CENTRO DE INVESTIGACIÓN Y DE
ESTUDIOS AVANZADOS DEL INSTITUTO
POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

“Análisis del algoritmo para cifrado de imágenes mediante curvas elípticas, implementado en un dispositivo programable FPGA”

TESIS

Que presenta

M. EN C. JOSÉ DE JESÚS MORALES ROMERO

Para obtener el grado de

Doctor en Ciencias

EN LA ESPECIALIDAD DE

INGENIERÍA ELÉCTRICA

Directores de la Tesis:

Dr. Mario Alfredo Reyes Barranca

Dr. David Tinoco Varela

Ciudad de México Diciembre, 2022

Agradecimientos

A CONACYT por el apoyo económico y administrativo que me otorgó para realizar mi programa de doctorado.

A mi familia y en especial a mi madre por todo el apoyo incondicional que me ha brindado durante mis estudios.

A mis asesores Mario Alfredo Reyes Barranca y David Tinoco Varela por toda su ayuda y apoyo para realizar esta tesis, además de M.C. Luis Martín Flores Nava y al M. en C. Emilio Rafael Espinosa García por su colaboración y consejos, a Yesenia Cervantes Aguirre por su gran apoyo, a mis maestros y personal de apoyo del CINVESTAV.

A mis amigos y compañeros que me han apoyado y dado consejos durante mi programa de doctorado.

Dedicatoria

A mi familia, por todo el apoyo que me han otorgado y principalmente a mi madre por su cariño y apoyo incondicional.

Índice general

Índice de figuras

Preface

In this work, a study is conducted on the encryption and protection of data, specifically, those delivered by the processing of a cellular neural network on images.

Chapter 1 introduces cryptography and artificial neural networks. The relationship of neural networks and their use in cryptography is addressed. In addition, we will discuss the so-called side-channel attacks on programmable devices, as well as the protection of such devices from this type of attack. In Chapter 2 we will talk about the importance of neural networks and their relationship with image processing. It will deal specifically with cellular neural networks, which are useful when processing images, as well as their implementation in FPGAs. Also, we will deal with the topic of searching for templates and propose a method for searching for these.

Chapter 3 will discuss cryptography and its implementation in FPGAs. As well as that of side-channel attacks, especially on RSA, and the protection of FPGAs devices that prevent such attacks. Chapter 4 will show the encryption of data using elliptic curves, as well as its corresponding implementation in FPGAs. The implementation of operations on finite fields is displayed.

In chapter 5 data encryption will be conducted using the implementations seen in the previous chapters, in addition a methodology will be proposed that reduces the amount of data to be processed using cellular neural networks. Examples of image encryption will be shown in the Results section. In the Conclusions section, a discussion of the most important points observed in the results of the work is presented, in addition to showing a discussion about these. Finally, in the Future Work section, topics will be proposed to improve this work.

Resumen

En este trabajo, se realiza un estudio sobre el cifrado y protección de datos, específicamente, los entregados por el procesamiento de una red neuronal celular sobre imágenes.

En el capítulo 1 se da una introducción a la criptografía y a las redes neuronales artificiales. Se aborda la relación de redes neuronales y su uso en la criptografía. Además, trataremos sobre los llamados ataques de canal lateral sobre los dispositivos programables, así como la protección de dichos dispositivos a este tipo de ataques. En el capítulo 2 hablaremos sobre la importancia de las redes neuronales y su relación con el procesamiento de imágenes. Se tratará específicamente de las redes neuronales celulares, las cuales son de utilidad cuando se procesan imágenes, así como su implementación en los FPGAs. También, trataremos el tema de búsqueda de plantillas y se propondrá un método para la búsqueda de estas.

En el capítulo 3 tratará el tema de la criptografía y su implementación en los FPGAs, así como el de los ataques de canal lateral, especialmente sobre RSA, y la protección de los dispositivos FPGAs que evitan dichos ataques. En el capítulo 4 se mostrará el cifrado de datos utilizando curvas elípticas, además de su correspondiente implementación en FPGAs. Se mostrará la implementación de operaciones en campos finitos.

En el capítulo 5 se realizará el cifrado de datos utilizando las implementaciones vistas en los capítulos anteriores, además se propondrá una metodología que reduce la cantidad de datos a procesar utilizando las redes neuronales celulares. En la sección de Resultados se mostrarán algunos ejemplos de cifrado de imágenes. En la sección de

Conclusiones, se presenta una discusión de los puntos más importantes observados en los resultados del trabajo, además de mostrar una discusión sobre estos. Finalmente, en la sección de Trabajo a futuro se propondrán temas que mejoren este trabajo.

Introducción

Actualmente los dispositivos inteligentes transfieren, manipulan, utilizan y/o transportan información valiosa del usuario, como por ejemplo el número de sus cuentas bancarias, datos personales como nombre y domicilio, u otra información sensible para el usuario.

Este tipo de dispositivos envían los datos sensibles del usuario por medios no seguros, es decir, medios por los cuales cualquier persona observa los datos transmitidos. Para que los datos lleguen de manera segura a su destinatario se cifran; de esta manera si alguna persona ajena o algún atacante observa dicha información, no será entendida por esta persona o atacante. A este proceso se le denomina cifrado de la información, el cual requiere generalmente de una clave o contraseña. Dicho valor generalmente es únicamente conocido por el ente que recibe los datos o por las dos partes que se están comunicando. Sin embargo, Kocher observó que realizando mediciones se puede deducir u observar el valor de las claves secretas. A estas observaciones se les denominó Side Channel Attacks (SCAs).

Los datos que son transmitidos por los dispositivos pueden ser desde simples archivos de texto, texto en formato enriquecido, y hasta imágenes. Una parte importante es la imagen, ya que puede contener información valiosa. Ejemplo de imágenes son los códigos QR, imágenes faciales, huellas dactilares, entre otros. Es importante, realizar un estudio sobre el cifrado de dichas imágenes en los dispositivos, especialmente, en los programables como los FPGAs.

Debido al descubrimiento de Kocher sobre la obtención de las claves secretas, es decir los SCAs, es necesario realizar propuestas que mejoren la seguridad del cifrado

de imágenes, específicamente, cuando se realizan sobre los FPGAs.

En este trabajo realizaremos un estudio sobre diversas implementaciones de algoritmos para el cifrado de datos, especialmente, de imágenes. Además, se propondrán mejoras en las implementaciones que sean probablemente seguras.

Objetivos

Objetivo general

Diseñar e implementar un sistema criptográfico basado en un dispositivo programable FPGA, el cual será capaz de evitar invasiones indeseables e ilegales (por ejemplo, ataques del tipo Side Channel Attacks – SCA), además de que los datos a cifrar serán los procesados por una Red Neuronal Celular (CeNN) multiplexada en tiempo. Dicha CeNN procesará imágenes realizando las tareas, entre otras, de búsqueda de contornos. Se realizarán ataques sobre las implementaciones propuestas para realizar mejoras. Finalmente, se pretende proponer y optimizar una nueva metodología utilizando lo desarrollado por el cifrado de imágenes utilizando la CeNN.

Objetivo particular

Optimizar un diseño previo de Red Neuronal Celular, para tener la capacidad de procesar imágenes de mayor tamaño, además, de mejorar el tiempo de respuesta. También, aprovechar la teoría, y estrategias para la implementación de cifrado de datos, como lo son RSA y Curvas Elípticas, así como, el desarrollo de metodologías alternativas para la implementación de algoritmos en dispositivos programables como lo es el FPGA. Se pretende realizar ataques físicos sobre las implementaciones propuestas para su evaluación y mejora, así como realizar las propuestas que ayuden a mejorar la seguridad de estos algoritmos cuando sean implementados en FPGA.

Capítulo 1

Introducción a la criptografía y redes neuronales artificiales

1.1. Introducción a la criptografía

Los *sistemas criptográficos* son utilizados para proteger la información relevante de una entidad ya sea gubernamental, empresarial y en general de cualquier individuo. En particular, esta información podría ser las cuentas bancarias, información médica, datos personales o empresariales, etc. Con el propósito de que esta información no sea fácilmente entendible por un ente ajeno o personas no autorizadas, se cifra (*cifrar* se entiende como poner en clave o cifra, siendo lo contrario descifrar) utilizando diversos sistemas criptográficos. Dichos sistemas basan su cifrado de datos en algoritmos, que a su vez utilizan fórmulas matemáticas.

Al respecto, se puede decir que existen diversos algoritmos para el cifrado y descifrado de la información. Generalmente, éstos se clasifican en dos tipos: los de *clave pública* y los de *clave privada* [2, 3].

La criptografía de clave privada (también llamada *criptografía de clave simétrica*) utiliza una única clave, tanto para el cifrado como para el descifrado de la información. Éste tiene la desventaja de que cuando se envía información cifrada con este método, la

persona que requiera descifrar la información necesitará la clave con la que fue cifrada dicha información. Lo anterior provoca que se deba enviar previamente la clave por medios no seguros, lo cual es un punto de debilidad debido a que un ente ajeno pueda obtener dicha clave.

Dentro de este tipo de criptografía se encuentran los algoritmos *DES* (Data Encryption Standard), *AES* (Advance Encryption Standard) y el *Triple DES* (que en realidad es hacer cifrado con DES tres veces con tres diferentes claves).

Este tipo de criptografía requiere longitudes de claves que van desde los 128 bits hasta los 256 bits, las cuales son claves cortas comparadas con la criptografía de clave pública.

Por otro lado, la criptografía de clave pública (también llamada *criptografía de clave asimétrica*) utiliza dos claves distintas. Por un lado, se tiene una clave para el cifrado de la información, la cual puede ser conocida por aquella persona a la que se le quiera enviar información protegida, y en general, esta clave puede ser conocida por cualquier persona, de aquí el nombre de clave pública. Y, por otro lado, la otra clave, la cual es privada, es aquella que será solo conocida por la persona que va a descifrar la información. Esto hace que no sea necesario que las dos personas o entes que se van a comunicar conozcan la misma clave secreta.

Dentro de esta criptografía encontramos los algoritmos *ElGamal*, *RSA* (algoritmo propuesto por **R**ivest, **S**hamir y **A**dleman, de ahí su identificación como RSA), y *Criptografía con Curvas Elípticas* (ECC de Elliptic Curve Cryptography), entre otras.

Este tipo de criptografías requieren de longitudes de clave más grandes en comparación con la criptografía de clave privada, estas claves van desde los 1024 bits hasta los 4096 bits.

Dada la importancia cotidiana de este tema, una propuesta de criptografía, que ha crecido en los últimos años, es la criptografía con curvas elípticas, la cual es considerada dentro de la criptografía de clave pública. Esta propuesta tiene como ventaja de que requiere de una cantidad menor de bits para las claves pública y privada, comparada con ElGamal o RSA. A pesar de lo anterior, la reducción del tamaño de claves no

reduce la fortaleza del cifrado de datos, por el contrario, se considera que tiene la misma fortaleza. De ahí lo atractivo de incursionar en esta línea, para evaluar y explotar sus ventajas.

La Criptografía con Curvas Elípticas es atractiva para su implementación sobre dispositivos programables debido a que existen curvas elípticas que están definidas en campos binarios, es decir, con base 2.

Es por lo mencionado anteriormente, y además de que actualmente se están utilizando cada vez más dispositivos con reducida capacidad de almacenamiento, como lo son las tarjetas inteligentes (o también llamados *Smart Cards*), sistemas embebidos y en el Internet de las Cosas (IoT, *Internet of the Things*), que se deben buscar métodos eficientes para la implementación de los algoritmos que reduzcan los recursos y, además, mejoren el tiempo de respuesta o procesamiento.

1.2. Redes Neuronales Artificiales en Criptografía

1.2.1. Las Redes Neuronales Artificiales

Las *Redes Neuronales Artificiales* son un modelo matemático-computacional que tratan de emular el comportamiento de sus contrapartes biológicas, como, por ejemplo, las neuronas en el cerebro humano. Sin embargo, estas redes de neuronas artificiales no tienen la capacidad de procesar información de las biológicas, sino son una simple abstracción de aquellas [4].

Desde su creación y propuesta, las redes neuronales han sido utilizadas para resolver diferentes tareas, como, por ejemplo, el procesamiento de imágenes, clasificador de imágenes, reconocimiento de objetos y de texto en imágenes, creación de mensajería (chatbot), reconocimiento de voz, traductor de idiomas, entre otros.

Existen diferentes modelos de redes neuronales, dentro de las cuales podemos encontrar las *Redes Perceptrón Multicapa* (MLP de Multi-Layer Perceptrón), las cuales son usadas para clasificación; las *Redes Neuronales Convolucionales* (ConvNets

de Convolutional Network o CNN de Convolutional Neural Network), las cuales son usadas para clasificar imágenes; las *Redes Neuronales Celulares* (CNN o CeNN[1] de Cellular Neural Network), las cuales son usadas para el procesamiento de imágenes; las *Redes Neuronales Pulsadas* (SNN de Spiking Neural Network), las cuales son usadas para clasificación; las *Redes Neuronales Profundas* (DNN de Deep Neural Network), las cuales son para realizar diversas tareas relacionadas con texto, voz, etc.; entre otras.

Todos los modelos de redes neuronales artificiales mencionadas anteriormente pueden ser utilizadas por separado o pueden ser combinadas para resolver tareas cada vez más complejas. Como un ejemplo, pueden ser conjuntadas las ConvNets junto con las MLP para clasificar imágenes o para el reconocimiento de objetos dentro de imágenes, ejemplos de éstos los podemos encontrar en [5–7].

Una de las redes neuronales que es atractiva para su implementación, debido a los resultados que arroja, es la CeNN. Este tipo de red neuronal, dentro de sus múltiples usos, tiene la capacidad de realizar la función de oscilador caótico, la cual es muy utilizada en criptografía para la generación de números aleatorios y éstos a su vez son generadores de claves criptográficas. Otra tarea que realiza, dando resultados de gran calidad, es como procesador de imágenes [8]. Por lo anterior, el empleo y propuesta del uso de las redes neuronales artificiales será una de las herramientas para la generación de información importante y regularmente empleada a cifrar mediante los métodos que se abarcarán en la presente tesis.

La CeNN puede realizar diferentes tareas sobre imágenes, ya sea tanto en imágenes en escala de grises, así como en escala a colores. Dentro de estas tareas podemos encontrar, por ejemplo, la búsqueda de contornos, la búsqueda de esquinas, la reducción de ruido, entre otros. Todas estas tareas las puede desempeñar la CeNN, por lo que puede ser utilizada para los procesamientos anteriormente dichos.

