necesario sin coste añadido. Esto permite disponer del diseño en un tiempo corto, lo que actualmente supone una enorme reducción de coste. Los precios de cada dispositivo no son significativos con respecto a los costes de desarrollo. Sin embargo, para un alto volumen, las soluciones ASIC suelen ser una opción más eficiente.
  La mayor parte de los trabajos realizados en el ámbito de las aplicaciones criptográficas sobre FPGA se basan principalmente, en la capacidad de acelerar el rendimiento empleando soluciones hardware puras sobre FPGA frente a procesadores de propósito general [37]. Sin embargo, los dispositivos reconfigurables actuales disponen de suficiente cantidad de lógica para incluir sistemas completos basados en uno o varios procesadores (hard-core o soft-core), los cuales pueden disponer de diferentes arquitecturas o incluso operar a diferentes velocidades de reloj, junto con una gran cantidad de lógica. Esta capacidad de incluir procesadores y lógica programable en un mismo dispositivo reconfigurable permite diseñar verdaderas soluciones on-chip que se conocen como CSoC [9]. Los CSoC ofrecen un espacio de diseño en cuyos extremos se encuentra la solución software pura ejecutada en procesadores de propósito general, que no presenta un excesivo gasto en área, y la implementación hardware pura ejecutándose en lógica reconfigurable, que permite obtener el mejor rendimiento pero requiere de una gran cantidad de área.

## 5.4 Nuestra plataforma de implementación

Como plataforma de implementación en esta tesis se ha desarrollado un CSoC basado en PowerPC y Microblaze sobre una FPGA de la familia Virtex-II Pro para las pruebas con nuestro algoritmo escalable y una FPGA Spartan 3 para la implementación de Rijndael. Para el diseño hardware de los algoritmos, utilizamos la lógica de la FPGA y para el diseño software, creamos dos sistema embebidos, uno con el procesador Microblaze y otro con el procesador PowerPC 405 a los cuales le añadimos distintos periféricos ensamblados como indica en la arquitectura de la figura 5.3.

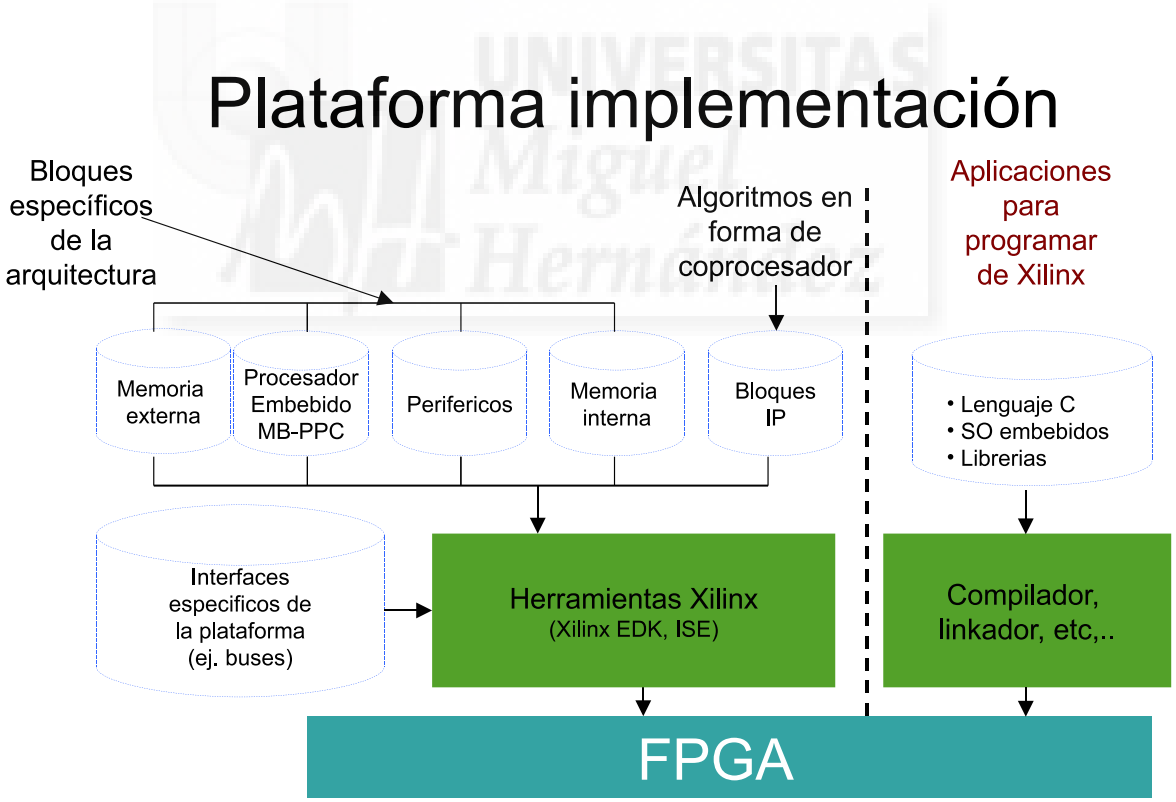


**Figura 5.3:** *Plataforma de implementación*

## 5.5 El procesador Xilinx MicroBlaze

MicroBlaze es un procesador Reduced Instruction set Computer (RISC) sintetizable en hardware reconfigurable de 32 bits tipo big endian, con arquitectura Hardvard desarrollado y mantenido por Xilinx. MicroBlaze dispone de una unidad de enteros segmentada en 3 etapas, y un bloque de registros que incluye 32 registros de propósito general de 32 bits. Al estar especialmente desarrollado para FPGAs de Xilinx, presenta una alta optimización para estas FPGAs tanto en área como en frecuencia de operación. Un diagrama de la arquitectura del procesador se puede ver en la figura 5.4. MicroBlaze se distribuye con el Xilinx Embedded Development Kit (EDK) como una netlist parametrizable [3].

- **Modelo de programación**: MicroBlaze tiene su propio set de instrucciones Instruction Set Architecture (ISA) especialmente diseñado para la arquitectura de este procesador. Presenta dos formatos diferentes de instrucciones y dos modos de direccionamiento: inmediato y desplazamiento. El conjunto de instrucciones incluye instrucciones para multiplicación y división, cuyas unidades operacionales pueden ser implementadas en hardware opcionalmente. La multiplicación, que presenta una latencia de 3 ciclos de reloj cuando se implementa por hardware, puede ser solo implementada si la FPGA dispone de multiplicadores embebidos. La división tiene, si se implementa en hardware, una latencia de 34 ciclos de reloj. Las instrucciones de salto necesitan de 1 ciclo de retardo hasta que la condición se comprueba, y el mecanismo de predicción siempre considera que los saltos son efectivos [3]. El ISA incluye además, instrucciones de lectura y escritura, bloqueantes o no bloqueantes, para el bus Fast Simplex Link (FSL) [4] que permite una rápida comunicación con una o varias unidades hardware a medida.
- **Sistema caché:** La arquitectura caché es Hardvard, y tanto la caché de datos como la caché de instrucciones pueden variar de tamaño entre 2 y 64 Kbytes, con 4 bytes por línea de la caché. Las cachés son de correspondencia directa y soportan bloques individuales en cada línea. Al ser una caché de correspondencia directa, bloquear una línea puede dejar otras direcciones de memoria bloqueadas desde la caché, lo que puede afectar negativamente al rendimiento. La caché de datos opera como una cache de escritura directa (write-through) e implementa asignación en escritura (allocate-on-write). La caché de Micro-

Blaze está limitada para cachear únicamente un subespacio continuo de la memoria total.

- **Interfaz del sistema:** Las interfaz del sistema MicroBlaze consiste de los buses denominados: Local Memory Bus (LMB), IBM CoreConnect On-Chip Peripheral Bus $v2{,}0$ (OPB) y FSL. La unidad de enteros se comunica con la memoria Block RAM mediante un acceso de un ciclo a través del bus LMB. El OPB, bus multi-master y multi-slave, provee un mecanismo de interfaz tanto con memorias como con periféricos internos o externos. El bus FSL es accesible a nivel de ISA, y permite una transferencia punto a punto de datos con una latencia de 2 ciclos. MicroBlaze se puede conectar a un gran número de tipos de memoria incluyendo Intel StrataFlash, Static Random Access Memory (SRAM), Synchronous Dynamic Random-Access Memory (SDRAM) y Double Data Rate SDRAM o SDRAM de tasa de datos doble (DDR SDRAM); además de a un amplio conjunto de periféricos: Xilinx Micro Debug Mode (MDM) para facilitar el debug, Ethernet MAC, Floating Point Unit (FPU) (MicroBlaze 4,00 la incluye por defecto), UARTs, Timers, etc.
- **Interfaz para coprocesador:** Aunque no se presenta como una verdadera interfaz para coprocesador, el FSL provee un mecanismo para el envío y recepción de datos desde/hacia los registros a alta velocidad.

MicroBlaze puede emplear hasta 16 buses FSL, 8 de salida y 8 de entrada, cuyo ancho es parametrizable (8, 16 o 32 bits). Además dispone de un bit adicional de control que puede ser usado, por ejemplo, para diferenciar si los bits son comandos o datos. La conexión con los periféricos FSL se realiza mediante los buses FSL, los cuales emplean un mecanismo First in, first out (FIFO) como buffer de datos y hacen posible que el procesador y los periféricos operen en diferentes dominios de reloj. La profundidad de la FIFO también es parametrizable, desde 1 a 8192 palabras (hasta 128 si se emplean diferentes relojes para lectura y escritura). Están disponibles dos instrucciones para usar el FSL que son get y put: get para cargar los registros con datos de la interfaz FSL, put para escribir el contenido de los registros en un determinado canal FSL. Cada instrucción tiene cuatro variantes, dependiendo de si la operación es o no bloqueante y si el bit de control esta a cero o a uno. En todos los casos no bloqueantes la latencia es de dos ciclos, lo que supone una velocidad de transferencia de 200 MB/sec para un MicroBlaze a 100 MHz.

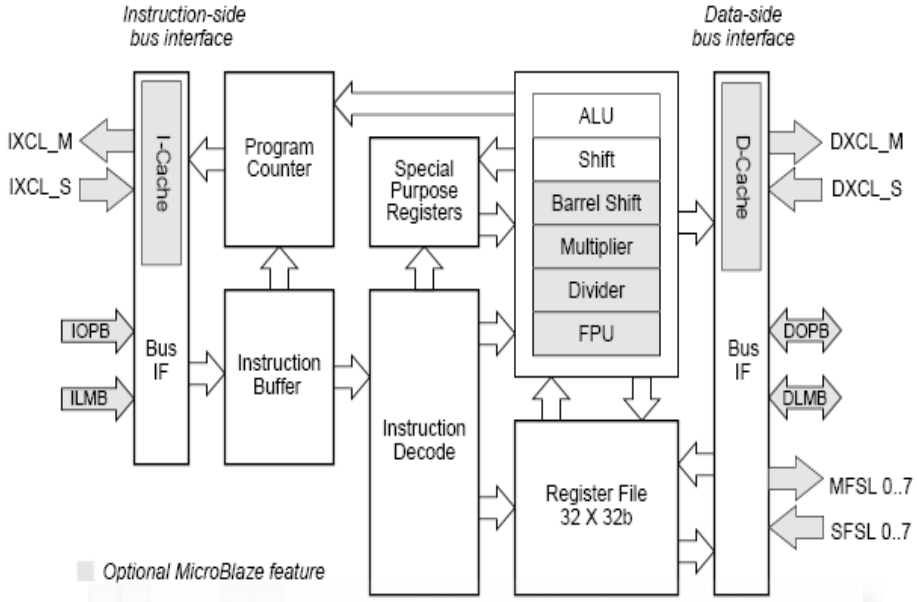


**Figura 5.4:** *Arquitectura Microblaze.*

## 5.6 El procesador PowerPC 405

El PowerPC 405 es una implementación de 32 bits de la arquitectura para sistemas embebidos PowerPC derivada de la arquitectura PowerPC. La arquitectura PowerPC provee un modelo software que asegura la compatibilidad entre toda la familia de microprocesadores, definiendo parámetros que garantizan la compatibilidad de los procesadores a nivel de programa, y permitiendo cierta flexibilidad en el desarrollo de variantes de la arquitectura PowerPC para cumplir ciertos requerimientos de mercado. Específicamente la versión incluida en las FPGAs Virtex-II Pro es el PowerPC 405D5, cuyo diagrama de organización se muestra en la figura 5.5.

Las principales características del PowerPC 405 son:

- Unidad de enteros completamente compatible con la arquitectura PowerPC.
- Arquitectura de 32 bits que incluye 32 registros de propósito general de 32 bits.
- Extensiones de la arquitectura para un completo soporte de aplicaciones embebidas: operaciones little-endian reales, manejo flexible de memoria, instruc-

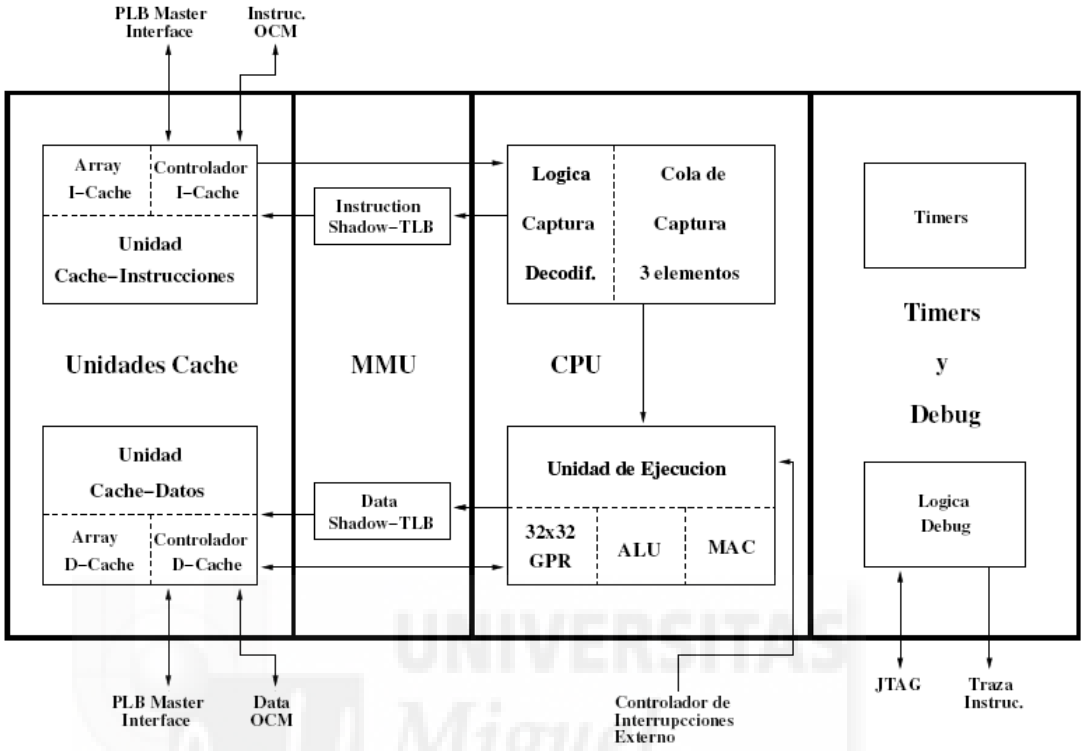


**Figura 5.5:** *Arquitectura PowerPC.*

ciones para computación intensiva, capacidad de debug, etc..