Las ecuaciones matemáticas que rigen el comportamiento de las CeNN son complejas. Esto provoca que su implementación en sistemas embebidos requiere una alta

[1]En este trabajo se utilizará la notación de CeNN para no confundir con la notación de las redes neuronales convolucionales CNN

demanda de recursos. Es por lo anterior que se deben de buscar métodos que reduzcan el gran consumo de hardware.

1.2.2. Uso de Redes Neuronales en Criptografía

En la literatura se pueden encontrar diferentes propuestas para el uso de las redes neuronales para el cifrado de datos o para la generación de diferentes partes que se requieren para hacer uso en la criptografía.

La información, que se puede y debe cifrar, es la que se genera con las imágenes. Hoy en día, el procesamiento de imágenes es una tarea altamente usada en la vida cotidiana. Ejemplos de lo anterior, los podemos encontrar en lectores de código *QR*, reconocimiento facial, lector de retina, clasificador de objetos, reconocimiento de caracteres alfanuméricos, entre otros.

Dentro de las propuestas de las redes neuronales para el uso en el cifrado y descifrado de datos, podemos encontrar las CeNNs. Algunas de las propuestas de estas redes para el uso en criptografía las podemos encontrar como calculador de funciones Hashv [9], como funciones no clonables [10] o, más importante para este trabajo, para el cifrado de imágenes [11, 12].

Otro modelo de red neuronal que ha sido propuesto para ser utilizado en criptografía tiene que ver con las ConvNets. Dentro de estas propuestas podemos encontrar las siguientes: R. Forgáč y M. Očkay las han utilizado como cifrador simétrico [13], mientras que J. Roh et al., lo han propuesto como generador de clase biométricas [14].

Aunque existen diversas propuestas del uso de redes neuronales artificiales en criptografía como se ha mencionado anteriormente, pocas de ellas hablan sobre la implementación de dichas redes sobre dispositivos programables, como lo son los microcontroladores, los microprocesadores, los arreglos de compuestas lógicas programables en campo (FPGA de *Field Programmable Gate Array*), etc., y más aún, no se han explorado del todo las vulnerabilidades que se puedan dar sobre el uso de redes neuronales en criptografía sobre hardware. Por lo tanto, es un campo abierto a

las investigaciones y propuestas sobre sistemas y métodos de cifrado seguro y en este trabajo se realiza el estudio de una metodología para la implementación de algoritmos de seguridad para el cifrado y descifrado seguro, de imágenes tratadas con CeNNs.

1.3. Ataques a los sistemas criptográficos

Todos los sistemas criptográficos han sido vulnerados, en otras palabras, han sido atacados ya sea para encontrar las claves de cifrado o encontrar el mensaje que fue cifrado (mensaje original), para esto se han utilizado diferentes técnicas. Generalmente, todas estas técnicas para vulnerar los sistemas criptográficos se han realizado a nivel de código (software). Dentro de estas técnicas se pueden encontrar los ataques por búsqueda de claves por fuerza bruta, ataques que buscan alguna debilidad en el algoritmo criptográfico, los ataques que buscan algún fallo en la implementación del algoritmo y finalmente se aplican técnicas de criptoanálisis (el criptoanálisis se encarga del estudio de los sistemas criptográficos).

Sin embargo, Kocher en la década de los 90's se dio cuenta que se podían obtener las claves utilizadas durante el cifrado o descifrado de datos a través de mediciones físicas sobre los dispositivos que ejecutan el algoritmo criptográfico [13, 15, 16], llegando así a los ataques sobre hardware. Estos ataques han sido denominados *Ataques por Canal Lateral* (SCAs de Side Channel Attacks).

Los SCAs se pueden dividir principalmente en dos categorías: los *ataques activos* y los *ataques pasivos*.

1.3.1. Ataques activos

Cuando una persona no autorizada desea conocer la clave secreta o la información que fue cifrada, realiza ataques que modifican el funcionamiento del algoritmo que está siendo ejecutado en el dispositivo criptográfico. De esta forma, el atacante puede notar diferencias durante la ejecución de los algoritmos criptográficos y así obtener las claves.

Este tipo de ataques son invasivos, esto quiere decir que un atacante utiliza diversos métodos para modificar la operación del dispositivo, por ejemplo, el atacante puede inducir una señal en el dispositivo, para así hacer que el dispositivo salte pasos del algoritmo o lea un valor incorrecto. Otro ejemplo de este tipo de ataques que se han realizado es el ir eliminando las capas que protegen a los dispositivos criptográficos y así realizar un análisis de la construcción del circuito integrado.

Con lo anterior, podemos concluir que este tipo de ataques dañan el dispositivo criptográfico, para lo cual en estos ataques se debe de contar con un número considerable de dichos dispositivos criptográficos. Un ejemplo de este tipo de ataques son los llamados *Ataques por Falla* (FA de Fault Attacks) [17]. Generalmente, este tipo de ataques requieren de un equipo especializado, lo cual lo hace costoso y poco práctico para atacantes individuales, pero no así para grupos de atacantes que tengan los recursos suficientes.

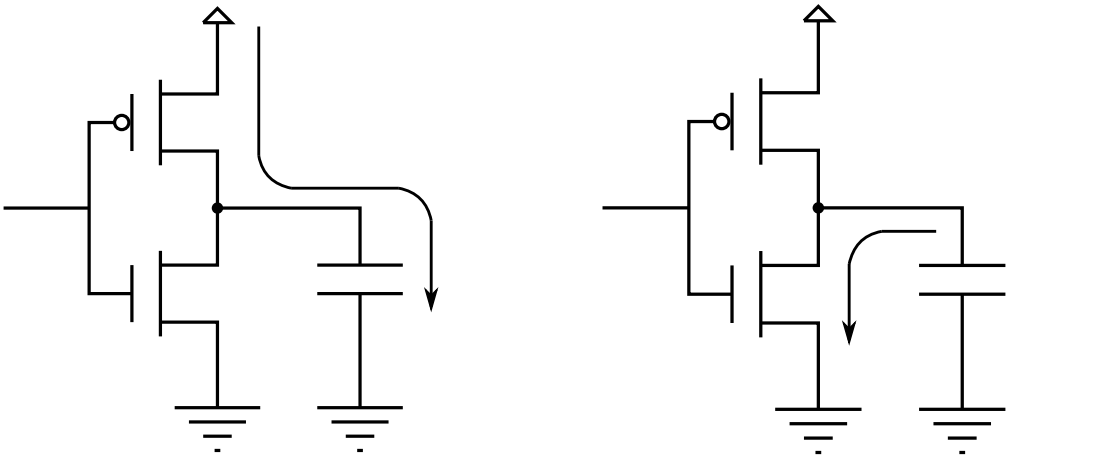
1.3.2. Ataques pasivos

Al contrario de los ataques activos, en los cuales se llegan a dañar los dispositivos, en este tipo de ataques no se modifica la operación de dichos dispositivos, por lo menos no de manera física; es por eso, por lo que a este tipo de ataques también se les conoce como ataques no invasivos.

Dentro de este tipo de ataques existen tres principales: los *Ataques por Análisis de Potencia* (PAA de Power Analysis Attack) [16], los *Ataques por Análisis de Tiempo* (TAA de Timing Analysis Attack) [15] y los *Ataques por Análisis Electromagnético* (EAA de Electromagnetic Analysis Attack) [18].

Para realizar los PAAs, el atacante solo requiere medir el consumo de potencia del dispositivo criptográfico durante la ejecución del algoritmo de cifrado o descifrado de la información. A partir de esto, el atacante toma ventaja del hecho de que el consumo de potencia está directamente relacionado con el algoritmo de cifrado/descifrado de la información.

Los dispositivos criptográficos son dispositivos digitales que pueden ser diseñados en *Circuitos Integrados de Propósito Específico* (ASICs de Application-Specific Integrated Circuit), en dispositivos programables como los FPGAs o en sistemas embebidos como los microcontroladores. Sin embargo, todos estos dispositivos basan su funcionamiento en los *Transistores Metal-Óxido-Semiconductor Complementarios* (CMOS de Complementary Metal-Oxide-Semiconductor). Para ejemplificar cómo es posible la medición de potencia se usará un inversor realizado en tecnología CMOS, el cual se ilustra en la Fig. 1.1.



(a) Flujo de corriente hacia la carga, que representa un 1 lógico.

(b) Sin flujo de corriente hacia la carga, que representa un 0 lógico.

Fig. 1.1. Flujo de corriente en un inversor CMOS.

De la Fig. 1.1a, se puede observar que fluye una corriente hacia la carga (“1” lógico), sin embargo, en la Fig. 1.1b se observa cómo no hay flujo de corriente hacia la carga (“0” lógico). Con esto se puede ver que la potencia consumida será diferente si se procesa un bit con valor de “1” al procesado con un valor de “0”. De lo anterior, se ve claramente que se puede determinar si durante la ejecución del algoritmo de cifrado se ejecutó un “1” o un “0” y de esta forma obtener la clave secreta.

Los TAAs, se realizan midiendo el tiempo que tarda un algoritmo de cifrado/descifrado en realizar las operaciones. Sin embargo, este tipo de ataques han ido quedando obsoletos debido a que actualmente se han modificado los algoritmos para que durante el cifrado/descifrado de la información tome el mismo tiempo de ejecución, sin importar el valor de la clave secreta o de la información.

Finalmente, los EAAs basan su funcionamiento en los PAAs, pero para éstos se requiere una sonda que sea capaz de medir los campos electromagnéticos que se generan en el dispositivo criptográfico.

1.4. Protección de los dispositivos criptográficos

Dentro de la literatura se han propuesto diversas metodologías para evitar los ataques tanto hacia los dispositivos criptográficos (hardware) como a los algoritmos de cifrado/descifrado de la información (software). Esas propuestas se pueden dividir en dos: 1) los métodos de protección a nivel de hardware y 2) los métodos de protección a nivel de algoritmo, [19].

Dentro de la categoría de protección a nivel de hardware podemos encontrar, entre otros, las siguientes propuestas:

- El uso de un supresor de señales integrado en el circuito cifrador [20]. Esta técnica ha sido desarrollada para evitar los PAAs y se basa en censar el consumo de potencia del circuito cifrador y al mismo tiempo generar un consumo de potencia constante.

- Otra medida propuesta es el uso del diseño con CMOS de ultra bajo voltaje de compuerta [21]. Principalmente esta propuesta utiliza la tecnología de compuerta flotante para el diseño de las compuertas lógicas. Con el uso de esta tecnología se pretende reducir el consumo de potencia de los dispositivos criptográficos y de esta forma sea difícil realizar las mediciones por un atacante.

- Una tercera medida es la encontrada en [22], donde se proponen diferentes estructuras que modifican el consumo de potencia para las diferentes compuertas lógicas, de esta manera se obtiene un consumo constante sin importar de qué compuerta lógica se trate.

- Finalmente, existen diferentes propuestas para evitar los SCAs sobre los dispositivos programables. En los trabajos [23, 24] se pueden encontrar algunos de

ellos. Dentro de estas propuestas destaca el Doble Riel (dual-rail), el cual se basa en realizar operaciones con el bit contrario al que está trabajando la implementación, es decir, si se ejecuta un "1" se hace la misma operación, pero ahora con un "0", estas dos operaciones se realizan al mismo tiempo. Obteniendo así, un complemento en el consumo de potencia y de esta forma tener siempre un mismo valor de potencia sin importar la operación que se esté realizando.

Las propuestas anteriores tienen la desventaja de que se incrementa el consumo de recursos para la implementación del dispositivo criptográfico. De aquí que es importante buscar un método para reducir el consumo de recursos mientras se mejora la protección de los dispositivos. Por otra parte, también se han realizado diversas propuestas para la protección de los algoritmos criptográficos a nivel de software. Algunas de estas propuestas son:

- El mensaje por cifrar se enmascara utilizando un valor aleatorio. Para recuperar el mensaje original se utiliza el mismo valor aleatorio.
- Para evitar los TAAs, los algoritmos de cifrado/descifrado se han modificado de tal manera que realicen la misma cantidad de operaciones, sin importar el valor de la clave secreta o pública, y con esto obtener el mismo tiempo de procesamiento.
- Las claves secretas se han incrementado en tamaño, para que así sea más complicado encontrar su valor a través de ataques por fuerza bruta.

Por lo anterior, que se deben de buscar metodologías que reduzcan las vulnerabilidades de los algoritmos y de sus respectivas implementaciones en dispositivos programables.

1.5. Conclusiones

De los diversos métodos que existen para el cifrado/descifrado de la información, el llamado ECC es atractivo para su implementación sobre dispositivos programables,

como lo es el FPGA. Esto es debido, a que con este método se requiere una menor cantidad de bits para su cifrado/descifrado de información comparados con otros métodos de criptografía de clave pública y con una cantidad cercana de bits a los métodos de criptografía de clave privada.

La utilización de redes neuronales artificiales en sistemas criptográficos ya sea como parte del sistema o para proteger los datos entregados por estos, han sido poco reportados. Los trabajos reportados son atractivos desde el punto de vista de software.

Como se ha mencionado anteriormente, la seguridad de los dispositivos que manejan o tratan información es importante. Se debe poner atención especial cuando se implementa criptografía en dispositivos programables. Los SCAs deben ser considerados de gran importancia al momento de implementar cualquier método criptográfico en dichos dispositivos programables.

Finalmente, los sistemas criptográficos que utilizan redes neuronales artificiales han sido poco estudiados. Además, no se ha realizado un estudio del comportamiento de estos sistemas sobre ataques como los llamados SCAs, más aún, estos sistemas no han sido reportados sobre dispositivos programables que realizan el procesamiento de imágenes. Lo anterior los hace atractivos para su estudio.

Capítulo 2

Redes Neuronales Artificiales

2.1. Introducción

Actualmente el procesamiento de imágenes es una tarea importante dado que con ella se puede mejorar la calidad de la imagen, eliminar el ruido, resaltar y/o detectar los bordes, proyección de sombras, detección de componentes conectados, entre otras tareas que se requieren para determinados propósitos. Todas estas técnicas tienen aplicación, por ejemplo: en imágenes médicas, videollamadas, reconocimiento de objetos y caracteres alfanuméricos, aplicaciones militares, robótica, etc. Es por esto por lo que se han propuesto diversos métodos para el procesamiento digital de imágenes; sin embargo, estos tienen la desventaja de que funcionan sobre procesadores digitales.

Aunque los procesadores digitales funcionan a grandes velocidades que van desde varios megahertcios (MHz) hasta los gigahertcios (GHz), éstos tienen la desventaja de que funcionan en forma serial, es decir, realizan operaciones una después de otra.

Hoy en día las imágenes son de un formato de gran tamaño que van desde unos cuantos pixeles hasta varios mega pixeles. Este gran tamaño de las imágenes actuales hace que el procesamiento de éstas utilizando procesadores digitales sea lento.

Debido a lo anterior, se han propuesto diversas soluciones para mejorar la velocidad y la calidad del procesamiento de imágenes, entre estas propuestas se encuentras

los autómatas celulares, las retinas de silicio, las Redes Neuronales Celulares, entre otras.

Las Redes Neuronales Celulares (a partir de este momento únicamente las mencionaremos como CeNNs) fueron propuestas por primera vez por *L. O. Chua* y *L. Yang* en el año de 1988 en la *Universidad de Berkeley* [1]. Estas CeNNs se han utilizado para la realización de tareas como lo son la solución de ecuaciones diferenciales, la creación de osciladores caóticos (estos osciladores caóticos son a su vez utilizados para la generación de claves criptográficas) y por supuesto, el procesamiento de imágenes. Las CeNNs tienen la ventaja de que realizan procesamiento paralelo, lo cual reduce significativamente el tiempo de procesamiento de las imágenes.

2.2. Las Redes Neuronales Celulares Digitales

Las CeNNs reúnen dos características importantes de dos modelos computacionales: las *redes neuronales* y los *autómatas celulares*. De las redes neuronales, las CeNNs toman la característica del procesamiento paralelo y de los autómatas celulares toman el mecanismo de interconexión.

La unidad básica que compone una CeNN es llamada célula. Esta *célula* es replicada para formar una interconexión como la mostrada en la Fig. 2.1. Aunque esta interconexión puede ser de n dimensiones, generalmente, es utilizada en dos dimensiones para el procesamiento de imágenes en escala de grises y en tres dimensiones para el procesamiento de imágenes a color [8]. En este trabajo nos enfocaremos en imágenes en escala de grises, es decir en CeNN de dos dimensiones.

Al conjunto de células interconectadas con la célula $C(i,j,k)$ en la Fig. 2.1 se le llama vecindario $N_r(i,j,k)$, el cual tiene un radio r. En la Fig. 2.2 se muestra un vecindario para una célula $C(i,j)$ (en una CeNN de dos dimensiones) con un radio $r = 1$, es decir, un vecindario de $N_r = 3 \times 3$ células.

Dentro de la estructura de una CeNN existen células en la frontera las cuales no tienen un vecindario completo debido a la inexistencia de células fuera de la CeNN.

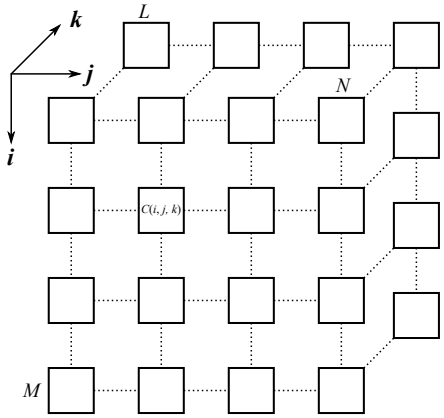


Fig. 2.1. Interconexión de una Red Neuronales Celular

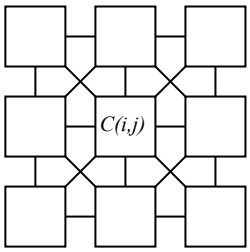


Fig. 2.2. Vecindario de 3×3 células pertenecientes a la célula $C(i, j)$ dentro de una CeNN de dos dimensiones.

Existen tres condiciones de frontera para solventar con el problema de estas células [8], las cuales se mencionan a continuación:

1. *Condición de frontera fija (Dirichlet)*. En esta condición se agregan células en la frontera con valores fijos en cada una de éstas.

2. *Condición de frontera de flujo-cero (Neuman)*. Al igual que la condición de frontera fija, se agregan nuevas células, sin embargo, los valores de cada una de estas células son fijados a los valores de las células en la frontera.

3. *Condición de frontera periódica (Toroidal)*. En esta condición de frontera, a diferencia de las dos previas, no se agregan nuevas células; en su lugar para completar el vecindario, las células son fijadas a las células que se encuentran en el otro extremo de la CeNN.

Para la reducción de la complejidad del diseño en el dispositivo programable, en las simulaciones y durante el entrenamiento, se utilizará la condición de frontera fija

o también llamada Dirichlet. El valor fijo de frontera será cero.

La evolución de una célula analógica dentro de una CeNN está definida por su ecuación de estado [1]. Sin embargo, para poder ser implementada en dispositivos programables, como lo son los FPGAs, la ecuación de estado ha sido aproximada por el método de Euler, dicha ecuación se muestra en (2.1).

$$v_{xij}(n+1) = v_{xij}(n) + h[-v_{xij}(n) + f_{ij}(n) + I_{kl}] \tag{2.1}$$

Donde:

$$f_{ij} = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(n) \tag{2.2}$$

y

$$I_{kl} = \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{ukl}(n) + I_{ij} \tag{2.3}$$

De las ecuaciones anteriores, n es el paso de integración en el método de Euler, h es el paso de tiempo. La célula $C(k,l)$ es la célula dentro del vecindario de la célula $C(i,j)$; el valor de $v_{xij}(n)$ es el estado de la célula en un determinado paso de tiempo; v_{ykl} y v_{ukl} representan las salidas y las entradas de las células vecinas respectivamente; I_{ij} es el umbral correspondiente a la célula $C(i,j)$; $A(i,j;k,l)$ es la plantilla de retroalimentación; y finalmente $B(i,j;k,l)$ es la plantilla de control.

Las plantillas **A** y **B**, junto con el valor del umbral I son los parámetros que determinan la tarea que va a realizar la CeNN.

La salida de la célula, al igual que en el resto de las redes neuronales artificiales, está definida por una función de activación. Para el caso concreto del procesamiento de imágenes, la función de activación utilizada es la función de activación de saturación lineal simétrica o también llamada *satlins*, cuya ecuación se muestra a continuación:

$$v_{yij}(n) = 0.5(|v_{xij}(n)+1| - |v_{xij}(n)-1|) \tag{2.4}$$

La salida de una célula en la CeNN determina el valor de color que tendrá la salida. Para nuestro caso, y utilizando la función de activación satlins, se tiene una salida en el rango $[-1.0, +1.0]$. El valor de -1.0 indica que se tiene un píxel blanco, un valor de $+1.0$ indica que se tiene un píxel negro, y los valores intermedios a estos indican que se tiene un píxel en escala de grises.

Como se verá más adelante, el obtener imágenes monocromáticas, es decir, en blanco y negro, nos ayudará a reducir el consumo de recursos en hardware, además de una reducción de la cantidad de datos a cifrar.

2.3. Plantillas de una CeNN

Los parámetros que determinan la tarea que realiza una CeNN son las plantillas de retroalimentación (plantilla **A**) y control (plantilla **B**) junto con el umbral (I). Para el caso de una CeNN con un vecindario de 3×3 células, las plantillas son representadas como siguen:

$$\mathbf{A} = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix}; \mathbf{B} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \tag{2.5}$$

A la operación de multiplicar las correspondientes plantillas con su respectivo valor de entrada o salida del vecindario de la célula $C(i, j)$ se le llama convolución espacial [8].

Para que el comportamiento de una CeNN sea estable, es decir, que no oscilen las células durante la evolución de éstas, las plantillas deben de cumplir con ciertas condiciones [1, 8]. Estas condiciones son las siguientes: los valores de las plantillas para **A** deben ser simétricas, es decir, $a_{-1,-1} = a_{1,1}$, $a_{-1,0} = a_{1,0}$, $a_{-1,1} = a_{1,-1}$ y

finalmente $a_{0,-1} = a_{0,1}$; esto se repite para el caso de la plantilla **B**.

Para el caso de que queramos utilizar la CeNN como oscilador caótico, las condiciones anteriores no deben ser tomadas en cuenta. Recordando que un oscilador caótico es importante dentro de la criptografía como generador de números aleatorios y estos a su vez como generadores de claves.

Generalizando las ecuaciones de (2.5) y tomando las consideraciones anteriormente mencionadas para el procesamiento de imágenes, podemos reescribirlas de la siguiente forma:

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_4 \\ a_3 & a_2 & a_1 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_4 \\ b_3 & b_2 & b_1 \end{bmatrix} \tag{2.6}$$

Esta nueva notación será útil (como se verá más adelante) para el cálculo, utilizando algún método de optimización para una determinada tarea que se desee realizar con la CeNN.

2.3.1. Búsqueda de plantillas

Como en todas las estructuras neuronales artificiales, en las CeNNs no existe un único método para encontrar los valores de las plantillas para que realicen un determinado procesamiento, sin embargo, estos problemas se tratan como un problema de optimización. Dentro de los métodos de optimización encontramos dos principalmente: los *métodos analíticos* y los *métodos heurísticos*.

Los métodos analíticos se basan en crear una serie de reglas locales que caracterizen la célula dentro de la CeNN. Para realizar esto, se crean una serie de ecuaciones que después son resueltas con algún algoritmo de optimización, como lo son los *algoritmos de aprendizaje local* y los *algoritmos de aprendizaje global*. Sin embargo, este último método va incrementando su nivel de dificultad conforme va incrementando el número de variables, esto propicia el aumento de la complejidad para encontrar la

solución óptima de las ecuaciones desarrolladas.

Por otro lado, los métodos heurísticos se han desarrollado en los últimos años para cubrir los inconvenientes de los métodos analíticos. Dentro de estos métodos, se pueden encontrar los siguientes algoritmos: *Optimización de Nubes de Partículas* también llamado PSO (de Particle Swarm Optimization) [25, 26]; *Algoritmos Genéticos* también llamados GA (de Genetic Algorithm) [27, 28]; *Algoritmo Colonia de Abejas Artificial* también llamado ABC (de Artificial Bee Colony) [29, 30]; y finalmente el método *Nelder-Mead* también conocido como NM [31, 32].

2.4. Búsqueda de plantillas con ABC y NM

El algoritmo ABC ha surgido como una idea interesante basada en el comportamiento de las abejas para la búsqueda de alimento como un método de optimización. Este algoritmo ha demostrado entregar buenos resultados como método de optimización para funciones multivariables [33, 34]. Además, ha sido utilizada con éxito para la búsqueda de plantillas para la CeNN como en [30, 34].

2.4.1. El algoritmo ABC

Este algoritmo ABC se basa en el principio de búsqueda de alimento que realizan las abejas. Este algoritmo pertenece a los algoritmos llamados bio-inspirados. Dentro de este algoritmo existen tres tipos de abejas: las *abejas trabajadoras*, las *abejas observadoras* y las *abejas exploradoras* [29]. El comportamiento de cada una de este tipo de abejas es explica a continuación:

- Las abejas trabajadoras EB (de *employed bees*), son enviadas a las diferentes fuentes de alimento y calculan la cantidad de alimento que existe a su alrededor. En términos matemáticos, estas abejas son enviadas a las posibles soluciones del problema planteado y calculan qué tan buena es esa solución y si hay alguna mejor en su alrededor.

- Las abejas observadoras OB (de *onlooker bees*), son enviadas a las fuentes de alimento y buscan en los alrededores de estos una mejor fuente de alimento. En otras palabras, estas abejas son enviadas a las posibles soluciones del problema, buscan en los alrededores una mejor solución y si encuentran alguna mejor, ésta es actualizada.

- Finalmente, las abejas exploradoras SB (de *scout bees*), son enviadas en la búsqueda de nuevas fuentes de alimento en un área determinada. Es decir, estas abejas son enviadas a la búsqueda de nuevas soluciones a las ya encontradas dentro del área de solución, de acuerdo con un valor llamado *Limite* (*Lim*).

El algoritmo general, para la optimización, es presentado en Algoritmo 2.1 [29]:

Algoritmo 2.1 Algoritmo general ABC

1: Asignar los parámetros del ABC.
2: Inicializar el enjambre de abejas.
3: Calcular la calidad inicial de las fuentes de alimento iniciales.
4: **mientras** el criterio de parada no es alcanzado **hacer**
5: Enviar a las EB a las fuentes de alimento y calcular su calidad.
6: Enviar a las OB a buscar a los alrededores de las fuentes de alimento y calcular su calidad.
7: Enviar a las SB en búsqueda de nuevas fuentes de alimento, esto de acuerdo al valor de Lim y calcular su calidad.
8: Memorizar la mejor fuente de alimento hasta el momento.
9: **fin mientras**

Del algoritmo anterior, el paso 1 asigna los siguientes parámetros: Se define el tamaño de la colonia *CS* (de *colony size*), este tamaño de colonia se divide en dos partes, la primera mitad representará la cantidad de abejas trabajadoras EB y la segunda mitad serán las abejas observadoras OB, finalmente la cantidad de fuentes de alimentos representarán a las abejas exploradoras SB, la cantidad de estas abejas se definirá como la misma EB y OB. De lo anterior definimos $EB = OB = SB$. Además, se define la cantidad de variables, D, del problema. Por último, se define el área de búsqueda de alimentos.

En el paso 2, todas las variables son inicializadas según los datos ingresados en el paso 1. En el paso 3 se envían las abejas exploradoras en búsqueda de alimento utilizando (2.7).

$$x_i = x_j^{min} + rand(0,1)(x_j^{max} - x_j^{min}) \tag{2.7}$$

Donde $i = 1, \cdots, SB$; $j = 1, \cdots, D$; y x_j^{min} junto con x_j^{max} representan los límites del área de búsqueda.

En el paso 5, se envían a las abejas trabajadoras v_i a los alrededores de las abejas exploradoras en búsqueda de un mejor alimento, esto es realizado utilizando la siguiente ecuación:

$$v_i = v_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}) \tag{2.8}$$

Donde $-1 \leq \phi_{i,j} \leq 1$ y $k = 1, \cdots, SB$, además se debe de cumplir que $k \neq j$, y ambos índices se escogen aleatoriamente. Para determinar si el nuevo alimento es mejor que el original se aplica una función de aptitud. La función de aptitud es determinada por el tipo de problema a tratar, si resulta mejor la nueva fuente de alimento ésta sustituye a la original. Además de esto, se debe asegurar que la nueva fuente de alimento se encuentre dentro del área de búsqueda.

En el paso 6, se calcula la calidad de cada una de las fuentes de alimento y basado en esta calidad, se envía una cantidad de abejas exploradoras a buscar en los alrededores una mejor fuente de alimento. Para el envío de las abejas exploradoras se utiliza un cálculo de distribución de probabilidad. Como ejemplo, a continuación, se muestra una función de cálculo de probabilidad utilizada en la literatura:

$$P(x_i) = \frac{f(x_i)}{\sum_{i=1}^{SB} f(x_i)} \tag{2.9}$$

En el paso 7, se determina si alguna fuente de alimento no tiene mejora después de determinados ciclos, si es así, en éste se envía a una abeja exploradora en búsqueda de una nueva fuente de alimento; esto se realiza utilizando (2.7).