- Característica para un alto rendimiento: predicción de salto estática, pipeline de 5 etapas con un solo ciclo de ejecución para la mayor parte de las instrucciones incluso load y store, multiplicación y división por hardware (4 ciclos para la multiplicación, 35 para la división), mejora del manejo de cadenas (strings) y dobles palabras.
- Cache de datos e instrucciones de 16 KBytes asociativa de 2 vías y 8 bytes (32 bits) por línea.
- Soporte para On Chip Memory (OCM) con rendimiento de acceso idéntico a la caché.
- Modos de direccionamiento: registro indirecto con índice inmediato, registro indirecto con índice y registro indirecto.
- Dispone de Memory Management Unit (MMU) para la gestión de un espacio direccionable de 4 Gb. Soporta tamaños de página de 1, 4, 16, 64 y 256 Kbytes

y 1 y 4 MBytes.

- Soporte para la arquitectura de bus OPB.

UNIVERSITAS
Miguel
Hernández

Capítulo 6

# Propuesta algoritmo escalable

## 6.1 Propuesta algoritmo

Un cuerpo finito es un conjunto finito junto con dos operaciones (adición y multiplicación). Estas operaciones poseen la propiedad asociativa y la conmutativa tanto de la adición como de la multiplicación, distributiva de la multiplicación respecto de la adición y la existencia de una identidad aditiva y una identidad multiplicativa, denotadas por 0 y 1 respectivamente. $GF(2)$ es el cuerpo finito más pequeño formado con los elementos 0 y 1. La adición y la multiplicación realizadas módulo 2, corresponden al XOR lógico y al AND lógico respectivamente.

Cuando el cuerpo finito es fijo, se pueden optimizar los algoritmos para un cuerpo en particular y como resultado, sus implementaciones hardware son bastante eficientes. Sin embargo, este enfoque tiene el inconveniente de la poco flexibilidad que proporciona, ya que, ante distintas situaciones, no podemos cambiar el tamaño del cuerpo finito sin reconfigurar y rediseñar el hardware. Para lograr la escalabilidad, proponemos una arquitectura para un multiplicador de cuerpos finitos, derivado de distintos algoritmos.

El cuerpo $GF(2^m)$ contiene $2^m$ elementos donde $m$ es un entero positivo. La adición en $GF(2^m)$ es la suma de polinomios la cual se implementa en la práctica con la operación XOR entre sus coeficientes. Una multiplicación en el cuerpo consiste en la multiplicación de dos polinomios y la reducción posterior mediante un polinomio irreducible $f(x)$ de grado $m$ con coeficientes en $GF(2)$. Un polinomio de grado $m$ se dice que es irreducible sobre $GF(2)$ si es mónico y no se puede factorizar como producto de polinomios de grado estrictamente menor que $m$ y coeficientes en $GF(2)$.

Un elemento en $GF(2^m)$ se puede representar mediante un polinomio de grado menor o igual que $m$. Para su uso en computación, un polinomio $a(x) = (a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0)$ se representa mediante el array de $m$ bits, $a(x) = (a_{m-1}, a_{m-2}, \ldots, a_0)$. En un procesador con un tamaño de palabra de $w$ bits, este array se divide en $s$ palabras, donde $s = \lceil \frac{m}{w} \rceil$.

Por tanto, un elemento de $GF(2^m)$ se almacena en $s$ palabras en procesadores con palabras de $w$ bits. Si el array de palabras se representa mediante $(A_{s-1}, A_{s-2}, \ldots, A_0)$, tenemos las expresiones siguientes:

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = \sum_{j=0}^{s-1} A_j(x) x^{wj}$$

donde

$$A_j(x) = \sum_{i=0}^{w-1} a_{i+wj} x^i$$

La palabra más significativa $A_{s-1}$ se rellena con ceros por la izquierda, si es necesario.

## 6.2 Diseño

Nuestra propuesta de algoritmo escalable, se basa en el algoritmo para la reducción modular orientado a palabras propuesto por Hutter, Großschädl y Kamendje [43], el cual tiene la ventaja de que no utiliza factores de escala ni conversión entre operandos. Sin embargo, impone restricciones sobre la forma del polinomio irreducible, ya que el algoritmo sólo admite polinomios irreducibles de la forma $p(x) = x^m + p_0(x)$, donde $p_0(x)$ es un polinomio de grado menor que $m - w$. En la practica, esta restricción no limita el uso del algoritmo en implementaciones escalables con polinomios de tamaño arbitrario porque los polinomios irreducibles recomendados por el estándar [45] son trinomios o pentanomios con $p_0(x)$ con el menor grado posible. El polinomio irreducible se alínea a la izquierda multiplicándolo por un factor constante $x^d$, donde $d = sw - m$. En el algoritmo de Kamendje, el autor supone que la operación de multiplicación de polinomios la realiza un procesador equipado con esa operación en hardware, de manera que la reducción del polinomio se realiza procesando $w$ bits a la vez. Además, para permitir operaciones a nivel de bit, el algoritmo emplea el principio de reducción incompleta propuesto por [38]. El polinomio de entrada solo se reduce parcialmente a un polinomio de grado hasta

$sw-1$. Hasta conseguir el resultado final, todas las operaciones aritméticas procesan polinomios en su forma reducida parcial. Por lo tanto, sólo se necesita una reducción final a nivel de bits del polinomio parcialmente reducido, a otro polinomio de grado menor que $m$.

Dado que la aritmética de cuerpos finitos no requiere de acarreo en sus operaciones, el método de multiplicación óptimo para la síntesis en FPGA, son los que utilizan operaciones de desplazamiento y suma, por tanto, combinando los métodos que aparecen en [74], el algoritmo de multiplicación de desplazamiento y suma, junto con el algoritmo intercalado de reducción de Kamendje, [43], proponemos el siguiente algoritmo para la multiplicación de dos elementos de un cuerpo finito:

---

**entrada:**
$a(x) = (A_{s-1}, A_{s-2}, \ldots, A_0)$;
$b(x) = (B_{s-1}, B_{s-2}, \ldots, B_0)$;
$f(x) = x^d p(x) = (F_{s-1}, F_{s-2}, \ldots, F_0)$;
**salida** $r(x) = a(x)b(x) \bmod (x^d p(x))$
donde $r(x) = (R_{s-1}, R_{s-2}, \ldots, R_0)$

$C(x) = 0$; $E(x) = 0$; $R(x) = 0$;
**for** $i = s-1$ **downto** $0$
  $X(x) = (R_{s-1}(x) \hookleftarrow 0^w) \oplus (A_i(x) \otimes B_{s-1}(x))$;
  $E(x) = X_{[2w-1\ldots w]}(x)$;
  $T(x) = X(x) \oplus (F_{s-1}(x) \otimes (E(x)) \oplus (C(x) \hookleftarrow 0^w)$;
  $C(x) = T_{[w-1\ldots 0]}(x)$;
  **for** $j = s-2$ **downto** $0$
    $X(x) = (R_j(x) \hookleftarrow 0^w) \oplus (A_i(x) \otimes B_j(x))$;
    $T(x) = X(x) \oplus (F_j(x) \otimes (E(x)) \oplus (C(x) \hookleftarrow 0^w)$;
    $R_{j+1}(x) = T_{[2w-1\ldots w]}(x)$;
    $C(x) = T_{[w-1\ldots 0]}(x)$;
  $R_0(x) = C(x)$;
**return** $R(x)$;

---

En el algoritmo anterior, $\oplus$ denota la operación XOR, $\otimes$ denota la operación de multiplicación de polinomios de grado $w$ y $C(x) \hookleftarrow 0^w$ significa que al polinomio al

que afecta, en este caso $C(x)$, se le añaden por la derecha tantos 0 como indique $w$. $C(x)$ y $E(x)$ tienen una longitud de $w$ bits, $T(x)$ y $X(x)$ tiene una longitud de $2w$ bits.

Por tanto, si $T(x) = t_{2w-1}x^{2w-1} + t_{2w-2}x^{2w-2} + \cdots + t_0$ entonces

$$\begin{aligned} T_{[2w-1\ldots w]}(x) &= t_{2w-1}x^{w-1} + t_{2w-2}x^{w-2} + \cdots + t_w \\ T_{[w-1\ldots 0]}(x) &= t_{w-1}x^{w-1} + t_{w-2}x^{w-2} + \cdots + t_0. \end{aligned}$$

Analogamente para $X(x)$. El resultado $R(x)$ se va construyendo una vez calculados los resultados parciales en $R_{j+1}(x)$

### 6.2.1 Ejemplo funcionamiento para $\mathbf{F_{2^8}}$

Vamos a desarrollar un pequeño ejemplo que muestre el funcionamiento del algoritmo.

**Ejemplo 6.1:** Primero fijamos el polinomio irreducible $f(x)$ de 8 bits, extraído de [84]. El polinomio irreducible es $f(x) = x^8 + x^4 + x^3 + x^1 + 1$, cuya representación en binario es $100011011_2$ y en hexadecimal $11B_{16}$.

Para comprobar la validez del resultado, utilizamos el método propuesto en [72].

Para unos valores de $A = 57_{16} = 01010111_2$ y $B = 83_{16} = 10000011_2$ el resultado correcto es $C1_{16} =$.Según este método, para calcular el resultado se obtiene el logaritmo de los operandos, $L(57) = 62$ y $L(83) = 50$, con estos valores obtenemos la suma en hexadecimal de los dos $62 + 50 = B2$ y finalmente, para obtener el resultado, tomamos la exponencial de $B2$ que es $C1$.

Vamos a ver a continuación si el algoritmo propuesto funciona correctamente y nos devuelve el mismo resultado.

Sabemos que $m = 8$ y tomamos un tamaño de palabra de $w = 4$ por lo tanto tendremos que partir las cadenas de bits en:

$$s = \left\lceil \tfrac{8}{4} \right\rceil \text{ palabras} \qquad \blacksquare$$

Por tanto $A(x) = [5, 7]$, $B(x) = [8, 3]$ y $F(x) = [1, 1, B]$. Comenzamos con el algoritmo.

En la línea 11 tenemos $R_1 \hookleftarrow 0^w$, esto es añadir 4 bits por la derecha con lo que el resultado de esta operación sería 00 ya que de momento $R_1 = [0]$, $A_1(x) \otimes B_1(x)$ es la multiplicación bit a bit con suma XOR. El resultado es 28, por lo tanto $X(x)$ será la suma XOR de 28 y 00 $X(x) = [2, 8]$.

De la línea 12 obtenemos que $E(x) = X_{[7...4]}(x)$ así que $E(x) = X_1(x) = [2]$.

De la línea 13 $F_1(x) = 1$; $C(x)x^w = 0$ solo queda hacer la suma de $X(x)$ con $E(x) \otimes F_1(x); T(x) = [2, A]$.

De la línea 14 $C(x) = T_{[w-1...0]}$; $C(x) = T_0(x) = [A]$.

Pasamos ahora al bucle for de la línea 15 que se encuentra anidado dentro del primero, en este caso $j = s - 2$, así que $j = 0$ solo tendremos que realizar una vez este bucle. Los pasos a realizar son los mismos que los anteriormente hechos así los valores que obtendríamos son los siguientes: 16: $X(x) = 0 \oplus (5 \otimes 3) = [F]$;
17: $T(x) = F \oplus (B \otimes 2) \oplus (A0) = [B, 9]$;
18: $R_1(x) = [B]$;
19: $C(x) = [9]$;
20: $R_0(x) = C(x) = [9]$;

Así finalizamos los dos bucles, ahora $i = 0$ así que:
11: $X(x) = (B0) \oplus (7 \otimes 8) = [8, 8]$;
12: $E(x) = [8]$;
13: $T(x) = 88 \oplus (1 \otimes 8) \oplus (90) = [1, 0]$;
14: $C(x) = [0]$;
15: $j = 0$
16: $X(x) = (90) \oplus (7 \otimes 3) = [9, 9]$;
17: $T(x) = (99) \oplus (B \otimes 8) \oplus 0 = [C, 1]$;
18: $R_1(x) = [C]$;
19: $C(x) = [1]$;
20: $R_0(x) = [1]$;

Aquí terminaría la ejecución del algoritmo y podemos comprobar como el resultado, compuesto por $R_1(x)$ y $R_0(x)$ es igual al obtenido anteriormente, $R(x) = C1$, así podemos comprobar que el funcionamiento del algoritmo es correcto. Haciendo calculos manuales hasta $m = 16$ se ha comprobado el funcionamiento del algoritmo. Para valores hasta $m = 256$, el algoritmo se ha simulado en VHDL y comparado los resultados con los resultados obtenidos de la ejecución de los algoritmos software propuestos por [80]

## 6.3 Implementación hardware

El algoritmo tiene la estrucutura hardware que muestra la figura 6.1.