Finalmente, en el paso 8, se memoriza la mejor fuente de alimento encontrada hasta el momento.

2.4.2. El algoritmo Nelder-Mead

El algoritmo Nelder - Mead (NM), también conocido como Downhill Simplex Method, es una técnica metaheurística ya que incluye elementos de programación dinámica, y ha sido utilizada para la búsqueda de plantillas, como en [31, 32].

Antes de realizar el algoritmo NM, primero es necesario definir una serie de puntos iniciales sobre los cuales se comenzará la búsqueda de un mínimo utilizando este método. La cantidad de puntos mínimos serán $n+1$ puntos, es decir, se tendrán tres puntos para un problema bidimensional, cuatro puntos para un problema tridimensional, etc.

Después son definidos los siguientes puntos: el punto que da el máximo (x_{h1}), el punto que da el segundo máximo (x_{h2}), el punto que da el mínimo (x_{min}) y finalmente se calcula el punto promedio de todos, también conocido como centroide (x_0).

Finalmente, tres operaciones son definidas: la primera se conoce como reflexión la cual se muestra en (2.10), la segunda llamada expansión la cual se muestra en (2.11), y por último el punto conocido como contracción el cual se muestra en (2.12).

$$x_r = x_0 + \alpha(x_0 - x_{h1}) \tag{2.10}$$

$$x_e = x_0 + \gamma(x_r - x_0) \tag{2.11}$$

$$x_c = x_0 + \beta(x_{h1} - x_0) \tag{2.12}$$

Los valores de α, γ y β son valores obtenidos heurísticamente. El algoritmo NM se muestra en la Fig. 2.3.

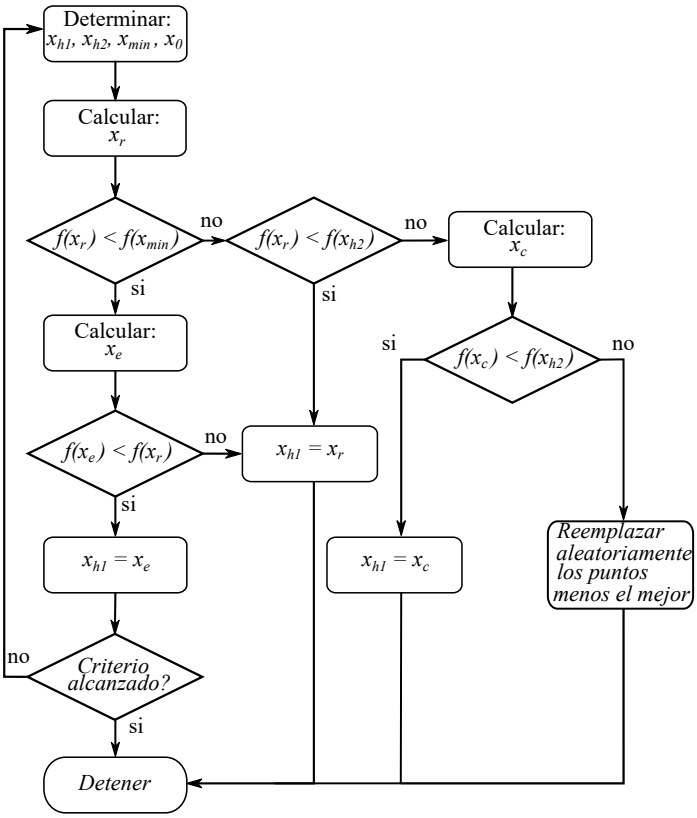


Fig. 2.3. Algoritmo Nelder-Mead

2.4.3. Búsqueda de plantillas utilizando un algoritmo propuesto ABC y NN hibrido

Los algoritmos ABC y NM son buenos para la búsqueda de mínimos, sin embargo, éstos tienen inconvenientes. El algoritmo ABC, aunque puede converger a un mínimo global, requiere la evaluación de una gran cantidad de puntos aleatorios para encontrar el mínimo global. Por otro lado, el algoritmo NM converge rápidamente a un mínimo con una menor cantidad de puntos que ABC, sin embargo, el mínimo puede no ser un mínimo global sino un mínimo local.

Dadas las restricciones mencionadas anteriormente, se propone una metodología para la búsqueda de mínimos utilizando una solución híbrida de los algoritmos ABC y NM. El algoritmo propuesto se muestra en Algoritmo 2.2.

Para la búsqueda de plantillas de una CeNN utilizando el algoritmo de optimización ABC y NM, se definen las plantillas como una función multivariable. Esto se

Algoritmo 2.2 Algoritmo híbrido ABC y NM

1: Asignar los parámetros del ABC y NM.
2: Inicializar el enjambre de abejas del ABC.
3: Calcular la calidad inicial de las fuentes de alimento iniciales de EB.
4: **mientras** el criterio de parada no es alcanzado **hacer**
5: Enviar a las EB a las fuentes de alimento y calcular su calidad.
6: Aplicar la selección del mejor.
7: **mientras** OBs **hacer**
8: Aplicar NM utilizando OBs.
9: Aplicar la selección del mejor.
10: **fin mientras**
11: Enviar a las SB en búsqueda de nuevas fuentes de alimento y calcular su calidad. Esto es realizado en función de Lim.
12: Memorizar la mejor fuente de alimento hasta el momento.
13: **fin mientras**

logra viendo los valores de las plantillas como un vector. De (2.6) se puede obtener el vector mostrado en (2.13).

$$x_i = [a_1, a_2, a_3, a_4, a_5, b_1, b_2, b_3, b_4, b_5, c] \tag{2.13}$$

La metodología utilizada para la búsqueda de plantillas para una CeNN utilizando el método de optimización híbrido ABC y NM se muestra en el diagrama a bloques de la Fig. 2.4.

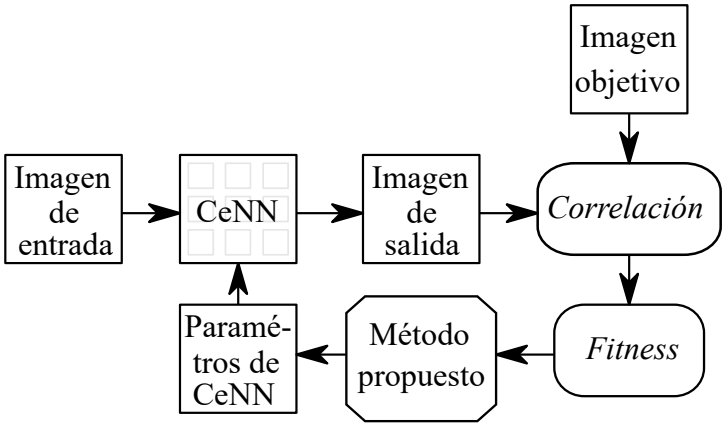


Fig. 2.4. Diagrama de flujo para la búsqueda de plantillas utilizando ABC y NM

En términos generales, primero se define una imagen de entrada y su respectiva imagen objetivo. Esta imagen estará determinada por la tarea que se desea realizar

con la CeNN. El algoritmo propuesto genera unas plantillas iniciales aleatorias que son enviadas a la CeNN y así ésta realice su procesamiento. La CeNN genera una serie de imágenes de salida que corresponden a las diferentes plantillas entregadas inicialmente por el algoritmo propuesto.

Teniendo las correspondientes imágenes entregadas por la CeNN, se calcula la correlación de éstas con la imagen objetivo. Se utiliza la función de correlación de imágenes como función de aptitud en el algoritmo propuesto. La función de correlación es la mostrada en [35], en la cual se toman tres características de las imágenes para realizar la similitud entre ambas imágenes, éstas son luminiscencia, contraste y estructural. En (2.14) se muestra diche ecuación.

$$C = \frac{\sum_i^M \sum_j^N (xP_{i,j-\overline{xP}})(xP_{i,j-\overline{xP}})}{\sqrt{((\sum_i^M \sum_j^N (xP_{i,j-\overline{xP}}))^2)((\sum_i^M \sum_j^N (yP_{i,j-\overline{yP}}))^2)}} \tag{2.14}$$

En donde, $xP_{i,j}$ es el valor del píxel de la imagen de salida de la CeNN; $\overline{xP}$ es el promedio de la imagen de salida de la CeNN; $yP_{i,j}$ es el valor del píxel de la imagen objetivo; y, $\overline{yP}$ es el promedio de la imagen objetivo.

Teniendo el valor de Correlación (C), se calcula el valor Fitness ($fit(C)$), este valor fitness depende del problema a tratar. La función $fit(C)$ utilizada en esta tesis es mostrada a continuación:

$$fit(C) = \begin{cases} |C| + 0.5 & C \leq 0 \\ \frac{1}{1+C} - 0.5 & C > 0 \end{cases} \tag{2.15}$$

En el algoritmo original se calcula la calidad de cada imagen y a partir de éstas se envía a las abejas exploradoras en búsqueda de mejores fuentes de alimento según el cálculo de probabilidad. Sin embargo, en este paso se ha optado por no calcular la función de probabilidad, en su lugar se ha buscado primero el mejor de los encontrados hasta el momento y enviar a ese mejor a todas las abejas exploradoras utilizando el algoritmo NM. Hacer lo anterior reduce la cantidad de cálculos durante el entrenamiento o búsqueda de alimento y mejora la búsqueda del mínimo encontrado hasta el

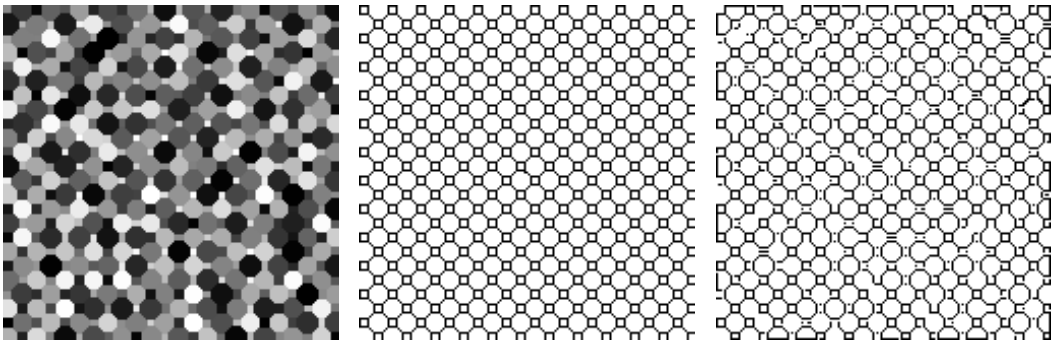
momento.

De acuerdo con los valores de correlación entregados al algoritmo propuesto, se memorizan las mejores plantillas y busca nuevas si se ha llegado al límite establecido.

2.4.4. Resultados de la búsqueda de plantillas

Se realizaron diversas búsquedas de plantillas utilizando diversos valores para el tamaño de la colonia que van desde 22, 44 y 88 abejas. Al igual que en el tamaño de la colonia, se utilizaron diversos valores para las épocas que van desde los 500 y hasta 1000.

La primera tarea seleccionada fue como detector de bordes. En la Fig. 2.5a se muestra la imagen que fue utilizada como entrada, en la Fig. 2.5b la imagen objetivo y en la Fig. 2.5c la imagen obtenida para los mejores valores de plantillas encontradas con el algoritmo propuesto.



(a) Entrada **(b)** Objetivo **(c)** Salida

Fig. 2.5. Imagen de entrada, objetivo y salida obtenidas en el entrenamiento para una CeNN como detector de bordes.

La imagen utilizada para el entrenamiento tiene las siguientes características: tamaño de 165×165 pixeles, una cobertura del $57.81\,\%$ de tonos de grises, es decir, 148 tonos de grises.

Los valores de entrenamiento fueron los siguientes: tamaño de colonia de 22 abejas, 500 épocas, un área de búsqueda de $[-64.0, +64.0]$, $\alpha = 1.0$, $\gamma = 2.0$ y finalmente $\beta = 0.5$. Además, los valores para la CeNN fueron los siguientes: paso de tiempo

$h = 0.03125s$, el tiempo máximo de procesamiento $t_{max} = 1.5s$. Estos valores generaron una imagen de salida con una correlación con la imagen objetivo de 90.06 % y un fitness de 0.0261.

Las plantillas encontradas, con el mejor valor de correlación, se muestran a continuación:

$$\mathbf{A} = \begin{bmatrix} 0.17 & -1.66 & -2.26 \\ 1.64 & 58.33 & 1.64 \\ -2.26 & -1.66 & 0.17 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} 0.54 & -21.21 & 2.33 \\ -32.58 & 49.46 & -32.58 \\ 2.33 & -21.21 & 0.54 \end{bmatrix}; I = -1.64 \quad (2.16)$$

En la Tabla 2.1 se muestra la comparativa de la búsqueda de las plantillas, para la tarea de detector de contornos, utilizando el algoritmo ABC, Nelder-Mead y el propuesto en esta tesis.

Tabla. 2.1. Comparativa de rendimiento para la detección de contornos.

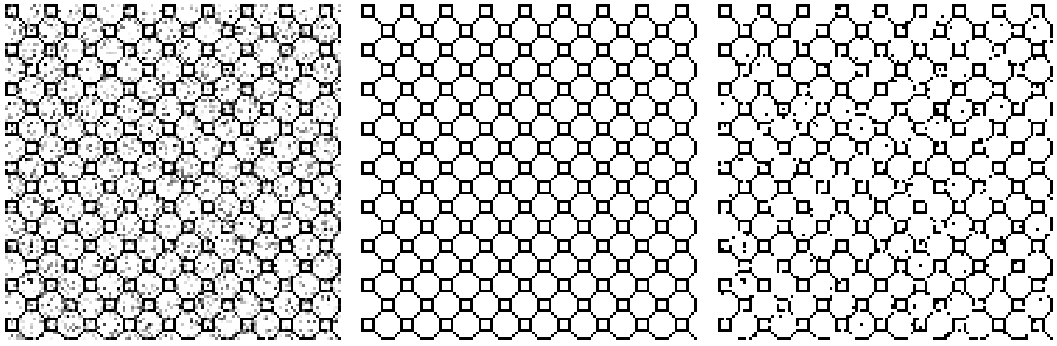
Ciclos	**Nelder Mead**		**ABC**		**Propuesta**	
	C	$fit(C)$	C	$fit(C)$	C	$fit(C)$
100	0.6602	0.1023	0.6860	0.0931	0.8500	0.0405
500	0.8625	0.0368	0.7644	0.0667	**0.9006**	**0.0261**

Cómo se observa, el método propuesto obtiene mejores para la búsqueda de contornos.

Una segunda tarea realizada con el método de búsqueda de plantillas propuesto fue como eliminador de ruido. En la Fig. 2.6a se muestra la imagen de entrada, en la Fig. 2.6b se muestra la imagen objetivo y finalmente en la Fig. 2.6c se muestra la imagen de salida obtenida con las plantillas encontradas.

La imagen utilizada para esta tarea tiene las siguientes características: tamaño de 120×120 pixeles y un ruido del 50 % en escala de grises.

Los valores de entrenamiento fueron los siguientes: tamaño de colonia de 22 abejas, 100 épocas, un área de búsqueda de $[-64.0, +64.0]$, $\alpha = 1.0$, $\gamma = 2.0$ y finalmen-



(a) Entrada **(b)** Objetivo **(c)** Salida

Fig. 2.6. Imagen de entrada, objetivo y salida obtenidas en el entrenamiento para una CeNN como detector de bordes.

te $\beta = 0.5$. Además, los valores para la CeNN fueron los siguientes: paso de tiempo $h = 0.03125s$, el tiempo máximo de procesamiento $t_{max} = 1.0s$. Estos valores generaron una imagen de salida con una correlación con la imagen objetivo de 90.00 % y un fitness de 0.0263.

Las plantillas encontradas se muestran a continuación:

$$\mathbf{A} = \begin{bmatrix} -0.10 & 20.38 & -21.45 \\ 12.59 & 54.39 & 12.59 \\ -21.45 & 20.38 & -0.10 \end{bmatrix}; \mathbf{B} = \begin{bmatrix} -1.07 & 2.63 & 27.07 \\ 3.47 & 59.06 & 3.47 \\ 27.07 & 2.63 & -1.07 \end{bmatrix}; I = -18.97 \quad (2.17)$$

En la Tabla 2.2 se muestra la comparativa de la búsqueda de las plantillas, para la tarea de removedor de ruido, utilizando el algoritmo ABC, Nelder-Mead y el propuesto en esta tesis.

Tabla. 2.2. Comparativa de rendimiento para la tarea de removedor de ruido.

Ciclos	**Nelder Mead**		**ABC**		**Propuesta**	
	C	$fit(C)$	C	$fit(C)$	C	$fit(C)$
100	0.8616	0.0371	0.8691	0.0350	0.8976	0.0269
500	0.8982	0.0268	0.8535	0.0309	**0.9000**	**0.0263**

2.5. Implementación de la CeNN en FPGA

Para poder implementar la CeNN en FPGA primero se realizó el diseño de una célula digital. Se utilizó (2.1) como base para el diseño del circuito de la célula digital. En la Fig. 2.7 se muestra el diagrama a bloques de dicha implementación.

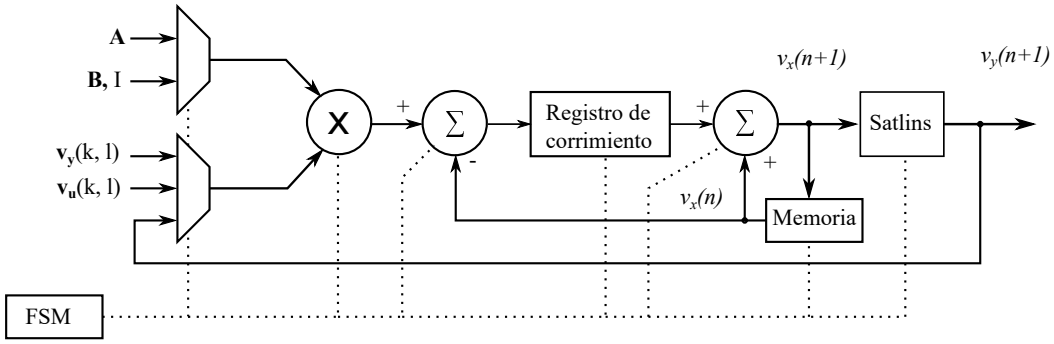


Fig. 2.7. Diagrama a bloques de una célula digital

Los códigos realizados durante la realización de este trabajo pueden ser encontrados en el Apéndice A.

Para controlar el funcionamiento de la célula digital se utilizó una máquina de estados, que en el bloque anterior es nombrado FSM (de *Finite State Machine*).

Esta célula digital se replicó 16 veces para formar una pequeña CeNN de 4×4 células. Comparando los resultados entregados con la reportada en la publicación original de Chua [1], fue comprobado el funcionamiento de esta CeNN digital.

En la Fig. 2.8 se puede observar la evolución de una célula mostrada en el trabajo original de Chua y en la Fig. 2.9 se observa la evolución de una célula digital implementada en FPGA. Los datos ingresados para la simulación en FPGA fueron los mismos que se utilizaron en el trabajo de Chua.

Comprobado el funcionamiento correcto de las células digitales se implementó finalmente una CeNN digital de tamaño 8×8 células, es decir, con un total de 64 células digitales.

Como se mencionó anteriormente, la condición de frontera utilizada para la implementación fue la condición de frontera fija, donde el valor seleccionado fue de 0.0.

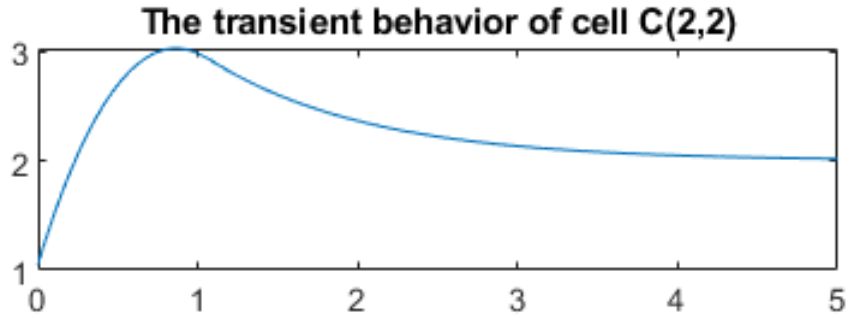


Fig. 2.8. Evolución de una célula mostrada en el trabajo original de Chua [1].

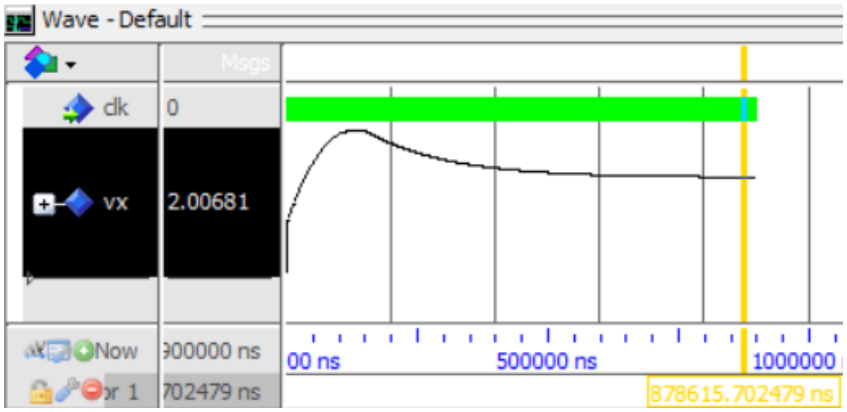


Fig. 2.9. Simulación de una célula digital implementada en FPGA.

2.5.1. Resultados de la implementación de la CeNN en FPGA

Para validar la implementación realizada en FPGA de la CeNN digital, se han realizado las mismas tareas que durante la búsqueda de plantillas. Además, se ha realizado una correlación entre la imagen obtenida con el FPGA y la entregada en simulación en software.

Primero, se comprueba con la tarea de detección de bordes. En la Fig. 2.10 se puede observar la imagen de entrada utilizada, así como la entregada vía software y hardware (FPGA). El tamaño de la imagen de entrada es de 384×384 píxeles. Los valores de paso de tiempo y tiempo máximo de procesamiento fueron las mismas para ambos casos. Las plantillas utilizadas fueron las encontradas con el método propuesto.

Segundo, se comprueba con la reducción de ruido. En la Fig. 2.11 se puede observar la imagen de entrada utilizada, así como la procesada vía software y hardware (FPGA). El tamaño de la imagen es de 254×254 píxeles con ruido agregado.

(a) Entrada **(b)** Salida por software **(c)** Salida por FPGA

Fig. 2.10. Imagen real procesada en software y FPGA para detección de bordes.

(a) Entrada **(b)** Salida por software **(c)** Salida por FPGA

Fig. 2.11. Imagen real procesada en software y FPGA para detección de bordes.

Los valores de paso de tiempo y tiempo máximo de procesamiento fueron los mismos en ambos casos. Las plantillas utilizadas fueron las encontradas con el método propuesto.

Los recursos utilizados por la implementación de la CeNN digital, además, de una célula digital individual, se muestran en la Tabla 2.3. La tarjeta de desarrollo utilizada es la Nexys 4 DDR, la cual contiene un FPGA Artix-7 XC7A100T-CSG324.

Tabla. 2.3. Recursos utilizados por la CeNN digital implementada en FPGA

	LUTs (63400)	**Registers** (126800)	**Slice** (15850)	**BRAM** (135)	**DSPs** (240)
CeNN	17601	8748	5660	32	64
▷ Cell	215	114	111	0	1

2.6. Conclusiones

Como se ha podido observar en este capítulo, el procesamiento de imágenes, utilizando las CeNNs es atractiva; debido a que las diferentes tareas se pueden lograr con solo cambiar los valores de las plantillas.

Como se pudo comprobar en el cálculo de las plantillas para determinadas tareas, son fácilmente encontradas utilizando las nuevas propuestas de optimización heurísticas. Dichas propuestas de optimización son atractivas ya que no dependen de la complejidad del problema a tratar. Además, con la técnica de búsqueda de plantillas propuesta en esta tesis se pueden encontrar las plantillas para diversas tareas sobre imágenes. Esta propuesta, como se mostró en la sección de resultados de búsqueda de plantillas y en específico en la Tabla 2.1 y Tabla 2.2, entrega mejores resultados comparados con utilizar únicamente ABC y NM por separado.

Dada la comprobación del funcionamiento en software y hardware de la CeNN, es posible realizar diferentes tipos de procesamiento sobre imágenes, lo cual se logra únicamente cambiando los valores de plantillas. El saber esto, tiene una gran ventaja en hardware ya que no es necesario cambiar la topología de la red neuronal, sino

únicamente los valores de las plantillas, lo cual evita el estar reconfigurando la CeNN en el dispositivo programable.

Los valores entregados por las CeNNs, es decir, en valores binarios, es funcional desde el punto de vista de hardware, ya que con esto se reduce la cantidad de recursos que se requieren sobre los dispositivos programables, al ser utilizado únicamente 1 bit y no 8 o 16 bits para representar un píxel. Es una gran ventaja tener esta reducción de recursos ya que se reducen los valores a ser cifrados. Otra ventaja de tener esta reducción de recursos es el poder implementar el sistema criptográfico en el dispositivo programable.

Capítulo 3

Criptografía y SCA sobre Dispositivos Programables

3.1. Sistemas criptográficos RSA y ECC

Hoy en día, el uso de dispositivos inteligentes es cada vez más empleado en la vida cotidiana de la población, inclusive se emplean cada vez más en la industria y el gobierno. Como ejemplo de esto lo podemos ver en las tarjetas inteligentes (*Smart Cards*), las cuales son ampliamente utilizadas para el acceso al transporte público o inclusive como acceso a las instalaciones de empresas y/o de gobierno. Un segundo ejemplo, es el uso de lectores biométricos, éstos son utilizados como controles de acceso, inclusive a países. Finalmente, un tercer ejemplo, es el uso de tarjetas bancarias, las cuales contienen información sensible sobre las cuentas de los propietarios. Como vemos, el uso de este tipo de dispositivos va en aumento.

Sin embargo, el uso de estos dispositivos conlleva riesgos como la pérdida o robo de información. Es por ello por lo que se han propuesto diversas metodologías que proporcionen protección de información contenida en los diversos sistemas electrónicos.

Como se ha mencionado anteriormente, existen diversos sistemas criptográficos

para la protección de datos en los diferentes dispositivos electrónicos. De los diferentes sistemas criptográficos podemos encontrar el RSA [36], ECC [37,38] y AES [39], entre otros.

Los sistemas RSA y ECC son atractivos ya que pertenecen a la *criptografía de clave pública*. Como fue mencionado anteriormente, la criptografía de clave pública no requiere de enviar la(s) clave(s) por medios no seguros.

3.1.1. Sistema Criptográfico RSA

La criptografía RSA fue desarrollada en el año de 1977 por Rivest, Shamir y Adleman [36]. A pesar del tiempo, este sistema sigue siendo uno de los más utilizados. La seguridad de este sistema radica en el hecho de que es complicado calcular la factorización de números enteros grandes. Actualmente se trabaja en llaves (o claves) que van desde los 1024 bits hasta los 4096 bits.

El algoritmo de cifrado funciona de la siguiente manera [2,3]: Supóngase que Bob requiere enviarle un mensaje a Alicia, entonces Alicia le envía a Bob una clave para que Bob cifre la información, esta clave es pública, ya que ésta puede ser leída por cualquier ente. Cuando Alicia reciba el mensaje cifrado por parte de Bob, ella puede descifrar la información utilizando su clave privada. Esta clave es privada debido a que solamente ella conoce dicha clave. Nótese que con la clave pública no puede descifrar la información cifrada previamente.

Para cifrar la información en este sistema se utiliza la exponenciación modular, la cual es mostrada en (3.1):

$$c = m^e \mod N \tag{3.1}$$

Donde c es el mensaje cifrado, m es el mensaje o texto en claro, e es la clave pública y finalmente N es el módulo.

Para descifrar la información, igualmente se utiliza la exponenciación modular, sin

embargo, cambian los parámetros, esto es mostrado en (3.2).

$$m = c^d \mod N \tag{3.2}$$

Donde m es el mensaje o texto en claro, c es el mensaje cifrado, d es la clave privada y finalmente N es el módulo. Como se ha podido observar, el valor del módulo N es público.

3.1.2. Sistema Criptográfico ECC

Las curvas elípticas han sido utilizadas en criptografía en los últimos años. Victor Miller y Neal Koblitz propusieron, en forma independiente, el uso de curvas elípticas en criptografía [37, 38], llegando así a la criptografía con curvas elípticas (ECC de *Elliptic Curve Cryptography*). A diferencia de RSA, la seguridad de ECC radica en el llamado *Problema del Logaritmo Discreto* (DLP de *Discrete Logarithm Problem*).