Aqui, $X_j$ es el resultado de la operación $(R_j(x)x^w) \oplus (A_j(x) \otimes B_j(x))$ guardado

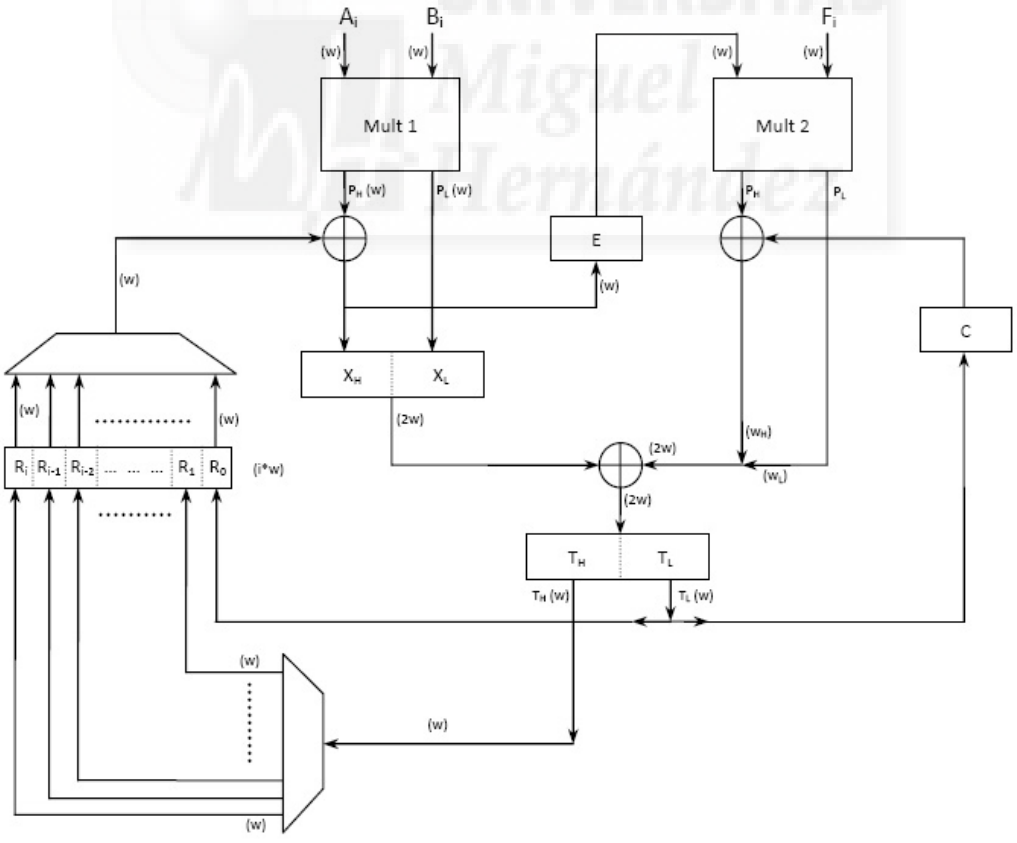


**Figura 6.1:** *Arquitectura hardware del multiplicador*

en el registro $X$, $E_i$ son resultados intermedios, que se guardan en el registro $E$ y $R_j$ es la salida del algoritmo.

El multiplicador tiene dos multiplicadores en paralelo de polinomios de tamaño $w$-bit cada uno etiquetados como $Mult1$ y $Mult2$.

Las entradas $A_i$, $B_i$ y $F_i$ llegan de 3 registros de desplazamiento, $A$, $B$ y $F$ de $m$ bits divididas en $s$ palabras de $w$ bits que representan al multiplicador, el multiplicando y el polinomio irreducible respectivamente. Para la salida del algoritmo, usamos otro registro de desplazamiento, etiquetado como $R$, que contiene los productos parciales obtenidos en los distintos pasos de le ejecución del algoritmo. Estos 4 registros son externos al multiplicador. El multiplicador procesa $w$ bits de $A_i$, $B_i$ y $F_i$ al mismo tiempo.

También utilizamos 4 registros auxiliares, internos al multiplicador, $C$, $E$, $X$ y $T$. Los registros $C$ y $E$, los cuales tienen un tamaño de $w$ bits, se utilizan para reducir los productos parciales. Los registros $X$ y $T$, tienen un tamaño de $2w$ bits. Para computar el resultado parcial en el registro $R$, utilizamos un demultiplexor de $w$ bits de una entrada y $s-1$ salidas, y un multiplexor de $s$ entradas de $w$ bits y una salida de $w$ bits.

Como alternativa al método clásico de multiplicación de polinomios, el cual tiene un retardo de X puertas, vamos a utilizar el algoritmo de Karatsuba-Ofman [48]. Este autor describe un método para la realización de la multiplicación en formato divide y vencerás el cual permite dividir una multiplicación de $n$ bits en varias de $\frac{n}{2}$ bits, lo cual reduce el número de puertas de los multiplicadores si $m \geq 8$ bits. Utilizando este método en $Mult1$ y $Mult2$, el tamaño del multiplicador puede verse en la tabla 6.1.

| Multiplier | #AND | #XOR | #MUX | #DMUX | #FF |
|---|---|---|---|---|---|
| Okada[73] | $2w^2$ | $2w^2$ | $w$ | $w$ | $6w$ |
| Hybrid [76] | $2sw^2$ | $s(2w^2+w-2)$ | – | – | $3sw$ |
| Super Serial[76] | $2w$ | $2w$ | – | – | $3(w+1)$ |
| nuestra propuesta | $\frac{3}{2}w^2$ | $\frac{3}{2}w^2+4w$ | $m$ | $m-w$ | $6w$ |

**Tabla 6.1:** *Área según número de puertas lógicas*

El camino crítico lo determinan los multiplicadores paralelos $Mult1$ y $Mult2$. A medida que aumentamos el tamaño de la palabra $w$, también aumenta el número de puertas AND y XOR (ver Tabla 6.2)

| Multiplier | #AND | #XOR | #MUX |
|---|---|---|---|
| Okada[73] | $w$ | $w+2$ | $w$ |
| Hybrid[76] | 1 | $w+1$ | – |
| Super Serial[76] | 1 | 2 | – |
| nuestra propuestak | 1 | $w+1$ | – |

**Tabla 6.2:** *Retardo máximo*

### 6.3.1 Análisis complejidad espacial y temporal

Puesto que estamos trabajando en $GF(2^m)$, dividiremos los datos de entrada en palabras de $n$ bits, de tal forma que $p = m/n$. Esto quiere decir que si hemos de multiplicar $A \times B$, realizaremos productos de $n$ bits que nos van a ir proporcionando de forma sucesiva $n$ de los $m$ bits que conformaran el resultado $R$, $R_i = A_i B_i$ con $i \in 0, \ldots p-1$

Para materializar los multiplicadores que van realizando estos productos podemos utilizar el algoritmo clásico de lápiz y papel. De esta forma tendremos que los productos parciales, que serán $n^2$ bits, nos darán lugar a que se requieran $n^2$ puertas AND mientras que la sumas de estos $n$ productos parciales requerirán $(n-1)^2$ puertas XOR. Puesto que el diseño completo está formado por dos multiplicadores de este tipo, el número de puertas se verá duplicado en ambos casos. Además, el circuito tiene tres grupos de puertas XOR que operan sobre las salidas de los multiplicadores. Dos de ellos necesitan $n$ puertas, mientras que el otro emplea $2n$. Como resultado obtenemos que la complejidad del circuito propuesto compuesto por multiplicadores que implementan el algoritmo de lápiz y papel será: *puertas* AND: $2n^2$ *puertas* XOR: $2n - 12 + 4n = 2n^2 + 2$.

Si el número de bits con que se trabaja es una cantidad importante, una alternativa al método clásico del algoritmo de lápiz y papel es el algoritmo de Karatsuba-Ofman [49]. Este método describe un algoritmo del tipo divide y vencerás que puede

reducir el número de puertas en este caso. Aquí, los productos de $n \times n$ bit se dividen en otros de $(n/2) \times (n/2)$ bit. Esto nos lleva a una mayor eficiencia en cuanto a espacio de este multiplicador si trabajamos con valores de $n$ de 8 bit o superiores (ha de ser potencia de 2).

#### 6.3.1.1 Frecuencia máxima de funcionamiento.

Para determinar la frecuencia máxima de funcionamiento, determinaremos el camino crítico del multiplicador. En el algoritmo de lápiz y papel multiplicamos $n$ bit por $n$ bit. Los productos se pueden realizar todos simultáneamente en paralelo, por lo que el retardo introducido será el de una puerta ($t_a$). Posteriormente tendremos que operar los $n$ sumandos. Si hemos de cuantificar con puertas XOR de dos entradas, el retardo máximo (camino crítico) será el sumando correspondiente a $n$ bits, que necesitará atravesar $n-1$ puertas XOR. En total, el período mínimo será: $T = t_a + n - 1t_x$.

Si utilizamos el multiplicador de Karatsuba tendrémos tres sumandos, que perfectamente podremos realizar previamente en paralelo. Lógicamente, el retardo producido se corresponderá con el que introduzca el más lento de estos tres caminos. El primero será el producto de $n \div 2$ por $n \div 2$ bits. Podemos realizar un razonamiento similar al del algoritmo de lápiz y papel y llegaremos a la conclusión de que requerirá un tiempo: $t1 = t_a + n^2 - 1tx$. El tercero es idéntico, por lo que el retardo será el mismo: $t3 = t1 = t_a + n^2 - 1t_x$. En cuanto al segundo, requiere dos restas de $n/2$ bits que se realizan en paralelo ($t_x$), su posterior producto (retardo idéntico al de los otros dos productos) y finalmente la suma con el primer producto y con el tercero. Esto nos indica que el camino crítico irá por esta vía. El tiempo total será: $t2 = t_x + t_a + n^2 - 1t_x + 2t_x = t_a + n^2t_x + 2t_x$. Y finalmente hemos de sumar los tres productos, luego el retardo total será: $T = t_a + n^2t_x + 2t_x + 2t_x = t_a + n^2t_x + 4t_x$ y la frecuencia máxima: $fmax = 2t_a + (n+8)t_x$.

El retardo del multiplicador Mult2, sumado al de las puertas XOR a las que se encuentra conectado, conforma el retardo máximo. En concreto, el camino crítico será igual a la suma de las puertas AND y XOR, $w+1$. El número de ciclos de reloj que se requiere para realizar un producto de $m * m$ bits es de $s^2$. Para determinar el rendimiento de nuestro multiplicador lo hemos comparado con otros multiplicadores también basados en Linear Feedback Shift Register (LFSR). Para compararlo con el

multiplicador de Okada, haremos que trabaje con un tamaño de palabra fijo; en la comparación con el híbrido le asignaremos una palabra de $w$ bits y, finalmente, en la comparación con el Super Serial, le asignaremos un tamaño de palabra de 1 bit. La ventaja de utilizar Karatsuba en Mult1 y Mult2 se refleja en una disminución del número de puertas AND y XOR que se requieren. En el multiplicador de Okada, el tamaño del multiplexor y del demultiplexor es $s$ veces menor, pero no intervienen en el camino crítico. En el híbrido, se comparte el camino crítico pero se reduce $s$ veces el hardware a costa de aumentar el tiempo necesario para realizar un producto. El Super Serial presenta un retardo muy reducido pero el tiempo necesario para realizar una multiplicación es proporcional a $m$. En resumen, en nuestro multiplicador conseguimos un equilibrio en la relación entre el hardware utilizado y el tiempo de proceso de una multiplicación.

## 6.4 Implementación sistemas

En este apartado, se propone estudiar el rendimiento del algoritmo escalable en una plataforma CSoC. Para ello, seleccionamos Microblaze y se pretende evaluar la capacidad de Microblaze para mejorar su arquitectura interna mediante la adición de unidades hardware específicas, ya sea dentro de la propia arquitectura del procesador, o mediante la interfaz FSL que permite una conexión de coprocesadores específicos. Además, los CSoC están compuestos de un conjunto de periféricos estándar para el acceso a los recursos del sistema como memorias, puertos de E/S, etc. El resto de la lógica reconfigurable queda disponible para el diseño de hardware a medida en forma de coprocesadores.

Como complemento al sistema anterior, también se ha desarrollado un sistema basado en PowerPC, el cual puede ser considerado como un sistema embebido tradicional, para su comparación con los sistemas basados en CSoC. Este sistema permite realizar una comparación más adecuada de los procesadores hard-core frente a los soft-core, al ejecutarse ambos sistemas sobre el mismo tipo de dispositivos reconfigurables. A continuación se describen los 3 sistemas implementados para las pruebas de nuestro algoritmo, uno basado en SCPs con Microblaze, uno con Microblaze y coprocesador y el tercero con un procesador hard-core PowerPC. También se detallan las diferentes mejoras arquitecturales que pueden ser añadidas a cada sistema, y los

diferentes periféricos que acompañan al procesador. Para el diseño de los sistemas, se ha empleado la herramienta de Xilinx EDK 8,2 [3].

## Sistema CSoC con Microblaze

El sistema MicroBlaze propuesto para estos experimentos se muestra en la figura 6.2 La memoria Block Random Access Memory, bloque de memoria de acceso aleatorio (BRAM) se emplea para almacenar un pequeño programa bootloader que permite la carga de los diferentes programas en la memoria externa. El controlador de memoria externa es el $OPB-EMC$ configurado para un acceso a la memoria SDRAM de 32 bits de datos. Al sistema se le ha incorporado una periférico $OPB-UARTLite$ para la comunicación externa con un terminal serie, y un $OPB-Timer$ para la obtención del tiempo empleado en cada prueba.

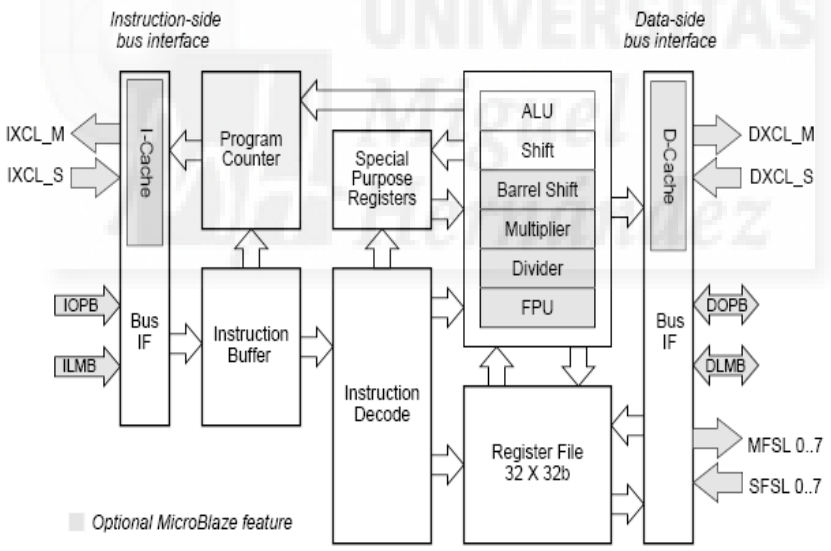


**Figura 6.2:** *Sistema Microblaze*

La arquitectura interna del sistema MicroBlaze puede ser mejorada con las unidades hardware de multiplicación, división y rotación (barrel shifter). Además es posible añadir una caché de datos y otra de instrucciones parametrizable en tamaño y forma idéntica, esto es, con organización de correspondencia directa (no se puede variar) y con un tamaño de 8192 bytes y 256 entradas. Este tamaño de caché, en particular la caché de instrucciones, se considera suficiente para mejorar adecuada-

mente la ejecución de todos los algoritmos sin sacrificar demasiado recursos lógicos. Los resultados de implementación para cada una de las arquitecturas se muestran en la misma tabla, y permiten determinar el gasto de recursos que supone cada una de las mejoras en la arquitectura. La implementación de las caches se realiza en BRAM lo que supone un incremento de 32 a 74 bloques. El sistema MicroBlaze más complejo, MBlaze-E, alcanza una frecuencia de operación máxima sobre el dispositivo Virtex-E de 50 MHz, frecuencia que se ha elegido para realizar todas las pruebas.