Con ECC se pueden tener longitudes de claves menores a las de RSA, en general, se dice que con una clave de 160 bits se tiene la misma seguridad que una clave de 1024 bits en RSA [40].

Existen diversos métodos con los cuales se puede cifrar información utilizando las curvas elípticas, algunos de ellos los podemos encontrar en [2,3,40,41]. Sin embargo, la operación principal utilizada es la multiplicación de un escalar por un punto en la curva, como es mostrado en (3.3).

$$c = kP \tag{3.3}$$

Para el descifrado, igualmente se utiliza como base la multiplicación de puntos de una curva elíptica, como fue mostrado en (3.3).

Donde el mensaje a cifrar m se representa como un punto dentro de la curva elíptica, como se verá más adelante, existen diversas formas de realizar esto. El valor k es

un valor entero aleatorio escogido y P es el punto dentro de la curva elíptica.

3.2. Ataques de Canal Lateral

Como se mencionó anteriormente, las operaciones principales para los sistemas criptográficos son la exponenciación modular y la multiplicación escalar en los sistemas RSA y ECC, respectivamente. En lo que respecta a las operaciones realizadas en los sistemas anteriores, estas son realizadas con la representación binaria de las claves pública o privada [2]. Es así como los SCA toman ventaja de este hecho.

3.2.1. Ataques de canal lateral sobre la Exponenciación Modular

La exponenciación modular es un tipo de exponenciación que se ejecuta sobre un módulo. Para el caso de la criptografía, el exponente puede ser la clave pública o la privada. Antes de que Kocher publicara su trabajo [15], los algoritmos más utilizados para la exponenciación modular eran los llamados Square and Multiply en sus dos versiones, Left to Right y Right to Left, SaML2R y SaMR2L respectivamente. En el Algoritmo 3.1 se puede observar el algoritmo SaML2R.

Algoritmo 3.1 Algoritmo *Square and Multiply, Lef to to Right*

Entrada: m, d, N con $d = (d_{n-1}, \cdots d_0)_2$
Salida: $S = m^d \mod N$

1: $R[0] \leftarrow 1$
2: **para** i desde $n-1$ hasta 0 **hacer**
3: $\quad R[0] \leftarrow R[0]^2 \mod N$
4: $\quad$ **si** $d_i == 1$ **entonces**
5: $\quad\quad R[0] \leftarrow R[0] \cdot m \mod N$
6: $\quad$ **fin si**
7: **fin para**
8: **devolver** $(R[0])$

Tanto el algoritmo SaML2R como el SaMR2L utilizan al exponente en su representación binaria para realizar las operaciones necesarias para el cálculo de la exponenciación. De esta forma, los algoritmos han sido atacados con los SCAs en los

dispositivos criptográficos para obtener la representación de los exponentes, es decir, las claves pública o privada.

Para evitar los SCAs en los dispositivos criptográficos, se han propuesto diversas mejoras en los algoritmos para su protección y así brindar una mejor protección en los sistemas que utilicen la exponenciación modular como operación matemática principal, como puede ser RSA, DH, ElGamal y ECC. Para el caso de ECC, no se utiliza la exponenciación modular como operación principal, sino que se utiliza, como se mencionó anteriormente, la multiplicación escalar. Ambas operaciones, desde el punto de vista algorítmico, realizan operaciones análogas [42].

3.2.2. Ataque N-1

De igual forma, se han propuesto diversos algoritmos para evitar los SCAs. Es así como, se han propuesto diversos métodos para el ataque a los dispositivos que contengan los algoritmos con protección. Un ataque pasivo interesante, es el llamado ataque por *mensaje elegido* (*chosen-message attack*). El ataque por mensaje elegido utiliza un mensaje de entrada particular que tiene un comportamiento específico durante la ejecución del algoritmo criptográfico. Algunos ataques de éstos son: el ataque desarrollado e implementado por *Fouque* y *Valette*, el cual es llamado *Ataque Doble* (DA de *Double Attack*) [43]; y el ataque llamado *Ataque Doble Relativo* desarrollado por *Yen et al.*, [44].

El ataque $N-1$, el cual fue propuesto por Yen *et al.*, [45] e implementado por Miyamoto *et al.*, [46], es un tipo de ataque por mensaje elegido. El ataque $N-1$ funciona tomando en cuenta las dos siguientes observaciones:

1. La primera observación es que $(N-1)^2 \equiv 1 \mod N$, esto puede ser extendido al hecho de que $(N-1)^j \equiv 1 \mod N$ para j como número par.
2. La segunda observación es similar a la anterior, $(N-1)^k \equiv (N-1) \mod N$ para k como número impar.

Las dos observaciones antes mencionadas hacen que, durante la exponenciación

modular, con el exponente representado en binario, se obtengan dos valores: 1 cuando el bit procesado del exponente es 0 y $N-1$ cuando el bit procesado es 1. De esta forma, el atacante puede observar dos patrones en las mediciones realizadas sobre el dispositivo y de esta forma obtener la clave.

3.2.3. Contramedidas a los SCA

Como se ha mencionado anteriormente, existen diversas propuestas para evitar los llamados SCAs, por ejemplo, para evitar los ataques por medición de tiempo, fue propuesto el algoritmo Square and Multiply Always (*SaMA*). Este algoritmo fue propuesto por *Coron* [47] y es un algoritmo regular, es decir, siempre realiza la misma cantidad de cálculos independientemente del valor del exponente o del mensaje a cifrar.

El algoritmo *SaMA* es mostrado en el Algoritmo 3.2 en su versión Left to Right (*SaMAL2R*).

Algoritmo 3.2 Algoritmo *Square and Multiply Always, Lef to to Right*

Entrada: m, d, N con $d = (d_{n-1}, \cdots d_0)_2$
Salida: $S = m^d \mod N$

1: $R[0] \leftarrow 1$
2: **para** i desde $n-1$ hasta 0 **hacer**
3: $\quad S[0] \leftarrow S^2 \mod N$
4: $\quad S[1] \leftarrow S[0] \cdot m \mod N$
5: $\quad S \leftarrow S[d_i] \mod N$
6: **fin para**
7: **devolver** (S)

El algoritmo *SaMAL2dR* ha sido vulnerado con los ataques $N-1$. Además de la propuesta del Algoritmo 3.2, se han sugerido otros métodos que evitan los ataques $N-1$, por ejemplo, se ha propuesto proteger al algoritmo blindando el mensaje con un valor aleatorio. Otro ejemplo, es comprobar que el mensaje de entrada sea diferente de $N-1$, sin embargo, estas propuestas han sido vulneradas con los ataques por *mensaje elegido* [44, 45].

Un algoritmo interesante contra los ataques $N-1$, es el propuesto en [48]. Ésta

está basada en el Algoritmo 3.2, tiene la ventaja de no requerir de blindar el mensaje de entrada con un valor aleatorio, además de que no se compara el mensaje con el valor $N-1$.

El método propuesto se muestra Algoritmo 3.3, éste es nombrado *MSaMAL2R* (de *Modified Square and Multiply Always, Left to Right*).

El Algoritmo 3.3 tiene la ventaja de que, al realizar las mediciones de potencia durante los SCAs, no se revelan dos patrones en las mediciones, sino únicamente un patrón, evitando así determinar el valor del exponente, por lo tanto, no se revela la clave pública o privada.

Algoritmo 3.3 Algoritmo modificado de *SaMAL2R*

Entrada: m, d, N, con $d = (d_{n-1}, \cdots, d_0)_2$
Salida: $S = m^d \mod N$

1: $S \leftarrow 1$
2: $M \leftarrow m \cdot m \mod N$
3: **para** i desde $(n-1)$ hasta 1 **hacer**
4: $\quad R[0] \leftarrow S^2 \mod N$
5: $\quad R[1] \leftarrow R[0] \cdot M \mod N$
6: $\quad S \leftarrow R[d_i] = \mod N$
7: **fin para**
8: **si** $d_0 = 0$ **entonces**
9: $\quad$ **devolver** 0
10: **si no**
11: $\quad$ **devolver** $S \cdot m \mod N$
12: **fin si**

3.3. Multiplicación Modular sobre dispositivos programables

Como fue mencionado anteriormente, la operación matemática principal de algunos sistemas criptográficos, como RSA, es la exponenciación modular. Sin embargo, como se puede observar tanto en el Algoritmo 3.2 como en el Algoritmo 3.3, la operación que se realiza mayoritariamente es la multiplicación modular.

Dado que la multiplicación modular es la operación matemática que se realiza ma-

yoritariamente durante la exponenciación modular, es importante que se implemente primero ésta en los dispositivos programables.

En la literatura existen diversas propuestas de algoritmos para realizar la multiplicación modular. El realizar una multiplicación modular desde el punto de vista tradicional, es complicado, debido a que se requiere una división o una reducción modular. Estas operaciones, aunque desde el punto de vista de software no representarían un gran costo computacional, desde el punto de vista de hardware éstos si representan un gran consumo de recursos y consumen mucho tiempo en su procesamiento.

Debido a lo anterior, se han propuesto diversos métodos para realizar la implementación de la multiplicación modular. Entre otros trabajos que se han publicado, se pueden encontrar los realizados por *Brickell* [49], *Barret* [50] y *Montgomery* [51]. Aunque estas propuestas fueron realizadas hace tiempo, aún siguen siendo la base para nuevas publicaciones con mejoras tanto a nivel algorítmico como a nivel de implementación. Todas estas propuestas tratan fundamentalmente el tema de no utilizar la división, lo cual las hace atractivas desde el punto de vista de la implementación en hardware.

3.3.1. Multiplicación Modular Montgomery

Para realizar la multiplicación en el contexto de criptografía, en especial de RSA y ECC, se requiere de realizar una multiplicación y una división de dos números enteros grandes, del orden que van desde 1024 bits hasta los 4096 bits para el caso de RSA y de 160 hasta los 500 bits para el caso de ECC.

Desde el punto de vista tanto del software como del hardware, vale la pena hacer notar que esta multiplicación y división consumen una gran cantidad de recursos y tiempo de ejecución. Como una solución a esto, se tiene la multiplicación Montgomery que es un algoritmo que realiza la multiplicación modular $x \cdot y \mod N$ sin necesidad de realizar la división por N.

El algoritmo original de Montgomery es presentado en Algoritmo 3.4.

Algoritmo 3.4 Multiplicación Montgomery

Entrada: $N = (N_{n-1}, \cdots, N_0)_b$, $x = (x_{n-1}, \cdots, x_0)_b$, $y = (y_{n-1}, \cdots, y_0)_b$, con $0 \leq x, y \leq N$, $R = b^n$ y con $\gcd(N, b) = 1$ y $N' = -N^{-1} \mod b$

Salida: $A = xyR^{-1} \mod N$

1: $A \leftarrow 0$ ▷ con $A = (A_n, \cdots, A_0)_b$
2: **para** i desde 0 hasta $(n-1)$ **hacer**
3: $u_i \leftarrow (A_0 + x_i y_0) N' \mod b$
4: $A \leftarrow (A + x_i y + u_i N)/b$
5: **fin para**
6: **si** $A \geq N$ **entonces**
7: $A \leftarrow A - N$
8: **fin si**
9: **devolver** A

Inicialmente, los valores ingresados en el Algoritmo 3.4, esto es x e y, deben ser ingresados en el dominio de Montgomery, lo cual puede ser realizado utilizando el mismo Algoritmo 3.4. Para el caso de x los valores de entrada son $\widetilde{x}$ y R^2, y para el caso de y los valores de entrada son $\widetilde{y}$ y R^2. Donde los valores $\widetilde{x}$ e $\widetilde{y}$ son los valores originales.

En el Algoritmo 3.4, los operandos x e y son representados en una base (o radix) b. El resultado es colocado en A y después de n iteraciones el resultado es igual a xyR^{-1} mód N, con lo cual, el resultado deber ser regresado al valor correcto $\widetilde{xy} \mod N$.

Para obtener el valor correcto de $\widetilde{A}$ se requiere realizar una multiplicación Montgomery adicional con los valores de A y 1 como entradas.

En un principio, realizar estas operaciones adicionales podrían parecer algo costoso, sin embargo, desde el punto de vista de una exponenciación modular, estos son insignificantes, debido a que se realiza una única vez.

En el Algoritmo 3.4 se realiza una comprobación final, para ser específicos en el paso 6. Esta comprobación podría resultar un punto débil ya que se puede utilizar para realizar SCAs; por otra parte, desde el punto de vista de hardware, esta comprobación resulta costosa en términos de recursos. Para evitar lo antes mencionado, se han propuesto diversos algoritmos que evitan esta última comprobación, entre otras propuestas se pueden encontrar las mencionadas en [52, 53].

La propuesta realizada por Walter [53], es interesante porque evita realizar la comprobación y no solo eso sino también evita la resta requerida, esto se logra haciendo los valores x e y menores a $2N$ y además $2N$ debe ser menor a b^n. Esta propuesta es mostrada en el Algoritmo 3.5.

Algoritmo 3.5 Multiplicación Montgomery sin resta final

Entrada: $N = (N_{n-1}, \cdots, N_0)_b$, $x = (x_{n-1}, \cdots, x_0)_b$, $y = (y_{n-1}, \cdots, y_0)_b$, con $0 \leq x, y \leq 2N$, $2N \leq R = b^n$ y con $\gcd(N, b) = 1$ y $N' = -N^{-1} \mod b$
Salida: $A = xyR^{-1} \mod N$

1: $A \leftarrow 0$ ▷ con $A = (A_{n-1}, \cdots, A_0)_b$
2: **para** i desde 0 hasta $(n-1)$ **hacer**
3: $\quad u_i \leftarrow (A_0 + x_i y_0) N' \mod b$
4: $\quad A \leftarrow (A + x_i y + u_i N)/b$
5: **fin para**
6: **devolver** A

3.3.2. Implementación de la Multiplicación Montgomery en FPGA

Para poder implementar el Algoritmo 3.5 en un FPGA, éste debe ser primero adaptado a términos de hardware. En el Algoritmo 3.6 se puede observar la adaptación realizada para que pueda ser implementado.