## Sistema CSoC con Microblaze y coprocesador

Con el objetivo de acelerar el rendimiento obtenido en el primer conjunto de experimentos,se procede a ampliar la funcionalidad de MicroBlaze conectando un core específico de multiplicación de polinomios utilizando el bus FSL.

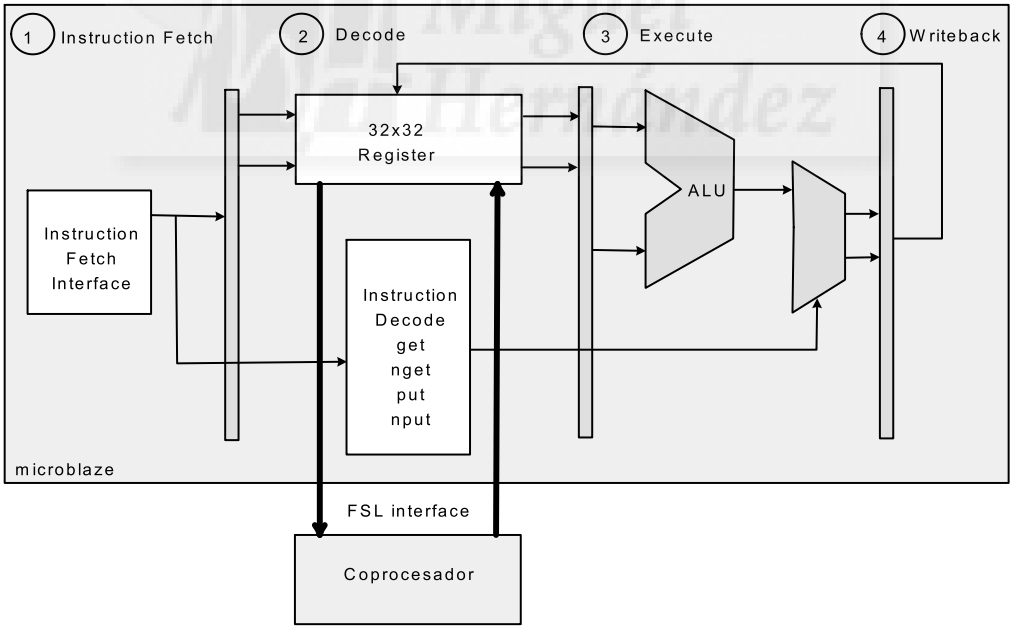


**Figura 6.3:** *Sistema Microblaze+Coprocesador*

## Sistema PowerPC

Al igual que en el caso de MicroBlaze, presenta una memoria BRAM para el bootloader, un $OPB - UARTLite$ para la comunicación con un terminal serie y el $OPB - Timer$ para la medición de los tiempo de ejecución. Del mismo modo, se ha empleado el controlador de memoria PLB-EMC para el acceso a la memoria SDRAM de la plataforma.

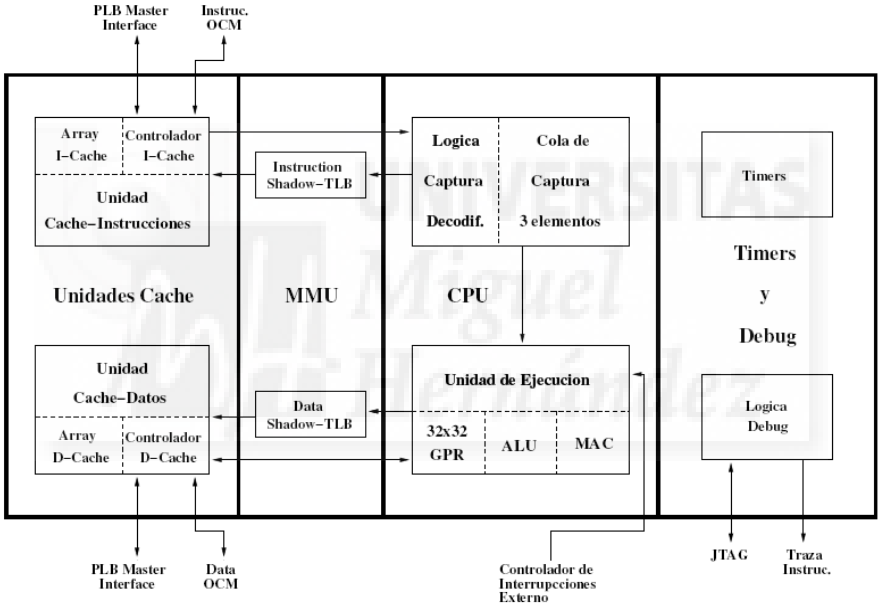


**Figura 6.4:** *Sistema PowerPC*

El PowerPC no presenta ninguna característica de configurabilidad. Al contrario, dispone de un conjunto de unidades operacionales en hardware como multiplicador, divisor, etc. además de la caché de datos y la caché de instrucciones. En el caso de las caches es necesario habilitarlas para que el procesador haga uso de ellas, y han sido configuradas para cachear la región de memoria que comprende la memoria SDRAM externa.

## 6.5 Experimentos

El primer conjunto de experimentos incluye la ejecución de la implementación software del algoritmo escalable sobre las arquitecturas MicroBlaze. Posteriormente, el segundo conjunto de experimentos consiste en emplear el core hardware conectado al bus FSL disponible en MicroBlaze. Y finalmente, el tercer conjunto de experimentos consiste en la ejecución de la implementación software del algoritmo escalable sobre la arquitectura PowerPC.

### Experimento 1. Ejecución en Microblaze

En esta prueba se pretende comprobar el tiempo empleado en ejecutar el algoritmo para los tres polinomios seleccionados para el estudio 160, 191 y 256 bits. Para obtener el tiempo se mide mediante el timer el número de ciclos de reloj empleado y se divide entre la frecuencia máxima obtenida en la síntesis:

$$tiempo = \frac{ciclos}{f_{max}}$$

Los ciclos de reloj devueltos por el timer son los que se muestran en la tabla 6.3

| | **Polinomio 160** | **Polinomio 191** | **Polinomio 256** |
|---|---|---|---|
| **Ciclos de reloj** | 9057804 | 12856146 | 27573763 |

**Tabla 6.3:** *Ciclos de reloj obtenidos con timer*

El tiempo en segundos teniendo en cuenta la ecuación anterior y habiendo obtenido en la síntesis una frecuencia máxima de 102,041 Mhz son los de la tabla 6.4

| | **Polinomio 160** | **Polinomio 191** | **Polinomio 256** |
|---|---|---|---|
| **Tiempo(ms)** | 88,766 | 125,990 | 270,222 |

**Tabla 6.4:** *Tiempo de ejecución en milisegundos*

En resultados vemos que conforme aumentamos el tamaño del polinomio va reduciéndose el rendimiento del algoritmo. Estos tiempos parecen excesivamente altos, así que, vamos a usar las mejoras que incluye MicroBlaze para ver si se reducen estos tiempos.

## Mejoras del sistema

Para intentar optimizar el algoritmo se van a usar algunas de las mejoras que MicroBlaze incorpora, que por defecto se encuentran desactivadas y que habrá que hacerlo de forma manual mediante la modificación de los parámetros del procesador.

Las mejoras que van a usarse son las siguientes:

- **Integer Multiplier**. Este parámetro activa las instrucciones *mul* y *muli*. Teniendo este parámetro desactivado tenemos poco efecto sobre el área ocupada por MicroBlaze. El compilador usa automáticamente la instrucción *mul* si está activado este parámetro.
- **Float Point unit**. Activa una unidad en coma flotante basada en el Standard IEEE-754. Este parámetro activa las instrucciones *fadd*, *frsub*, *fmul*, *fdiv*, y *fcmp*. Activando este parámetro se mejorará significativamente el rendimiento cuando se usen operaciones en coma flotante y además aumentara bastante el tamaño de Microblaze. El compilador usa automáticamente la instrucción *FPU* si el parámetro está activado.
- **I-Cache y D-Cache**. Uso de Caché para datos e instrucciones.
- **Barrel Shifter**. Este parámetro activa las instrucciones *bsrl*,*bsra*,*bsrai* y *bslli*. Activando está opción puede mejorar el rendimiento considerablemente, pero se incrementará el tamaño del procesador. Como en los casos anteriores el compilador usará automáticamente las instrucciones *barrel shift* si este parámetro se encuentra activado.

La tabla 6.5 muestra los resultados de una multiplicación en *ms* para cada una de las opciones.

Observando los resultados obtenidos, tomando como base el tiempo de ejecución del algoritmo sin opciones activadas, podemos ver como el uso de los parámetros *FPU Support* e *I/D Caché* no influyen prácticamente en el rendimiento del sistema pero si aumenta el tamaño de MicroBlaze por lo que estos parámetros no sirven para nuestros propósitos.

Solo nos quedan los parámetros *Hard Multiplier*,*Barrel Shifter* y todas las opciones activadas, teniendo en cuenta que cada uno de los parámetros ocupa un área considerable en MicroBlaze, se desecha la idea de utilizar todas los parámetros conjuntamente. Finalmente con el que obtenemos un rendimiento mayor es con la opción

| | $m$ | | |
|---|---|---|---|
| | 160 | 191 | 256 |
| Ninguna Opción | 85,05 | 136,57 | 186,97 |
| Hard Multiplier | 83,05 | 126,58 | 183,96 |
| Hard Divider | 85,05 | 136,57 | 186,97 |
| Barrel Shifter | 83,05 | 125,57 | 183,96 |
| FPU Support | 85,04 | 136,68 | 186,97 |
| I/D Caché | 85,04 | 136,57 | 186,93 |
| Todas las opciones | 83,04 | 126,56 | 183,96 |

**Tabla 6.5:** *Tiempo en ms para una multiplicación*

*Barrel Shifter*, debido a que la frecuencia máxima obtenida es mayor, pero el que menos ciclos de reloj tarda en ejecutar el algoritmo es con la opción *Hard Multiplier*. Teniendo en cuenta que se tiene un rendimiento casi igual usando la opción *Barrel Shifter* que con la opción *Hard Multiplier* quizás lo más recomendable sería usar sola esta ultima ya que ocupa muy poca área en el procesador.

## Experimento 2. Arquitectura mejorada mediante un módulo especifico para multiplicar polinomios

Una vez encontradas las mejores opciones de configuración del procesador Microblaze, vamos a realizar el "profiling". Esta técnica nos proporciona información del rendimiento del algoritmo, indicándonos cual es la función que consume un mayor tiempo de procesamiento, el porcentaje de cada función respecto al tiempo total de ejecución del algoritmo, y el número de llamadas a cada una de las funciones. La función que más tiempo consume en una multiplicación de dos elementos del cuerpo $GF(2^{160})$ para el polinomio irreducible $f(x) = x^{160} + x^5 + x^3 + x^2 + 1$, es la función `poly_mult`. Esta función realiza la multiplicación de dos polinomios, ocupando un 68,69 % del tiempo total de ejecución. A la vista de estos resultados, sería conveniente usar algún tipo de implementación hardware (o coprocesador)del algoritmo

de multiplicación de polinomios, de esa forma, podríamos acelerar la multiplicación en el cuerpo.

Al tiempo que se adapta el core al bus FSL, es igualmente necesario adaptar los algoritmos en C, sustituyendo las funciones software de multiplicación de polinomios por sus equivalentes hardware. El código resultante es bastante reducido y sencillo, al estar toda la funcionalidad del algoritmo implementada en hardware. De igual modo que con los algoritmos software, el sistema se ha chequeado mediante los test de verificación, para comprobar que las implementaciones hardware cumplen con las especificaciones del algoritmo. Es importante destacar que todos ellos se han ejecutado sobre las distintas arquitecturas del procesador MicroBlaze propuestas para el primer conjunto de experimentos. Los resultados obtenidos para el multiplicador con el coprocesador aparecen en la tabla 6.6.

| | $m$ | | |
|---|---|---|---|
| | 160 | 191 | 256 |
| Ninguna Opción | 5,54 | 8,44 | 17,07 |
| Hard Multiplier | 5,29 | 8,11 | 16,5 |
| Hard Divider | 5,53 | 8,44 | 17,07 |
| Barrel Shifter | 5,28 | 8,1 | 16,29 |
| FPU Support | 5,53 | 8,44 | 17,07 |
| I/D Caché | 5,53 | 8,44 | 17,11 |
| Todas las opciones | 5,28 | 8,1 | 16,51 |

**Tabla 6.6:** *Tiempo de la multiplicación en ms obtenidos con el coprocesador*

## Experimento 3. Arquitectura con PowerPC

Se va a implementar y realizar las mismas pruebas que se han realizado sobre el procesador MicroBlaze, pero en un sistema con un procesador hard-core. En estas pruebas se usará el procesador PPC405 que ya incorpora nuestra plataforma de implementación y para aprovecharlo al máximo se usará una frecuencia de reloj de 300 Mhz, además se incluirán los mismos periféricos (módulo de debug, puerto RS232,

timer, controlador de interrupciones y memoria externa en este caso conectada por el bus PLB). Se usará el mismo código fuente, ya que no es necesario realizar ningún cambio sobre él.

En este caso el tamaño del ejecutable que obtenemos es de 62,68 Kbytes, este tamaño es menor que el obtenido para MicroBlaze, debido a que el ensamblador para PowerPC está más optimizado, obteniendo un menor número de instrucciones. Si comparamos estos resultados con los obtenidos sobre MicroBlaze, lo primero que llama la atención es que dentro de la función `poly_mult`, aunque sigue siendo la que más tiempo consume para los tres polinomios analizados, el porcentaje de tiempos está ahora más repartido entre las funciones `field_add` y `bitstr_lshift`. Lo que puede llevar a pensar que probablemente el algoritmo se ejecute más rápido y que este procesador está mejor optimizado para usar el algoritmo en él, como se verá en la siguiente sección. Podríamos usar, como haremos con MicroBlaze, un coprocesador para acelerar aun más la ejecución del algoritmo aunque el procesador PowerPC de la familia Virtex-II Pro no incluye ninguna interfaz específica para coprocesador. En la tabla 6.7 se aportan los resultados obtenidos:

| | **Polinomio 160** | **Polinomio 191** | **Polinomio 256** |
|---|---|---|---|
| Ciclos de reloj | 2303936 | 3824528 | 8005751 |

**Tabla 6.7:** *Ciclos de reloj obtenidos con timer para PPC*

Ahora una vez se tienen los ciclos de reloj puede obtenerse el tiempo empleado en la ejecución, esto se puede ver en la tabla 6.8

| | **Polinomio 160** | **Polinomio 191** | **Polinomio 256** |
|---|---|---|---|
| **Tiempo(ms)** | 7,679 | 12,748 | 26,686 |

**Tabla 6.8:** *Tiempo de ejecución en milisegundos para PPC*

## 6.6 Resultados

El uso de las técnicas de codiseño hardware-software nos permite acelerar algoritmos diseñados para ejecutarse en un entorno software. Como podemos ver,

utilizando el coprocesador tenemos un rendimiento mucho mayor, por ejemplo para el polinomio de grado 160 el sistema tarda 83 $ms$ para realizar una multiplicación, mientras que con el coprocesador tenemos un tiempo de 5,28 $ms$. Si comparamos los resultados obtenidos para PowerPC con los de MicroBlaze, se aprecia que los ciclos de reloj y el tiempo obtenido para cada uno de los polinomios es mucho menor. La mejora con respecto a MicroBlaze se debe, además de una mayor frecuencia de reloj (PowerPC está trabajando a 300 Mhz, mientras que MicroBlaze lo hace a aproximadamente 100 Mhz) a que se ha utilizado un controlador de memoria PLB_EMC para el acceso a la memoria externa. Esto es posible ya que la memoria SDRAM está configurada para bus de datos de 64 bits, y es por ello que se puede emplear el controlador PLB_EMC en vez del OPB_EMC, lo que supone una clara ventaja con respecto a MicroBlaze. Además PowerPC dispone de varias unidades operacionales en hardware como es el multiplicador hardware lo que en el caso de este algoritmo en el que las multiplicaciones son constantes es una ventaja.

En las tablas 6.9 y 6.10 se incluye y el área ocupada, como se hizo con MicroBlaze, se han quitado el modulo de debug, el temporizador y el controlador de interrupciones. Puede verse como en varios casos es menor el área ocupada que la de MicroBlaze ya que parte del área de la FPGA se utiliza para sintetizar al propio Microblaze.

| | **Slices** | **Slice Flip Flops** | **4 input LUTs** |
|---|---|---|---|
| **Tamaño Total** | 13696 | 27392 | 27392 |
| **PPC** | 1343 | 1262 | 1729 |

**Tabla 6.9:** *Área ocupada por PPC*

| | **GCLKs** | **Block RAMs** | **IOBs** |
|---|---|---|---|
| **Tamaño Total** | 16 | 136 | 556 |
| **PPC** | 5 | 8 | 117 |

**Tabla 6.10:** *Área ocupada por PPC (2)*

La única manera de mejorar los tiempos de proceso del algoritmo en el procesador Microblaze, es ejecutar parte del algoritmo en hardware.

UNIVERSITAS
Miguel
Hernández

Capítulo 7

# Implementación Rijndael en FPGA

## 7.1 Introducción

En el mundo de la criptografía, una clave es un valor numérico generado mediante computación que los algoritmos de cifrado usan para el cifrado y descifrado de datos. Al no ser la clave de longitud fija, cuanto mayor sea ésta mayor seguridad provee. Existen dos tipos de claves [83, 90], privada y pública, por lo que también existen dos métodos de cifrado: de clave privada y de clave pública.

La criptografía de clave privada fue el primero de los métodos criptográficos basados en clave. También se la conoce como criptografía simétrica porque se emplea la misma clave para el procedimiento de cifrado y de descifrado. La clave debe ser compartida por los diferentes participantes de la operación, de modo que en el origen se puedan cifrar los datos mediante la clave propuesta, y en el destino se puedan descifrar los datos con la misma clave. El problema de este método es que si la clave es conocida por otras personas, entonces tendrán acceso a todos los datos.

### Algoritmos de clave privada

Los algoritmos de clave privada o simétrica son comúnmente divididos en cifradores de bloques y cifradores de flujo [83]. En los cifradores de bloque el mensaje es dividido en cadenas, llamadas bloques, de longitud fija y se cifra un bloque cada vez. Ejemplos de cifradores de bloque son DES [69], IDEA [53], y AES [30]. Los cifradores de flujo operan con un único bit de texto sin cifrar al tiempo. En cierto modo, son cifradores de bloque que operan sobre bloques de longitud igual a uno.

Son útiles porque la transformación de cifrado puede cambiar para cada símbolo del mensaje a ser cifrado. En particular, son muy útiles en situaciones donde los errores de transmisión son muy probables, porque no propagan los errores. Además, pueden ser usados cuando los datos son procesados símbolo a símbolo debido a la falta de memoria del sistema o que el buffer es limitado. Un ejemplo de este tipo de algoritmos es RC4 [51].

Es importante destacar que el objetivo al diseñar los modernos cifradores de clave simétrica actuales, como AES, ha sido optimizar los algoritmos para una implementación eficiente tanto en software como en hardware, al contrario que DES, que fue diseñado pensando en una implementación únicamente hardware. Estos criterios de diseño son evidentes si se analiza el rendimiento del algoritmo AES sobre diferentes plataformas. Las operaciones internas del algoritmo AES se pueden dividir en operaciones de 8 bit, lo cual es importante porque la mayoría de los algoritmos de cifrado se ejecutan sobre smart cards, que tradicionalmente emplean CPUs de 8 bits. Además, es posible combinar ciertos pasos para obtener un rendimiento aceptable en el caso de plataformas de 32 bits. Al mismo tiempo, las implementaciones de AES en hardware pueden alcanzar fácilmente los Gbits/s cuando se emplean ASIC o FPGA.

## 7.2 Rijndael

Rijndael es un cifrador de bloques basado en una red de sustitución y permutación, diseñado por Joan Daemen y Vincent Rijmen como algoritmo candidato para el AES [23]. Permite el uso de diferentes tamaños de bloque y de clave (normalmente 128, 192 ó 256 bits). En función de estos tamaños varía el número de rondas del cifrado y del descifrado.El algoritmo presenta una longitud de bloque y clave variables. Actualmente se especifica como usarlo con claves de longitud 128, 192, o 256 bits para cifrar bloques de longitud de 128, 192 o 256 bits (todas las combinaciones de longitud de clave y longitud de bloque son posibles). Tanto la longitud de bloque como la longitud de la clave pueden ser extendidos fácilmente a múltiplos de 32 bits. Rijndael puede ser implementado muy eficientemente en un amplio rango de procesadores y en hardware.

La base matemática en Rijndael es la representación de sus bytes como poli-

nomios en el cuerpo $GF(2^8)$. Durante la ejecución realiza operaciones de suma y multiplicación de polinomios en en $GF(2^8)$.

El algoritmo consta de las siguientes fases:

**Generación de tablas**. En ciertos sistemas en los que el tamaño del código fuente es determinante, se opta por generar en tiempo de ejecución las tablas y reducir el tamaño del fuente. Estas tablas son las empleadas para las sustituciones y para optimizar las multiplicaciones.

**Expansión de clave**. Rijndael parte de que la importancia de la seguridad reside principalmente en la clave, por lo que genera diferentes subclaves a partir de la clave inicial para no tener que repetir el uso de ninguna de ellas en las diferentes rondas de cifrado y descifrado.

**Cifrado**. Para este proceso se repite la utilización de cuatro funciones sobre el bloque a cifrar, durante un número variable de rondas dependiente del tamaño de bloque y de clave. Esta funciones son: AddRoundKey (XOR entre bloque y subclave), MixColumns(multiplicación de las columnas del bloque con un polinomio constante), SubBytes(sustitución de cada byte por su imagen en la tabla Sbox) y ShiftRows (rotación de las filas del bloque). En procesadores de 32 bits, las tres últimas funciones pueden ser optimizadas cambiándolas por la sustitución con cuatro tablas T o para ahorro de memoria con una sola tabla T y cuatro rotaciones de bytes.

**Descifrado**. En el proceso de descifrado se usan las inversas de estas funciones de un modo similar, empleando las mismas subclaves pero en orden inverso. Para procesadores de 32 bits también podemos usar la optimización de las tablas T, pero lógicamente éstas serán diferentes.

En este trabajo, se ha estudiado la implementación de Rijndael en una plataforma CSoC de implementación mixta (hardware/software), utilizando como FPGA, la Spartan3e de Xilinx. Las funciones en software se ejecutan en un procesador Microblaze mientras que las funciones hardware se implementarán en la FPGA. La plataforma se amplia con periféricos de entrada/salida de datos. Se pretende optimizar el rendimiento del algoritmo Rijndael en un sistema hardware/software de bajo coste. Se estudiará el código para convenir qué partes del software deben realizarse en hardware y así ganar en tiempo de ejecución sin tener que ampliar en exceso el tiempo de diseño.

Para la realización del estudio se utiliza un algoritmo optimizado con una tabla $T$, por lo que las rotaciones serán una dificultad añadida al cifrado. Se generarán las tablas en tiempo de ejecución y las subclaves, tanto del cifrado como del descifrado, de una sola vez al comenzar la ejecución.

## 7.3 Diseño de sistemas de prueba

Partiendo de una implementación base (Sistema A), la iremos ampliando y modificando creando varios sistemas más, en cada una de las cuales se ejecutará el algoritmo, hasta 6 en total.

**Sistema A**. El procesador MicroBlaze ofrece multitud de configuraciones. Para poder ejecutar el algoritmo Rijndael será necesario diseñar un sistema básico, a partir del cual se formarán el resto de sistemas. Junto con el procesador se incluye una Universal Asynchronous Receiver-Transmitter (UART), un temporizador y un módulo Debug para el estudio del sistema. El multiplicador hardware de MicroBlaze está activado y el programa se almacenará en una memoria externa de 1 MB de capacidad. Los parámetros que se han ajustado para la elaboración del sistema son las siguientes:

Características generales:

- Frecuencia de reloj de referencia a 50 MHz.
- Frecuencia del bus a 50 MHz.
- Interfaz de debug hardware.
- Memoria ($OPB_EMC$): SRAM $256K$ x 32.
- Timer ($OPBTIMER$).

Interfaces de entrada y salida, que nos permiten activar y configurar una UART (Entrada/Salida serie) de la siguiente forma:

- 57600 baudios.
- Sin bit de paridad
- Sin Interrupciones.

**Sistema B**. El código fuente del Algoritmo Rijndael empleado se basa en la optimización mediante una tabla $T$, por lo que para el cifrado y el descifrado de bloques necesita la rotación de bytes. Esta es una operación muy costosa para su

realización en software, así que el sistema B se ayudará en su ejecución de un coprocesador hardware que acelere el proceso de rotación. El coprocesador no tiene la necesidad de acceder a memoria, sino que actuará con los datos enviados desde el procesador y devolverá la respuesta al mismo. Por ello, y para incrementar la velocidad, se utilizará para la comunicación una conexión FSL, que permitirá recibir y enviar datos directamente desde el hardware. El bus FSL es un canal de comunicación unidireccional punto a punto utilizado para conseguir una rápida comunicación entre elementos de una FPGA. La principal idea de FSL es conectar coprocesadores utilizando una interface muy simple.

**Sistema C**. Otro método para optimizar el código para la rotación de bits, es la activación del hardware barrel shifter del procesador MicroBlaze. Con ello se espera obtener un aumento del rendimiento gracias a que el algoritmo llama de forma directa a las funciones rotadoras. De forma indirecta, el hecho de tener una implementación hardware de la rotación, beneficiará al resto del código puesto que esta operación se empleará en la ejecución de otras pseudo-instrucciones. Para optimizar las funciones de rotación de bytes se activa el rotador de bits hardware que incluye el procesador.

**Sistema D**. En las fases de generación de tablas y de claves se emplea de forma reiterada la multiplicación en campos de Galois $GF(2^8)$. Esta operación, está definida en el código por la función `Bmul`, y será capaz de realizar el producto de 2 bytes de datos, produciendo un resultado también de 8 bits. Con el Sistema D se pretende optimizar la ejecución del código mediante un coprocesador capaz de realizar este producto de forma eficiente. Puesto que se emplean dos datos de entrada y uno de salida, todos ellos de 8 bits y se dispone de un bus de 32 bits, los empaquetaremos todos ellos de forma conjunta. Los datos de entrada se colocarán de forma sucesiva en la parte alta, mientras que la salida vendrá en los ocho bits de menos peso. Se incluye al sistema A un coprocesador encargado de multiplicar 2 bytes, representación de dos elementos del cuerpo campos $GF(2^8)$. Así se pretende optimizar las fases de generación de tablas y de expansión de clave.

**Sistema E**. Este sistema trata de estudiar la posibilidad de conjugar dos de los anteriores que realizan funciones similares. En concreto, se ha introducido el coprocesador hardware para la rotación de bits (`ROTL`) diseñado para el Sistema B, simultáneamente con la habilitación del hardware barrel shifter de MicroBlaze.

**Sistema F**. En este caso, como en el anterior, se han combinado dos opciones de las utilizadas anteriormente para observar si su funcionamiento simultáneo produce alguna mejora. Específicamente, se ha activado de nuevo el hardware barrel shifter y también el coprocesador diseñado para el sistema D capaz de realizar el producto en $GF(2^8)$.

En general, los sistemas son combinaciones entre si, siendo el **Sistema E** una combinación entre el B y el C y el **Sistema F** combinación del C y el D. Para tener una visión más completa del sistema con el que se está trabajando, se hace necesario conocer algunas características del mismo como son el área ocupada y la frecuencia máxima de trabajo. Dependiendo de las funcionalidades del sistema integrado, se necesitará ocupar un área de FPGA. Esta característica es importante puesto que suele ser proporcional al coste de fabricación. Se representa mediante el número de slices ocupados por el diseño. La frecuencia máxima de trabajo nos permitirá conocer la tasa de bits con que puede operar y nos ofrecerá un elemento de comparación entre sistemas

Tras la síntesis de cada sistema se obtienen las características de área empleada y frecuencia máxima de reloj, indicadas en la tabla 7.1.

| Sistema | Área | Frec. máxima |
|---|---|---|
| A | 1472 slices | 53.299MHz |
| B | 1760 slices | 52.609MHz |
| C | 1643 slices | 58.153MHz |
| D | 1556 slices | 55.463MHz |
| E | 1918 slices | 50.906MHz |
| F | 1727 slices | 51.932MHz |

**Tabla 7.1:** *Características de los sistemas*

## 7.4 Experimentos y resultados

Para realizar todas las pruebas, dentro de las posibilidades que ofrece Rijndael, se toma como tamaño de bloque fijo 128 bits y la clave podrá variar entre 128, 192

y 256 bits, al ser estas cifras las definidas por el estándar AES. Pero el algoritmo está preparado para poder varíar el tamaño de bloque, por lo que podría ser otra la elección.