Algoritmo 3.6 Multiplicación Montgomery para FPGA

Entrada: $N = (N_{n-1}, \cdots, N_0)_b$, $x = (x_{n-1}, \cdots, x_0)_b$, $y = (y_{n-1}, \cdots, y_0)_b$, con $0 \leq x, y \leq 2N$, $2N \leq R = b^n$ y con $\gcd(N, b) = 1$ y $N' = -N^{-1} \mod b$
Salida: $A = xyR^{-1} \mod N$

1: $A \leftarrow 0$
2: **para** i desde 0 hasta $(n-1)$ **hacer**
3: $\quad$ Tomar los k bits menos significativos de $u_i \leftarrow (A_0 + x_i y_0) N'$
4: $\quad$ **para** j desde 0 hasta $n-1$ **hacer**
5: $\quad\quad (c_j, A_j) \leftarrow (A_j + x_i y + u_i N + c_j) \gg k$
6: $\quad$ **fin para**
7: **fin para**
8: **devolver** A

El Algoritmo 3.6 utiliza una técnica llamada Sistólica. La técnica sistólica toma como ventaja el hecho de que los números son representados en una base b y así

conformar bloques de procesamiento con dicho tamaño de base; con dichos bloques se construye un arreglo lineal de bloques al tamaño de los operandos.

Para el caso aquí estudiado, se toma como ventaja que se puede representar la base b como potencia de 2, es decir, $b = 2^k$, donde k es el número de bits. Teniendo en cuenta este hecho, se logra simplificar la implementación en un FPGA.

Aunque está técnica sistólica ha sido implementada en [54, 55], en este trabajo, se presenta una variante a los trabajos antes publicados. Con la técnica que se propone, como se verá más adelante, se incrementa la velocidad de operación y se disminuye la cantidad de recursos requeridos del FPGA.

La estructura básica del sistema sistólico es llamado *Elemento de Procesamiento* (EP). Estos EPs son replicados hasta obtener la cantidad necesaria para formar un arreglo lineal. En la Fig. 3.1 se muestra el diagrama a bloques de la estructura sistólica implementada en FPGA.

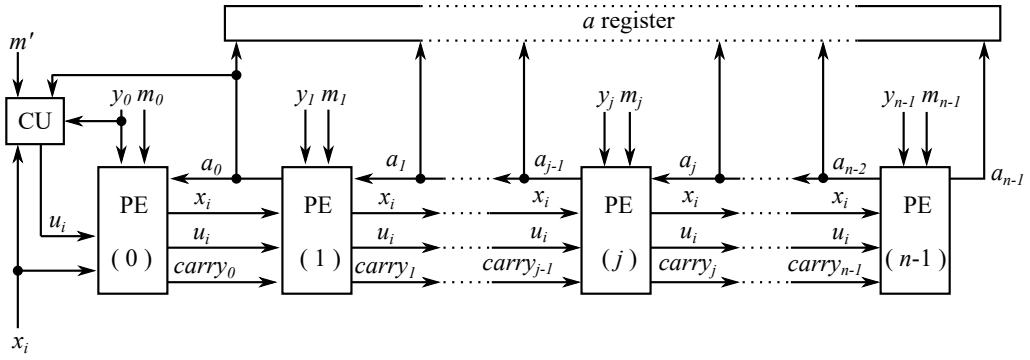


Fig. 3.1. Diagrama a bloques de la arquitectura sistólica

La cantidad total de EPs, es decir n, dependerá del tamaño de los valores x e y. Para un tamaño de los operandos de 1024 bits y una base b de 16 bits ($k = 16$), se tendrían 64 palabras de 16 bits o $n = 64$.

El arreglo unidimensional formado por los EPs realiza la operación del paso 5 del Algoritmo 3.6, esto es, la operación $(c_j, A_j) \leftarrow (A_j + x_i y + u_i N + c_j) \gg k$, donde A ha sido representado con la misma base que el resto de los operandos. El valor c_j representa el acarreo obtenido durante la operación de cada EP.

Los operandos son representados en una base b de k bits. Así, la división en el

paso 4 del Algoritmo 3.5 es realizada por un corrimiento de k bits en el paso 5 del Algoritmo 3.6.

La operación $u_i \leftarrow (A_0 + x_i y_0)N'$ del Algoritmo 3.6, es ejecutada en un bloque separado llamado **CU** (Calculo de u_i).

El control de los bucles es realizado con una máquina de estados finitos, FSM. Donde estas FSMs proveen los valores correspondientes a cada uno de los EPs y u_i. El control general de la Multiplicación Montgomery es realizado según el Algoritmo 3.7.

Algoritmo 3.7 Control de la Multiplicación Montgomery para FPGA

1: Esperar hasta que se habilita la Multiplicación Montgomery. $i = 0$.
2: Habilita la operación de los EPs.
3: $i = i + 1$
4: **si** $i \geq n - 1$ **entonces**
5: Ir al estado 9.
6: **si no**
7: Ir al estado 2.
8: **fin si**
9: **devolver** el resultado.

En la Fig. 3.2 se muestra el diagrama de flujo de la máquina de estados implementada.

El control funciona de la siguiente manera:

Estado 1: La variable i es inicializada a 0 y mientras tanto se espera la activación. Al mismo tiempo, y y m son separadas en n palabras con una base b y son enviadas al correspondiente EP.

Estado 2: El primer EP es activado y al mismo tiempo este EP activa al segundo EP, y así sucesivamente hasta el último EP.

Estado 3: Aquí, i incrementa su cuenta en 1, con la ayuda de un contador. En este estado, la siguiente x_i es leído.

Estado 4: En este estado, i es comparado con el valor $n - 1$ y si este es igual o mayor salta al estado 5, de lo contrario salta al estado 2.

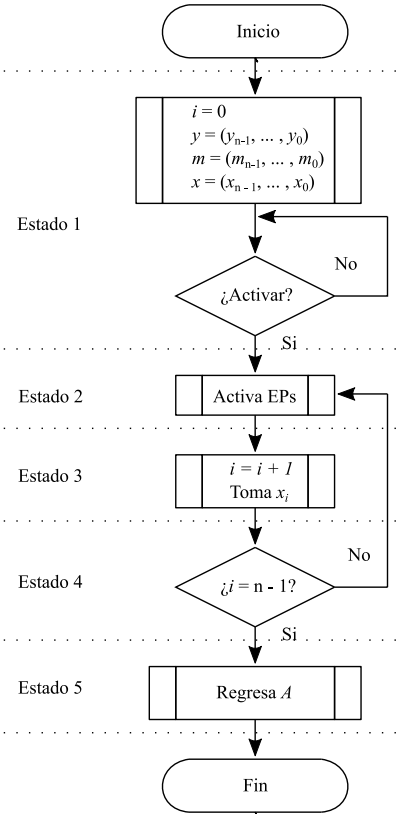


Fig. 3.2. Diagrama de flujo para la multiplicación modular.

Estado 5: Este estado termina el ciclo y regresa el resultado de la multiplicación.

Mientras la máquina de estados del control de la multiplicación está realizando su labor, el bloque CU realiza el cálculo de u_i al mismo tiempo.

Como se puede observar de la Fig. 3.1, el primer EP y el último EP son distintos a los demás. Es así como se presentan tres diferentes tipos de EP, los cuales se explican a continuación:

El **primer EP**, el cual se identificará como EPI (Elemento de Procesamiento Inicial), tiene comunicación directa con el control de la multiplicación y es el que recibe la señal de habilitación, además de que recibe los valores x_i y u_i. El diagrama a bloques de este EPI es mostrado en Fig. 3.3.

El funcionamiento de este primer EP está controlado por una máquina de estados, el cual consiste en tres estados, que se explican a continuación:

Durante el *primer estado*, x_i y u_i son guardados en registros, y al mismo tiempo

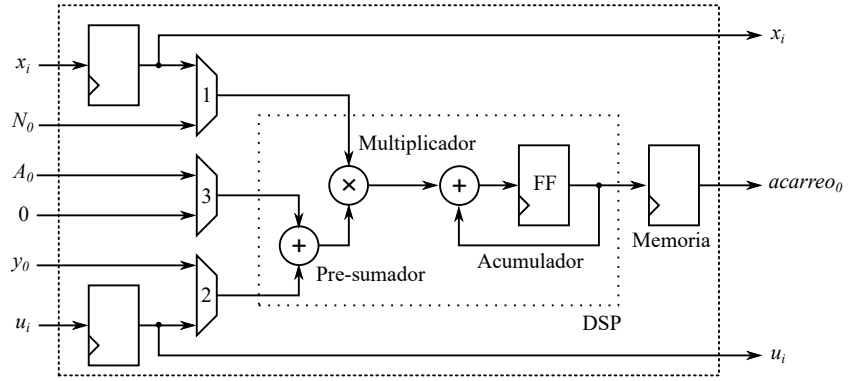


Fig. 3.3. Diagrama a bloques del elemento de procesamiento inicial (EPI)

son enviados al segundo EP. Aun estando en el primer estado, los valores de N_0 y u_i son enviados al multiplicador a través de los multiplexores 1 y 2 respectivamente. Además, un valor de 0 es enviado al presumador dentro del *Procesador Digital de Señales* (DSP de *Digital Signal Processor*) a través del multiplexor 3. El resultado es guardado en el acumulador del DSP.

Durante el *segundo estado*, la multiplicación entre x_i y y_0 se lleva a cabo, al mismo tiempo se realiza la suma con el valor de A_0. El resultado es sumado al valor previo guardándose en el acumulador del DSP.

Finalmente, en el *tercer estado*, el resultado de las multiplicaciones con los valores de suma es guardado en un bloque llamado *Memory*. El resultado final tiene una longitud de $2k+2$ bits. Sin embargo, los primeros k bits son descartados, esto significa realizar la división por b en $A \leftarrow (A + x_i y + u_i N)/b$, el resto de los bits son guardados y enviados al siguiente EP. Después de esto, los valores en el acumulador son reiniciados a cero.

En la Fig. 3.4, se puede observar el **segundo EP**, el cual se identificará como EPG (Elemento de Procesamiento General).

El EPG, al contrario del EPI, recibe como entrada un valor de acarreo y tiene como salidaA_{j-1}. El primer EPG recibe de entrada el acarreo del EPI y este primer EPG envía su acarreo al segundo EPG, esto se realiza hasta el último EPG.

El comportamiento de los EPG es similar al EPI, sin embargo, en el primer estado se realiza una suma adicional del valor de acarreo. El resultado, al igual que en el EPI,

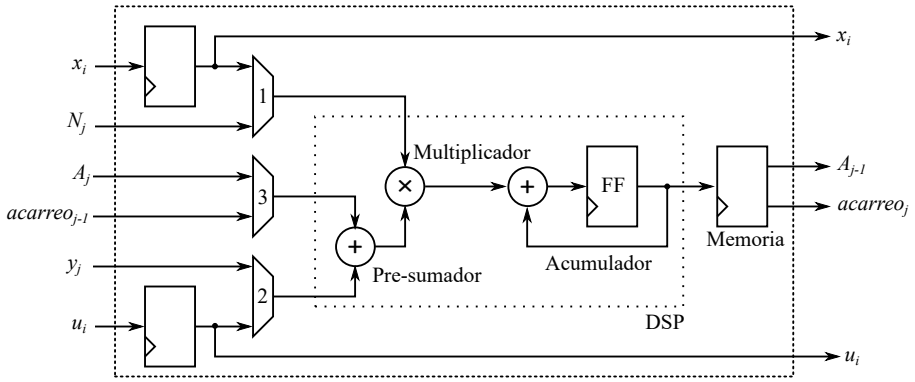


Fig. 3.4. Diagrama a bloques del elemento de procesamiento general (GPE)

tiene una longitud de $2k+2$ bits. Los primeros k bits son equivalentes al valor A_{j-1}, el resto de los bits son enviados como acarreos.

La salida A_{j-1} de cada EPG es desplazada a la izquierda para complementar la división por b, así como fue mencionado en el funcionamiento del EPI.

El **tercer y último EP** es llamado EPF (Elemento de Procesamiento Final). Este EPF es mostrado en la Fig. 3.5.

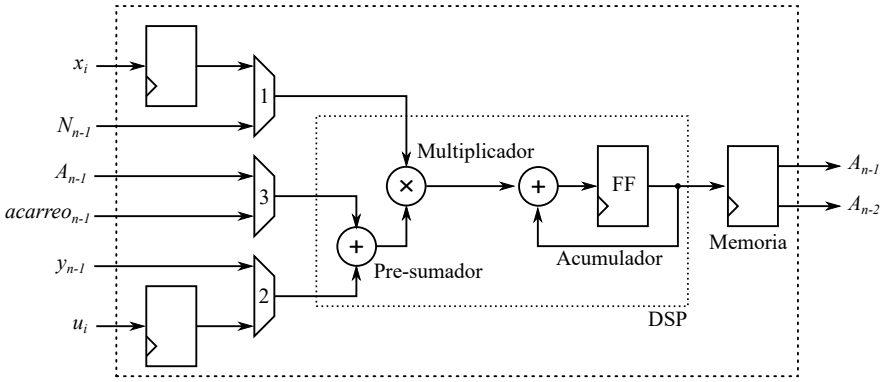


Fig. 3.5. Diagrama a bloques del elemento de procesamiento final (EPF)

El funcionamiento del EPF es similar al EPG, sin embargo, el acarreo de salida es evitado. En este EP, la salida tiene una longitud de $2k$ bits. Los primeros k bits de la salida son llamados A_{j-2} y los últimos k bits son llamados A_{n-1}.

Por último, el **bloque CU** calcula el valor u_i. Como fue mencionado anteriormente, tiene una base en potencia de 2. Así la operación mód b necesita únicamente los k bits menos significativos. En la Fig. 3.6 se muestra el diagrama a bloques de éste.

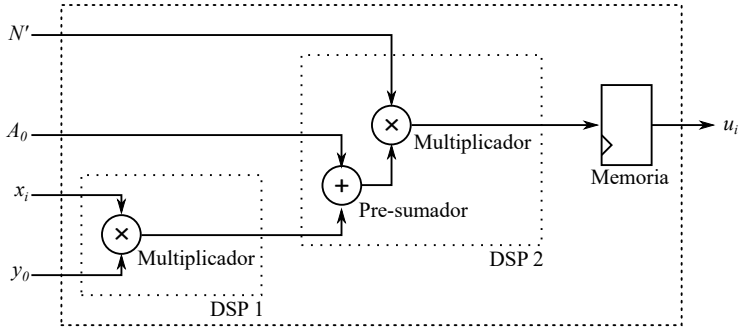


Fig. 3.6. Diagrama a bloques del calculo de u_i (CU)

Para poder reducir el consumo de recursos dentro del FPGA, se han utilizado dos DSPs, como fue mostrado en la Fig. 3.6.

3.4. Propuesta de mejora a la implementación de la Exponenciación Modular en FPGA

3.4.1. Algoritmo de exponenciación modular para FPGA

Una vez implementada la multiplicación modular, a continuación, se procedió a implementar la exponenciación modular. Se tomó como base el Algoritmo 3.3 y tomando en cuenta la implementación de la Multiplicación Modular Montgomery, se procedió a realizar las modificaciones necesarias para implementar la exponenciación modular en el FPGA.