El objetivo final es mejorar el rendimiento de los diferentes sistemas, por lo que se procede a su estudio empleando dos métodos diferentes. El primero mediante el módulo debug hardware, con el que se realizará el *profile* del sistema y nos indicará el rendimiento de cada una de las funciones del algoritmo. Por otro lado, se empleará el *timer* para contabilizar el número de ciclos de reloj que tarda cada fase del algoritmo en ejecutarse y así conocer la tasa de bits en función de la frecuencia de reloj.

## Prueba A: Debug hardware

Un aspecto importante en el diseño es tratar de obtener un rendimiento óptimo para el propósito perseguido, siendo uno de los puntos más importantes la localización de cuellos de botella para mejorarlos en medida de lo posible. A través de los tiempos de ejecución de las funciones, y de la jerarquía de llamadas entre éstas, se puede evaluar el rendimiento del sistema. Esta prueba se realiza mediante el uso de profiles. En concreto se han empleado dos: `histogram`, que indica el tiempo empleado en ejecutarse por cada función, el número de llamadas totales y el tiempo empleado por cada una de ellas, y `call_graph`, que nos permite observar el tiempo que necesita una función y sus decendientes.

En las estadísticas de un profile, el tiempo empleado por las funciones puede variar de ejecución en ejecución, ya que se basa en un sistema de muestras. Por otro lado, el sistema sufre un ralentizamiento debido al software intrusivo para poder realizar el debug. Esto nos lleva a que los resultados obtenidos no pueden ser tomados al pie de la letra, pero sí como comparación entre los diferentes sistemas. En las figuras 7.1 y 7.2 se muestra dicha comparación tomando como referencia el sistema A, observando el tiempo de ejecución de las cuatro funciones principales del algoritmo y el total empleado para generar las tablas, expandir la clave y cifrar y descifrar 1 MByte de datos.

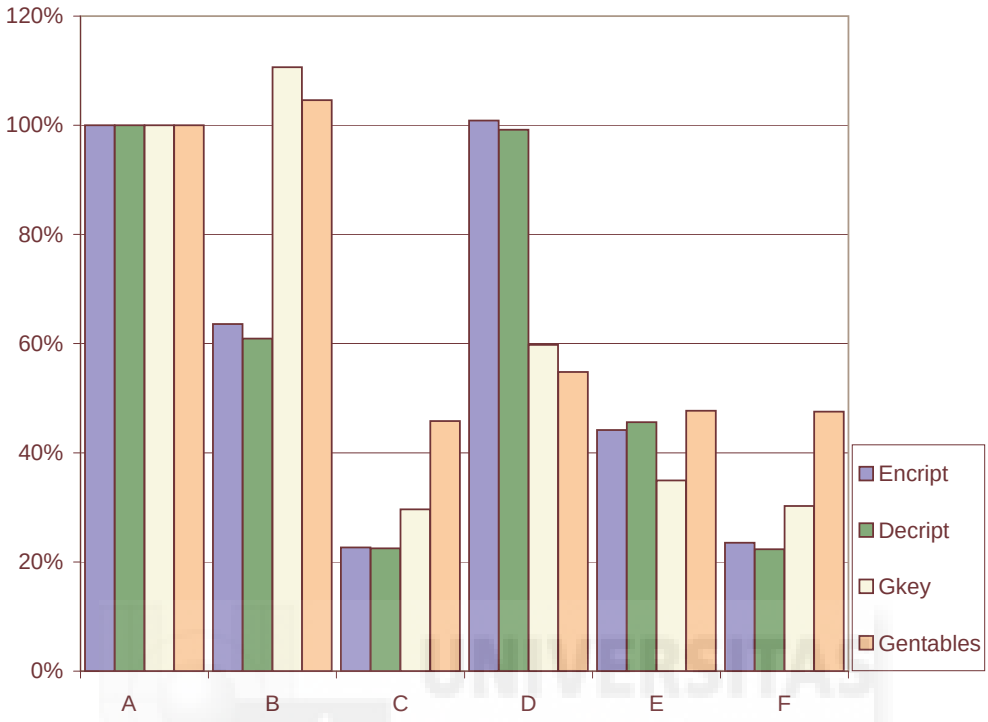


**Figura 7.1:** *Porcentaje de tiempo de ejecución en función del sistema A*

## Prueba B: Timer

Para conocer el número de ciclos de reloj que emplea una parte o el total del algoritmo se puede emplear el timer incluido con el sistema configurándolo para que durante un intervalo de tiempo controlado en software cuente estos ciclos de reloj. En función de la frecuencia máxima del sistema (tabla 7.1) los ciclos de reloj significarán un tiempo de ejecución diferente. El timer está conectado como esclavo al bus OPB y puede contener valores de hasta 32 bits. En este segundo estudio se pretende obtener el tiempo necesario para generar las tablas, expandir la clave y para cifrar y descifrar un texto de 128 bits y otro de 1 Mbyte. Este tiempo lo obtendremos a partir del número de ciclos de reloj proporcionado por el timer y la frecuencia máxima obtenida en el proceso de síntesis. El tiempo empleado para cifrar o descifrar lo emplearemos para calcular la tasa de bits del sistema, parámetro que podremos asociar a la velocidad con que se procesan los datos. Las pruebas se realizan para las longitudes de clave de 128, 192 y 256 bits, mientras que el tamaño del bloque se ha mantenido en 128 bits.

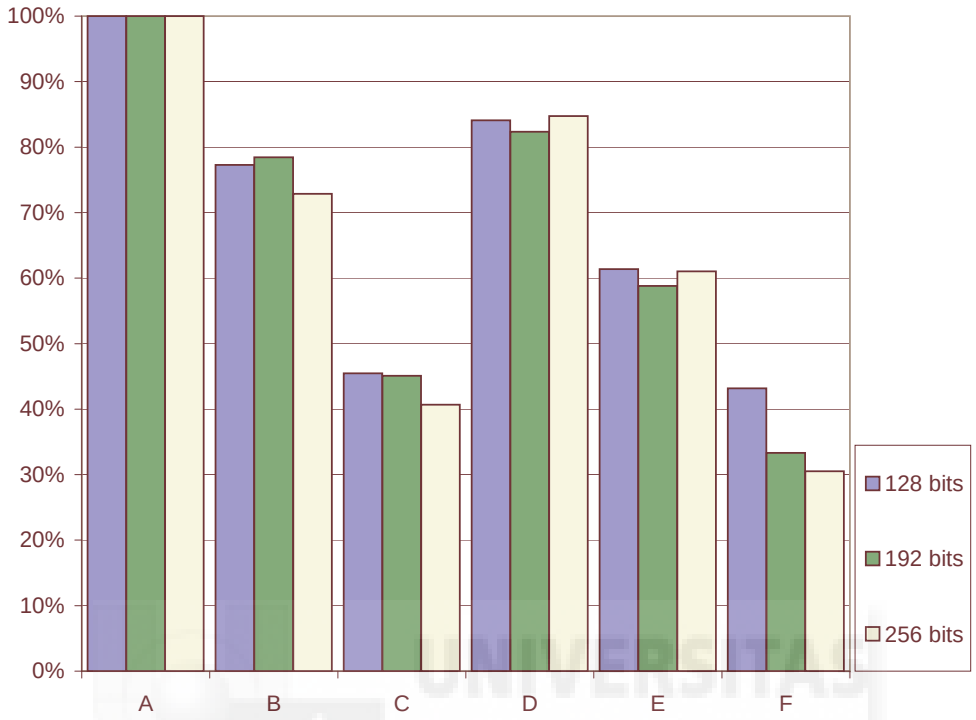


**Figura 7.2:** *Porcentaje de tiempo de ejecución total en función del sistema A*

En los experimentos se ha medido la velocidad del proceso de cifrado de forma independiente y también la de cifrado completo, incluyendo la expansión de clave y cifrado para un tamaño de 1 Mbyte y de 100 Mbytes. Las figuras 7.6 y 7.7 muestran los resultados obtenidos.

Para conocer el número de ciclos de reloj que emplea una parte o el total del algoritmo se puede emplear el timer incluído con el sistema. Configurándolo para que durante un intervalo de tiempo controlado en software cuente estos ciclos de reloj.

En función de la frecuencia máxima del sistema (tabla 7.1) los ciclos de reloj significarán un tiempo de ejecución diferente. Teniendo en cuenta que:

$$tiempo = \frac{ciclos}{f_{max}}$$

El tiempo empleado por la función de generación de tablas (gentables) y por la expansión de clave (getkey) se muestran en las figuras 7.3 y 7.4.

Si se divide el tamaño de los datos cifrados entre el tiempo empleado para ello se obtiene la tasa de bits, parámetro que mide la velocidad del sistema.

$$tasa = \frac{datos}{tiempo}$$

En las pruebas se ha medido por un lado la velocidad del proceso de cifrado de forma independiente y por otro la del algoritmo completo, incluyendo expansión de clave y cifrado, y variando el tamaño de los datos a cifrar con 1 MByte y 100 MBytes. Las figuras 7.5, 7.6 y 7.7 muestran los resultados.

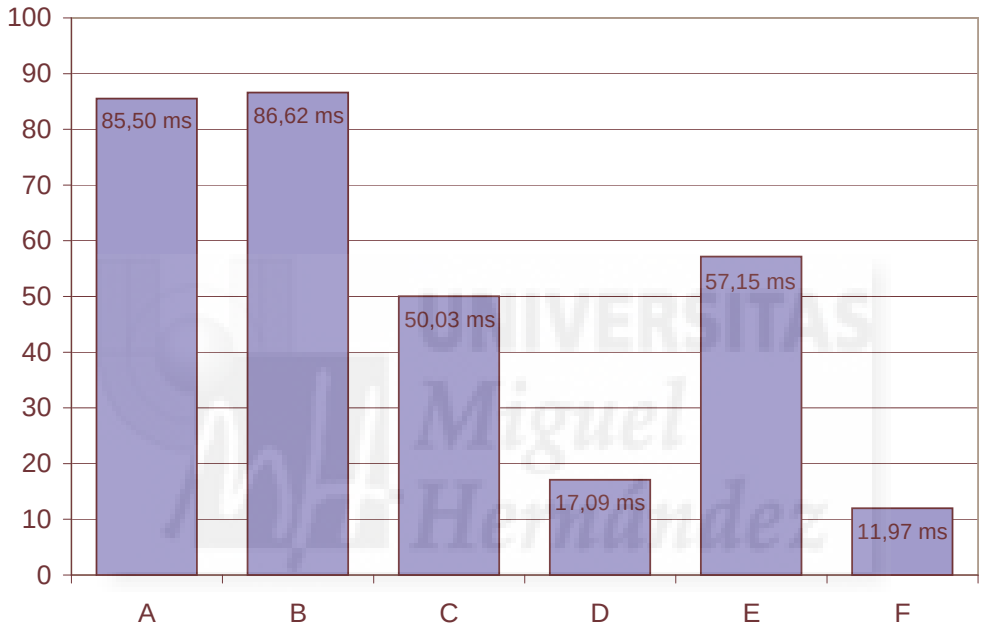


**Figura 7.3:** *Tiempo empleado para la generación de tablas (ms)*

# 7.5 Conclusiones

El objetivo final en el diseño de los sistemas para las pruebas, ha sido ahorrar tiempo en su diseño y a la vez ser capaz de sacar el máximo potencial al rendimiento. Todos están basados en pequeñas modificaciones que no aumentan el tiempo de desarrollo pero que pueden aumentar en un alto porcentaje la velocidad del algoritmo. Comparando siempre los sistemas a partir del sistema básico y separando las cuatro fases del algoritmo se diferencian las secciones que se comentan a continuación.

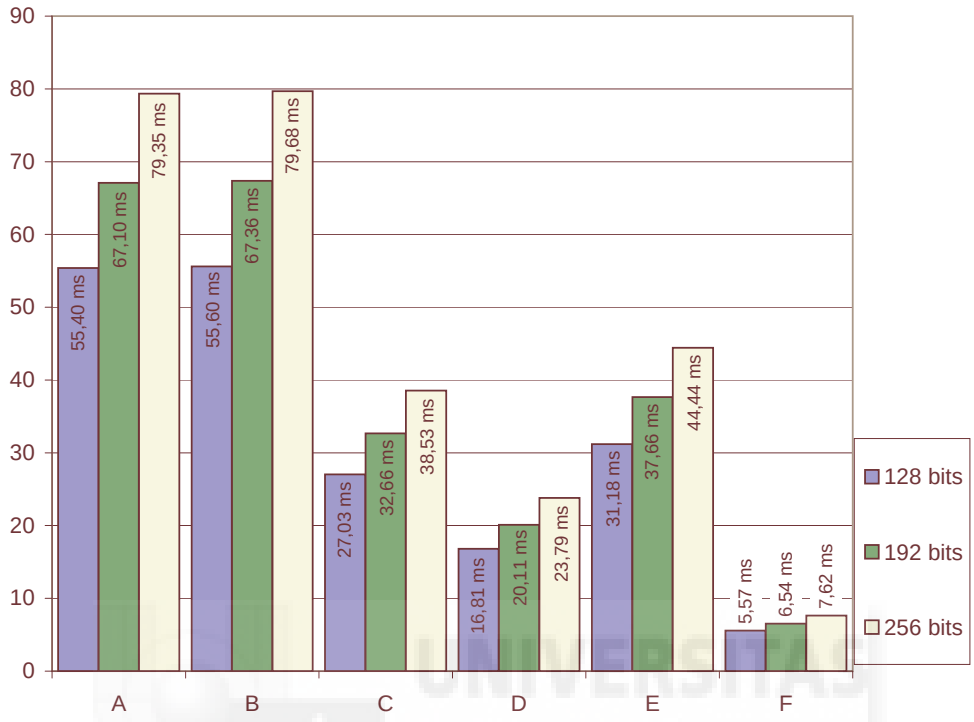


**Figura 7.4:** *Tiempo empleado para la expansión de clave (ms)*

## Generación de tablas

El hecho de que las tablas utilizadas por el algoritmo no estén definidas en el código fuente, sino que sea una función `gentables` la que se encarga de crearlas, producirán un ahorro notable en el tamaño en el código del ejecutable. Esto puede ser interesante si el programa no está permanentemente en memoria, sino que se carga siempre antes de la ejecución. Si existe la necesidad de generar las tablas, de la figura 7.3 se deduce que se consigue un aumento del rendimiento si se incluye en el diseño el Hardware Barrel Shifter (36 % de mejora con respecto a sistema A) y el coprocesador Bmul (casi 80 % de mejora). El sistema más rápido es el que incluye ambas opciones, el sistema F (con un 86 % de mejora). En otros casos no se produce optimización con respecto al software.

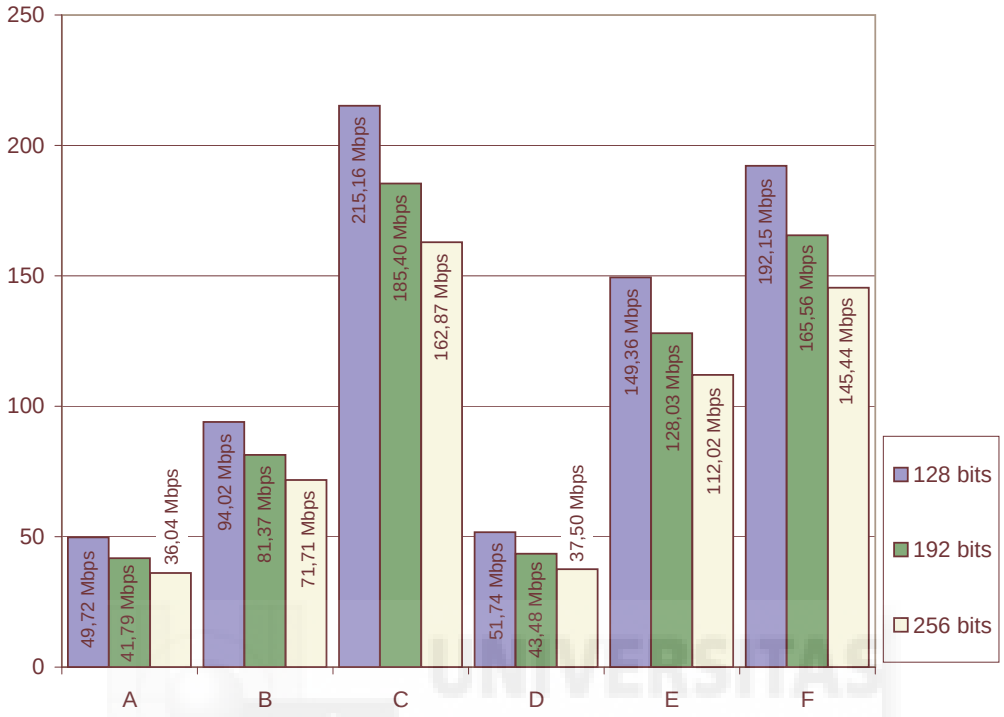


**Figura 7.5:** *Tasa de bits para el proceso de cifrado*

## Expansión de clave

Únicamente al principio del algoritmo es necesario generar las subclaves para el cifrado y el descifrado, ya que si la clave de usuario no cambia éstas tampoco lo harán.

De nuevo, los sistemas con mejor rendimiento son los que incluyen el Hardware Barrel Shifter (reducción del tiempo de ejecución de más del 50 %) y el coprocesador Bmul (reducción del 70 %) y su combinación es la que más optimiza el código software (reducción del 90 %). Por otro lado, los coprocesadores ROTLs aportan una mejora del sistema despreciable (ver figura 7.4).

## Proceso de cifrado

Para los procesos de cifrado y descifrado se obtiene unos tiempos de ejecución exactamente iguales, por lo que se tratan como el mismo proceso. Ésta es la fase más importante, ya que es la única que tiene que repetirse muchas veces durante

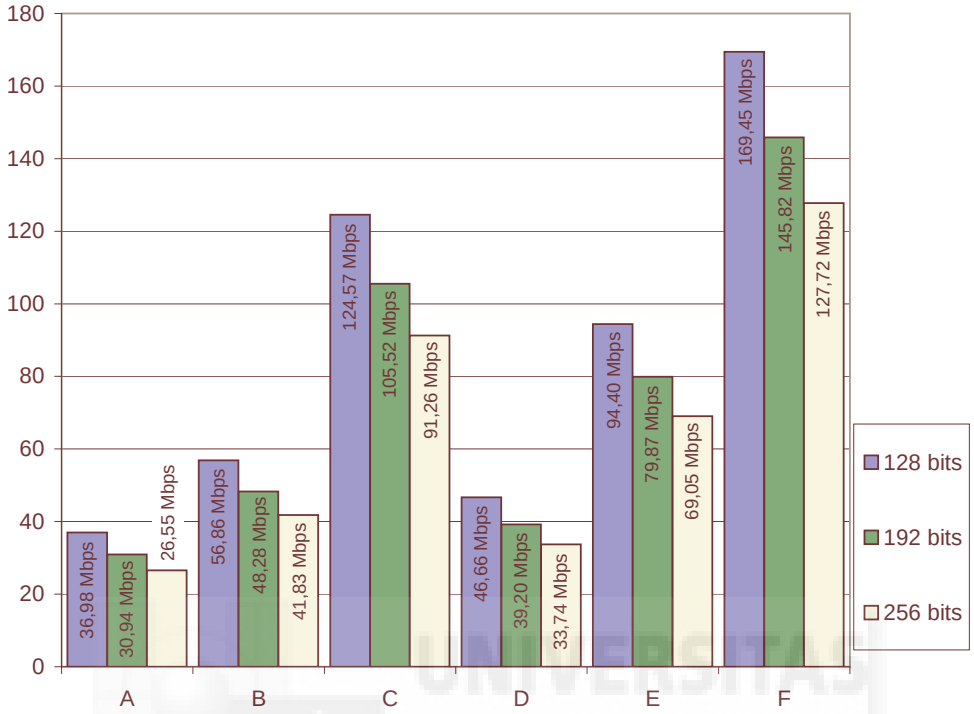


**Figura 7.6:** *Tasa de bits para el cifrado de 1 MByte*

la ejecución total del sistema, y dependerá del tamaño de los datos a cifrar. La principal mejora conseguida coincide con la inclusión de la rotación de bits hardware, conseguida en los sistemas que incluyen los coprocesadores ROTLs (aumento de la tasa de bits de casi el doble) y en los que incluyen el Hardware Barrel Shifter (por encima de cuatro veces más rápido). Pero al contrario que en los casos anteriores, la combinación de ambas no produce un incremento de la mejora, sino que suaviza la opción óptima, la del Barrel Shifter (con un aumento del triple de la velocidad si se combinan). El coprocesador Bmul no aporta mejora al cifrado y al descifrado, ya que la optimización software de sustitución en tablas T tiene como fin que no sean necesarias este tipo de operaciones (ver figura 7.5).

## Conclusiones finales

Si las características del sistema donde se quiera implementar Rijndael exigen cifrar bloques de tamaño pequeño y variar la clave para cada uno de estos bloques, se obtendrán mejores resultados si se incluye el coprocesador Bmul junto con el

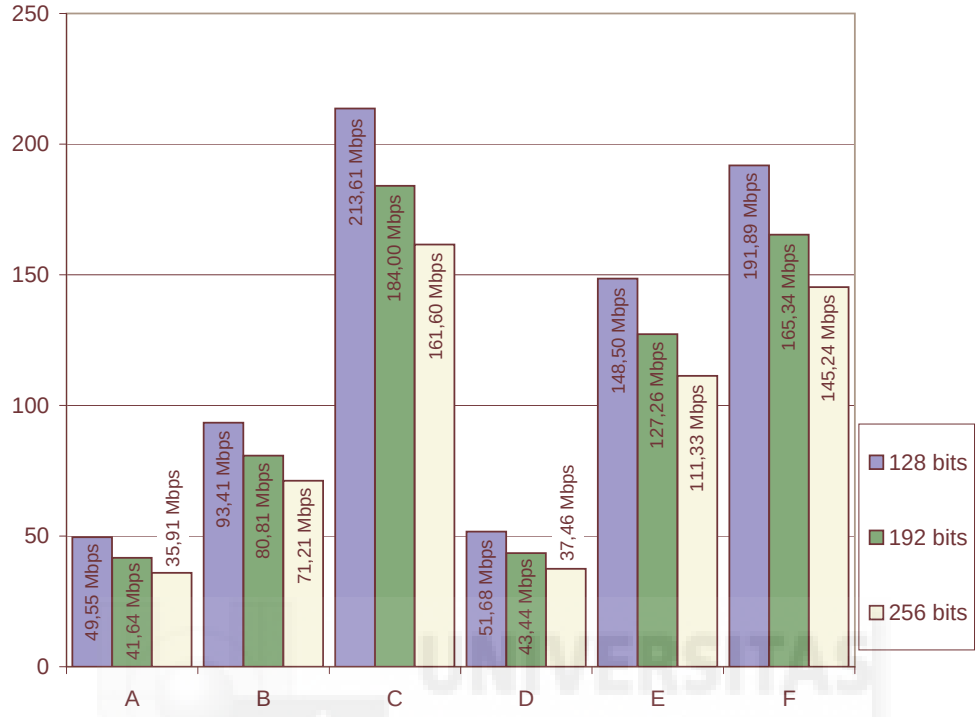


**Figura 7.7:** *Tasa de bits para el cifrado de 100 MBytes*

Hardware Barrel Shifter como en el caso del sistema F (ver figura 7.6), ya que se disminuye la velocidad del proceso de cifrado pero se compensa reduciendo el tiempo de expansión de clave. Si el caso que se necesita es el del cifrado de grandes bloques de datos, en los que la expansión de clave significa un porcentaje pequeño del tiempo de ejecución total, la opción recomendable es la del sistema C, donde el aumento de la tasa de bits en el proceso de cifrado hace que la velocidad total aumente (ver figura 7.7). A la vista de los resultados obtenidos, la opción que implementa el sistema E ilustra claramente cómo al duplicar una tarea y realizarla conjuntamente mediante hardware y software, la mejora introducida por el coprocesador hardware se reduce considerablemente ya que la resolución de la tarea desviada hacia el software se produce en un tiempo mucho mayor.

# Capítulo 8

# Conclusiones y futuras investigaciones

El objetivo principal de la tesis se ha conseguido construyendo un nuevo algoritmo escalable de multiplicación en cuerpos finitos (del tipo $GF(2^m)$). El algoritmo escalable propuesto ha sido diseñado en hardware reconfigurable y se basa en el algoritmo de reducción modular por palabras propuesto en [44] el cual resulta en una arquitectura eficiente y que no necesita de ninguna conversión de datos ya que opera en base polinomial.

Para este algoritmo se presenta un diseño mixto hardware-software de un multiplicador en cuerpos finitos. Este algoritmo soluciona el problema de la escalabilidad y por tanto, se puede utilizar con valores de hasta cierto grado $m$ como máximo y para cualquier polinomio irreducible $f(x)$. Además, es posible utilizarlo para cualquier polinomio irreducible en forma de trinomio o pentanomio sin el re-diseño del algoritmo en hardware. Se han realizado distintas pruebas del algoritmo sobre una plataforma CSoC, proporcionando unos valores de rendimiento similares a otros algoritmos de la literatura aunque el objetivo final, no era obtener un mejor rendimiento, sino solucionar el problema de la escalabilidad.

También se presenta un diseño mixto hardware-software del algoritmo estándar de la criptografía simétrica, Rijndael. A tenor de los resultados obtenidos, se puede afirmar que mediante el análisis del algoritmo es posible optimizar la ejecución de un sistema criptográfico como es Rijndael en una plataforma CSoC.

## 8.1 Recomendaciones para futuras investigaciones

- La dependencia de los datos en el algoritmo escalable del capítulo 6 puede permitir realizar operaciones en paralelo.
- Es posible utilizar el algoritmo escalable en el diseño de un procesador criptográfico de curvas elípticas
- Sobre la implementación de Rijndael, la optimización del software (optimización con tablas T) únicamente es válida para procesadores de 32 bits, teniendo que usar el código fuente tradicional (que usa las funciones ByteSub, MixColumns, AddRoundKey y ShiftRows de Rijndael en cada ronda) en procesadores menores como los de 8 bits. Para estos casos habría que estudiar si estas optimizaciones son positivas. También se pueden construir otro tipo de coprocesadores para implementar los sistemas de forma más veloz.

## 8.2 Contribuciones de este trabajo

Los resultados obtenidos durante esta investigación han sido presentados en conferencias internacionales y en revistas especializadas. Los más destacados son:

(a) Analysis and Implementation Hardware-Software of Rijndael Encryption, [34]
(b) A scalable finite field multiplier with interleaving reduction, [19]
(c) A Ghost Bit Based Finite Field Arithmetic for FPGAs ,[18]
(d) IP core_ethernet for firewall architectures, [36]
(e) A Note About Binary Finite Fields Multiplication on FPGA,[35]

# Bibliografía


[1] Virtex 2.5v field programmable gate arrays xilinx data sheet,. http://direct.xilinx.com/bvdocs/publications/ds003.pdf.

[2] Virtex ii xilinx guide,. http://www.xilinx.com.

[3] Xilinx, inc. http://www.xilinx.com.

[4] X. A. N. 529. Connecting customized ip to the microblaze soft processor using the fast simplex link (fsl). Informe técnico, Xilinx, 2003.

[5] G. B. Agnew, T. Beth, R. C. Mullin y S. A. Vanstone. Arithmetic operations in $GF2^m$). *Journal of Cryptology*, **6** (1993).

[6] G. Agnew, R. C. Mullin y S. A. Vanstone. An implementation of elliptic curve cryptosystems over $gf(2^155)$. *IEEEJSAC: IEEE Journal on Selected Areas in Communications*, **11** (1993).

[7] T. Baigneres, P. Junod y S. Vaudenay. How far can we go beyond linear cryptanalysis. En *Advances in Cryptology - Asiacrypt04, volume 3329 of LNCS*, páginas 432–450. Springer-Verlag, 2004.

[8] L. Batina, S. B. Örs, B. Preneel y J. Vandewalle. Hardware architectures for public key cryptography. *Integr. VLSI J.*, **34(1-2)**: 1–64, mayo 2003.

[9] J. Becker. Configurable systems-on-chip (csoc). En *Proceedings of the In Integrated Circuits and Systems Design*, FPGA '04, páginas 379–384. 2002.

[10] J. Becker. Configurable systems-on-chip (csoc). *Integrated Circuit Design and System Design, Symposium on*, **0**: 379 (2002).

[11] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi y J. Teich. Reconfigurable implementation of elliptic curve crypto algorithms. IEEE, apr 2002.

[12] D. J. Bernstein. Circuits for integer factorization: A proposal, 2001.

[13] T. Beth, B. M. Cook y D. Gollmann. Architectures for exponentiation in gf (2n). En *Proceedings on Advances in cryptology—CRYPTO '86*, páginas 302–310. Springer-Verlag, London, UK, 1987.

[14] I. BLAKE, G. SEROUSSI y N. SMART. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Notes Series 265. Cambridge University Press, 1999.

[15] K. Bondalapati y V. K. Prasanna. Reconfigurable computing systems, 2002.

[16] W. E. Burr. Selecting the advanced encryption standard. *IEEE Security and Privacy*, **1(2)**: 43–52, marzo 2003.

[17] S. Casselman. Virtual computing and the Virtual Computer. En *FPGA '93: Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, páginas 43–48. IEEE Computer Society, Reseda, CA, USA, abril 1993.

[18] J. Climent, F. Crespi y A. Grediaga. A ghost bit based finite field arithmetic for fpgas. En *Proceedings International e-Conference on Computer Science*, volumen 8 de *Lecture Series on Computer and Computational Sciences*, páginas 44–49. 2006.

[19] J. Climent, F. Crespi y A. Grediaga. A scalable finite field multiplier with interleaving reduction. En *Proceedings International e-Conference on Computer Science*, volumen 8 de *Lecture Series on Computer and Computational Sciences*, páginas 50–53. 2006.

[20] D. Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM J. Res. Dev.*, **38(3)**: 243–250, mayo 1994.

[21] A. Corporation. *Excalibur Devices Hardware Reference Manual, Version 3.1*.

[22] A. Corporation. *Nios Embedded Processor System Development.Disponible Online: http://www.altera.com/products/ip/processors/nios/nio-index.html*.

[23] J. Daemen y V. Rijmen. Rijndael. En *Encyclopedia of Cryptography and Security (2nd Ed.)*, páginas 1046–1049. 2011.

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

[25] W. Diffie y M. E. Hellman. *New Directions in Cryptography*, 1976.

[26] A. Elbirt, W. Yip, B. Chetwynd y C. Paar. An fpga-based performance

evaluation of the aes block cipher candidate algorithm finalists. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, **9**: 545–557 (2001).

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

[28] D. S. A. Elminaam, H. M. Abdual-Kader y M. M. Hadhoud. Evaluating the performance of symmetric encryption algorithms. *I. J. Network Security*, **10(3)**: 216–222 (2010).

[29] T. D. et al. *The TLS Protocol, Version 1.0*. IETF RFC2246, 1999.

[30] FIPS. *Advanced Encryption Standard (AES)*, noviembre 2001.

[31] A. O. Freier, P. Karlton y P. C. Kocher. The ssl protocol — version 3.0. Internet Draft, Transport Layer Security Working Group, November 1996.

[32] G. Gaubatz. *Versatile montgomery multiplier architectures*. Tesis Doctoral, Worcester Polytechnic Institute, 2002.

[33] Goodman y Chandrakasan. An energy efficient reconfigurable public-key cryptography processor architecture. 2000.

[34] A. Grediaga, F. Brotons, B. Ledesma y F. Crespi. Analysis and implementation hardware-software of rijndael encryption. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, **8(1)**: 82 –87, march 2010.

[35] A. Grediaga, F. Crespi y R. Gutierrez. A note about binary finite fields multiplication on fpga. *WSEAS TRANSACTIONS ON CIRCUITS AND SYSTEMS*, **3**: 1924–1928 (2004).

[36] A. Grediaga, F. Ibarra, F. G. Crespi y F. Brotons. Ip core_ethernet v1.02a for firewall architectures. *Lecture Series on Computer and Computational Sciences*, **8**: 50–53 (2007).

[37] Z. Guo, W. Najjar, F. Vahid y K. Vissers. A quantitative analysis of the speedup factors of fpgas over processors. En *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, FPGA '04, páginas 162–170. ACM, New York, NY, USA, 2004.

[38] N. Gura, H. Eberle y S. C. Shantz. Generic implementations of elliptic curve cryptography using partial reduction. En V. Atlury (editor), *Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS-02)*, páginas 108–116. New York, ACM Press 2002.

[39] Y. Han, P.-C. Leong, P.-C. Tan y J. Zhang. Fast algorithms for elliptic curve cryptosystems over binary finite field. En K.-Y. Lam, E. Okamoto y C. Xing (editores), *ASIACRYPT*, volumen 1716 de *Lecture Notes in Computer Science*, páginas 75–85. Springer, 1999.

[40] D. Hankerson, J. L. Hernandez y A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *Lecture Notes in Computer Science*, **1965** (2001).

[41] D. Hankerson, J. L. Hernandez y A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *Lecture Notes in Computer Science*, **1965** (2001).

[42] I. S. Hsu, T. K. Truong, H. M. Shao, L. J. Deutsch y I. S. Reed. A Comparison of VLSI Architecture of Finite Field Multipliers Using DualNormal or Standard Basis. *Telecommunications and Data Acquisition Progress Report*, **90**: 63–75, abril 1987.

[43] M. Hütter, J. Grossschädl y G.-A. Kamendje. A versatile and scalable digit-serial/parallel multiplier architecture for finite fields gf($2^m$). En P. K. Srimani, W. W. Bein, R. R. Hashemi, E. Lawrence, M. Cannataro, E. E. Regentova y A. H. Spink (editores), *Proceedings of the 4th International Conference on Information Technology: Coding and Computing (ITCC 2003)*, páginas 692–700. IEEE Computer Society Press, 2003.

[44] M. Hütter, J. Grossschädl y G.-A. Kamendje. A versatile and scalable digit-serial/parallel multiplier architecture for finite fields gf($2^m$). En P. K. Srimani, W. W. Bein, R. R. Hashemi, E. Lawrence, M. Cannataro, E. E. Regentova y A. H. Spink (editores), *Proceedings of the 4th International Conference on Information Technology: Coding and Computing (ITCC 2003)*, páginas 692–700. IEEE Computer Society Press, 2003.

[45] IEEE. Standard specifications for public-key cryptography. En *The IEEE P1363 Home Page*. IEEE, http://grouper.ieee.org/groups/1363/.

[46] D. Johnson y A. Menezes. The elliptic curve digital signature algorithm (ecdsa). Informe técnico, 1999.

[47] A. Jurisic y A. J. Menezes. Guide to elliptic curve cryptography. **19**: 173–193 (2004).

[48] A. A. Karatsuba y Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, **7**: 595–596 (1963). URL: `http://cr.yp.to/`

bib/entries.html#1963/karatsuba.

[49] A. A. Karatsuba y Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, **7**: 595–596 (1963). URL: http://cr.yp.to/bib/entries.html#1963/karatsuba.

[50] S. Kent y R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.

[51] A. Klein. Attacks on the rc4 stream cipher. *Des. Codes Cryptography*, **48**: 269–286, September 2008.

[52] K. P. L. Song. Efficient finite field serial/parallel multiplication. International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '96), August 19 - 23, 1996 1996.

[53] X. Lai y J. Massey. A proposal for a new block encryption standard. En *Advances in Cryptology — Eurocrypt '90*, páginas 389–404. Springer-Verlag, Berlin, 1991.

[54] B. A. Laws y C. K. Rushforth. A cellular-array multiplier for gf(2m). *IEEE Trans. Comput.*, **20(12)**: 1573–1578 (1971).

[55] J.-H. K. D.-H. Lee. A compact finite field processor over gf(2m) for elliptic curve cryptography. *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, **2**: II–340–II–343 vol.2 (2002).

[56] R. Lidl y H. Niederreiter. *Introduction to Finite Fields and Their Application*. Cambridge University Press, Cambridge, MA, USA, 1994.

[57] J. López y R. Dahab. High-speed software multiplication in $GF(2^m)$. *Lecture Notes in Computer Science*, **1977** (2000).

[58] J. López y R. Dahab. High-speed software multiplication in $GF(2^m)$. *Lecture Notes in Computer Science*, **1977** (2000).

[59] P. Lysaght y J. Dunlop. Dynamic reconfiguration of fpgas. En *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs*, páginas 82–94. Abingdon EE&CS Books, Oxford, UK, UK, 1994.

[60] R. L. Lysecky y F. Vahid. A study of the speedups and competitiveness of fpga soft processor cores using dynamic hardware/software partitioning. En *DATE*, páginas 18–23. IEEE Computer Society, 2005.

[61] D. R. Martinez, T. J. Moeller y K. Teitelbaum. Application of recon-

figurable computing to a high performance front-end radar signal processor. *J. VLSI Signal Process. Syst.*, **28**: 63–83, May 2001.

[62] E. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. Tesis Doctoral, Linkoping Univ, Linkoping Sweden, 1991.

[63] R. J. McEliece. *Finite field for scientists and engineers*. Kluwer Academic Publishers, 1987.

[64] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone y R. L. Rivest. Handbook of applied cryptography, 1997.

[65] V. S. Miller. Use of elliptic curves in cryptography. En H. C. Williams (editor), *85*, volumen 218 de *417-426*. 1985.

[66] V. S. Miller. Use of elliptic curves in cryptography. En *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*, páginas 417–426. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[67] T. J. Moeller y D. R. Martinez. Field programmable gate array based radar front-end digital signal processing. En *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '99, páginas 178–. IEEE Computer Society, Washington, DC, USA, 1999.

[68] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone y R. M. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Appl. Math*, **22(2)**: 149–161. 10, Feb 1988/89.

[69] National Bureau of Standards. *FIPS Publication 46-1: Data Encryption Standard*, January 1988.

[70] National Institute of Standards and Technology (NIST). The Digital Signature Standard, proposal and discussion. *Communications of the ACM*, **35(7)**: 36–54, July 1992.

[71] C. Negre. Parallel Multiplication in GF($2^n$) using Condensed Matrix Representation. En *SECRYPT'06, Setúbal,Portugal*, páginas 254–259. aug 2006.

[72] X. A. Note. Coolrunner-II CPLD Galois field $GF(2^m)$ multiplier. Informe técnico, Xilinx, 2003.

[73] S. Okada, N. Torii, K. Itoh y M. Takenaka. Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA. *Lecture Notes in Computer Science*, **1965** (2001).

[74] S. Okada, N. Torii, K. Itoh y M. Takenaka. Implementation of elliptic

curve cryptographic coprocessor over $GF(2^m)$ on an FPGA. *Lecture Notes in Computer Science*, **1965** (2001).

[75] J. K. Omura y J. Massey. Computational method and apparatus for finite field arithmetic. U.S. Patent No. 4,587,627, 1986.

[76] G. Orlando y C. Paar. A super-serial galois field multiplier for fpgas and its applications to public-key algorithms. En *7th IEEE Symposium on Field Programmable Custom Machines (FCCM '99)*, páginas 232–239. IEEE Computer Society Press, Apr 1999.

[77] C. Paar. *Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields.* Tesis Doctoral, *Universität* GH Essen, Essen, Germany, June 1994.

[78] G. Research. Leon2 processor user's manual. http://www.gaisler.com.

[79] R. L. Rivest, A. Shamir y L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, **21(2)**: 120–126, febrero 1978.

[80] M. Rosing. *Implementing elliptic curve cryptography.* Manning Publications Co., Greenwich, CT, USA, 1999.

[81] RSA Data Security, Inc. *PKCS #5: Password-Based Encryption Standard*, June 1991. Version 1.4.

[82] A. Satoh y K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Trans. Computers*, **52(4)**: 449–460 (2003).

[83] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* 1994.

[84] G. Seroussi. Table of low-weight binary irreducible polynomials. Informe técnico, Visual Computing Dept., Hewlett Packard Laboratories, 1998.

[85] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal, Vol 28, pp. 656-715*, , Oktober 1949.

[86] I. E. Shparlinski. *Finite fields: Theory and computation.* Kluwer Academic Publishers, 1999.

[87] I. E. Shparlinski. *Finite fields: theory and computation: the meeting point of number theory, computer science, coding theory, and cryptography*, volumen 477 de *Mathematics and its applications.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[88] J. H. Silverman. Fast multiplication in finite fields gf(2n). En *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware*

*and Embedded Systems*, páginas 122–134. Springer-Verlag, London, UK, 1999.

[89] L. Song y K. K. Parhi. Low-energy digit-serial/parallel finite field multipliers. *J. VLSI Signal Process. Syst.*, **19(2)**: 149–166 (1998).

[90] W. Stallings. *Cryptography and Network Security*. NJ: Prentice Hall, 2003.

[91] L. Surhone, M. Tennoe y S. Henssonow. *Openrisc 1200*. Betascript Publishing, 2011.

[92] R. R. Taylor y S. C. Goldstein. A high-performance flexible architecture for cryptography. En *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems 1999 (CHES99)*. Worcester, MA, 1999.

[93] S. Tazawa. Integer factorization and discrete logarithm problem are neither in p nor np-complete. *CoRR*, **abs/1207.2171** (2012).

[94] J. Turley. Soft computing reconfigures designer options. En *Embedded Systems*. April 1997.

[95] H. C. A. van Tilborg y S. Jajodia (editores). *Encyclopedia of Cryptography and Security, 2nd Ed.* Springer, 2011.

[96] Wang, Truong, Shao, Deutsch, Omura y Reed. VLSI architectures for computing multiplications and inverses in $GF(2^m)$. *IEEETC: IEEE Transactions on Computers*, **34** (1985).

[97] A. Weimerskirch y C. Paar. Generalizations of the karatsuba algorithm for efficient implementations. Cryptology ePrint Archive, Report 2006/224, 2006.

[98] E. D. Win, A. Bosselaers, S. Vandenberghe, P. D. Gersem y J. Vandewalle. A fast software implementation for arithmetic operations in $gf(2^n)$. En *Advances in Cryptology – AISACRYPT '96*, páginas 65–76. 1996.

[99] T. Wollinger, J. Guajardo y C. Paar. Security on fpgas: State-of-the-art implementations and attacks. *ACM Trans. Embed. Comput. Syst.*, **3(3)**: 534–574, agosto 2004.

[100] Xilinx. *MicroBlaze processor reference Guide*, April 2010. UG081 (v11.0).

[101] C.-S. Yeh, I. Reed y T. Truong. Systolic multipliers for finite fields gf(2m). *Transactions on Computers*, **C-33(4)**: 357–360 (April 1984).