Para realizar correctamente la Multiplicación Montgomery, los operandos deben ser convertidos al dominio de Montgomery, además de que esto no era un gran problema en la exponenciación modular. Tomando en cuenta lo antes mencionado, se procedió a modificar el Algoritmo 3.3 para poder ser implementado en el FPGA, dicha modificación se puede observar en el Algoritmo 3.8.

Del anterior algoritmo, se eliminó la comparación (el Si - If). La eliminación del Si se debe a que en el contexto de RSA siempre el valor del exponente es impar, es decir, el bit menos significativo es uno.

Algoritmo 3.8 Algoritmo modificado de *SaMAL2R*

Entrada: $N = (N_{n-1}, \cdots, N_0)_b$, $m = (m_{n-1}, \cdots, m_0)_b$, $d = (d_t, \cdots, d_0)_2$ con $m < 2N < R = b^n$ y con $\gcd(N, b) = 1$ y $N' = -N^{-1} \mod b$

Salida: $S = R \mod N$

1: $S \leftarrow R \mod N$
2: $M[0] \leftarrow MultMont(m, R^2, N)$
3: $M[1] \leftarrow MultMont(M[0], M[0], N)$
4: **para** i desde $(n-1)$ hasta 1 **hacer**
5: $\quad R[0] \leftarrow MultMont(S, S, N)$
6: $\quad R[1] \leftarrow MultMont(R[0], M[1], N)$
7: $\quad S \leftarrow R[d_i] \mod N$
8: **fin para**
9: $S \leftarrow MultMont(S, M[0], N)$
10: $S \leftarrow MultMont(S, 1, N)$
11: **devolver** S

En la Tabla 3.1 se muestra un ejemplo del algoritmo propuesto. Hay que recordar que este algoritmo está pensado para evitar los SCA, especialmente los ataques $N-1$. En este ejemplo se toma como exponente $d = (89)_{10} = (1100001)_2$ y como mensaje de entrada el valor $N-1$.

Como se observa en la Tabla 3.1, el resultado siempre es el mismo, es decir, S siempre obtiene el valor de $R \mod N$. Lo anterior evita observar dos patrones diferentes cuando se procesa un bit cero o uno del exponente (clave secreta), evitando así dicha vulnerabilidad.

Los valores $R \mod N$ y $R^2 \mod N$ en el algoritmo propuesto, pueden ser ingresados como entradas. Como es requerido en la Multiplicación Montgomery, los valores de entrada son trasladados al dominio de Montgomery, esto se realiza en el paso 2 del Algoritmo 3.8 y devueltos en el paso 10.

De lo anterior, los valores $R \mod N$ y $R^2 modN$ pueden ser públicos, esto debido a que únicamente dependen de la cantidad de bits del exponente y no de su valor, y del valor del módulo. Por lo cual no representa una vulnerabilidad al algoritmo implementado.

Tabla. 3.1. Ejemplo de ejecución del algoritmo propuesto

i	d_i	***Valores de R***	***Resultado***
6	1	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[1] = R \mod N$
5	1	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[1] = R \mod N$
4	0	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[0] = R \mod N$
3	0	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[0] = R \mod N$
2	0	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[0] = R \mod N$
1	0	$R[0] = R^2 \mod N = R \mod N$ $R[1] = R \cdot R \mod N = R \mod N$	$S = R[0] = R \mod N$
		Último bit	
0	1	$S = S \cdot M[0] \mod N$	$S = M[0] \mod N$
		Recupera el resultado	
		$S = S \cdot 1 \mod N$	$S = (N-1) \mod N$

3.4.2. Resultados de la exponenciación modular

El diagrama a bloques para la implementación de la exponenciación modular propuesta es mostrado en la Fig. 3.7.

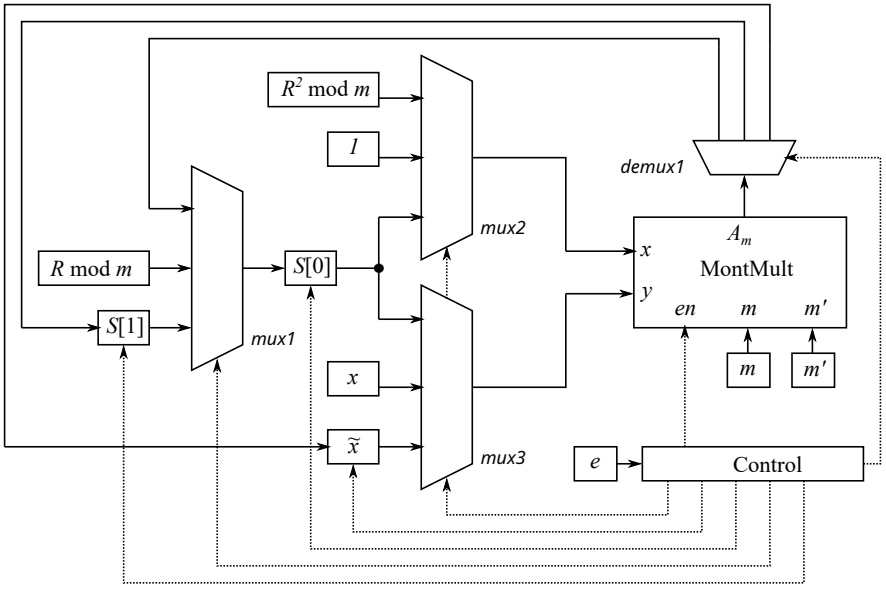


Fig. 3.7. Diagrama a bloques de la exponenciación modular para FPGA

Se ha implementado una máquina de estados finita para controlar la exponenciación modular. La exponenciación modular fue hecha a 1024 bits. Se utilizaron 9 registros en total de 1024 bits cada uno. Se utilizó un registro de 1024 bits para guardar el mensaje m, uno para guardar el valor $R \mod N$, uno para almacenar el valor de R^2 mód N, dos registros para guardar los valores de M, dos más para guardar los valores intermedios de **R** y dos para almacenar los valores de salida S. Finalmente, se utilizó un registro de 16 bits para almacenar el valor de N' y uno para guardar el valor de u_i.

Se utilizó una longitud de palabra de $k = 16$ bits. Con palabras de 16 bits, se implementaron para RSA de 1024 bits, 64 EPs, de los cuales 1 EP es para el EPI, uno para el EPF y el resto, es decir 62 para los EPGs.

En Tabla 3.2 se muestra los recursos utilizados tanto para la Multiplicación Montgomery como para la Exponenciación Modular SaML2R y la Exponenciación modular SaML2R Modificada. Como se observa en la tabla, la mayor cantidad de recursos es debido a la Multiplicación Montgomery. Es decir, del total de recursos tomada por

la exponenciación modular, gran parte de esos recursos son en realidad de la multiplicación modular, esto es para ambas implementaciones.

Tabla. 3.2. Recursos utilizados en la Multiplicación Montgomery y la Exponenciación Modular

	LUTs (63400)	**Registers** (126800)	**Slice** (15850)	**BRAM** (135)	**DSPs** (240)
SaMAL2R	7587	9499	4253	0	66
▷ *Multiplicación*	6736	5337	3429	0	66
SaMAL2R Modificado	10502	10548	4632	0	66
▷ *Multiplicación*	10290	5338	4199	0	66

Considerando que la máquina de estados de los EP es de 3 ciclos y que n=64, más el ciclo de espera y el ciclo final, se tienen en total 164 ciclos de reloj para la multiplicación modular. Con el Algoritmo 3.8 y considerando los 194 ciclos de la multiplicación modular, se tienen en total 399,113 ciclos de reloj total. Considerando una frecuencia de trabajo del FPGA de 100Hz, es decir un periodo de 10ns por ciclo, tenemos que el tiempo total de la exponenciación modular es de 3.99113ms.

En la Tabla 3.3 se muestra una comparativa de la implementación realizada con tres propuestas similares. La primera de ellas realizada por Wangchen *et al.* [56], la cual muestra una arquitectura basada en la transformada rápida de Fourier - McLauhlin. La segunda es realizada por Vankatesh *et al.* [57], la cual realiza la exponenciación modular a través de computar los pares de base y exponente. En el tercer y último trabajo presentado por Somnath *et al.*, [58], se utiliza la arquitectura Sistólica.

La Tabla 3.3 compara diferentes arquitecturas usadas para la ejecución de la exponenciación modular. Los resultados muestran que la implementación propuesta en este trabajo tiene una reducción de LUTs (Lookup Tables), comparado con el resto de los trabajos. Además, se requiere una menor cantidad de FFs (Flip-Flops).

Una ventaja de la arquitectura presentada es que no requiere del uso de Bloques de Memoria (BRAMs), caso contrario con el resto de los trabajos, los cuales utilizan una gran cantidad.

Tabla. 3.3. Estado-del-Arte para la Exponenciación Modular con $N = 1024$ bits

Autor	FPGA	LUTs	FFs	BRAM	Slices	DSPs	f (MHz)	Ciclos
Wangchen	Virtex-6	11,834	-	$32/2$[1]	3,470	18	-	822
Vankatesh	Virtex-7	128,814	75,839	132[2]	37,134	-	200	14,813
Somnath	Artix-7	19.77 %	6.95 %	7.81 %	-	-	90.9	-
Propuesta	Artix-7	9,803	10,568	0	4,485	66	100	399,113

[1] 32 bloques de 32kb ay 2 bloques de 18kb.
[2] 132 bloques del total.
[-] Sin mencionar.

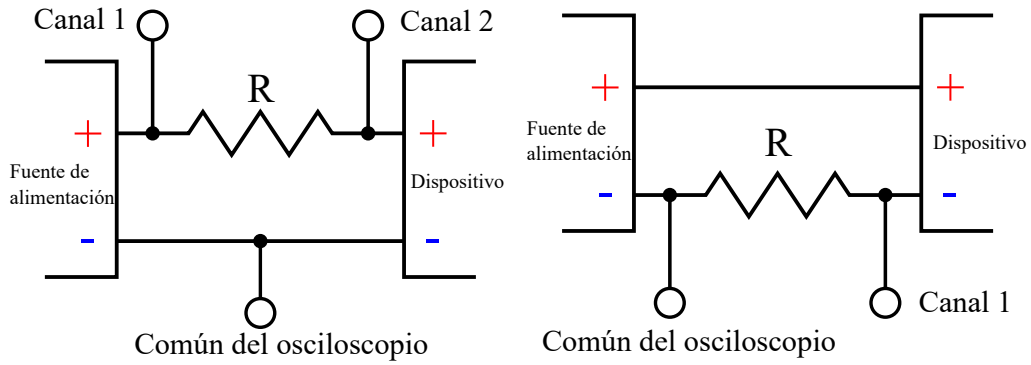
3.5. Ataques de Canal Lateral a la implementación de la Exponenciación Modular

Los ataques de canal lateral que se presentan en este trabajo son los ataques pasivos, en específico los ataques por medición de potencia. En esta sección se muestra el cómo se realizaron este tipo de ataques.

Para realizar los ataques se utilizó la implementación de la exponenciación modular expuesta en la sección anterior.

Los ataques por medición de potencia se dividen en dos categorías, el primero es llamado *Análisis de Potencia Simple* (SPA de *Simple Power Analysis*) y el segundo *Análisis de Potencia Diferencial* (DPA de *Differential Power Analysis*). **Sin embargo, el estudio se enfocó en los SPA.**

Para realizar estos dos tipos de ataques se implementó un circuito que ayudó en la medición de potencia, el cual consiste en agregar una resistencia. Dicha resistencia puede ser agregada entre la salida positiva de la fuente de alimentación y la entrada de alimentación positiva del dispositivo a atacar. Una segunda forma de agregar esta resistencia es entre la salida negativa de la fuente de alimentación y la entrada negativa del dispositivo a atacar. En Fig. 3.8 se muestran ambas configuraciones antes mencionadas.



(a) Resistencia R en el positivo

(b) Resistencia R en el negativo

Fig. 3.8. Colocación de la resistencia para realizar la medición de potencia.

3.5.1. Análisis de Potencia Simple

El primer ataque que se reporta es el SPA, y como su nombre indica, es el más simple de realizar y observar [19]. Dicho ataque consiste en realizar una medición de potencia consumida por el dispositivo criptográfico.

Como primer paso se realizó el ataque sobre el Algoritmo 3.2. Esto con la finalidad de comprobar la eficacia de realizar los ataques $N-1$ a través de la medición de potencia. En la Fig. 3.9 se observa un ataque, en dicha gráfica, se observa la caída de voltaje en el resistor serie con el dispositivo FPGA.

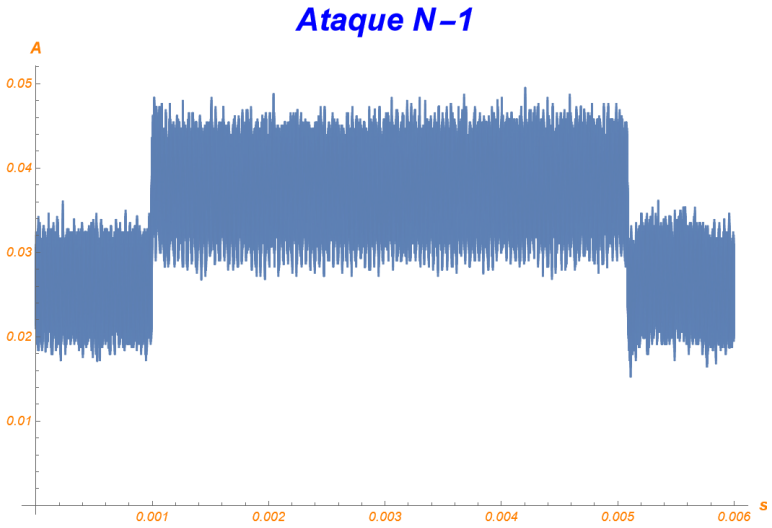


Fig. 3.9. Señal obtenida durante un ataque $N-1$ al Algoritmo 3.2.

De la Fig. 3.9, se observa claramente el momento cuando inicia la exponenciación modular y cuando finaliza. Es decir, antes de $1ms$ inicia la exponenciación y termina

en aproximadamente $5ms$; con esto se deduce que la duración de la exponenciación modular dura aproximadamente $4ms$. Este tiempo medido concuerda con el calculado en la sección anterior.

Para comprobar que efectivamente se observan los patrones que indican los cambios pertenecientes a la clave secreta, se hace un acercamiento a una señal medida. En Fig. 3.10, se observa dicho acercamiento.

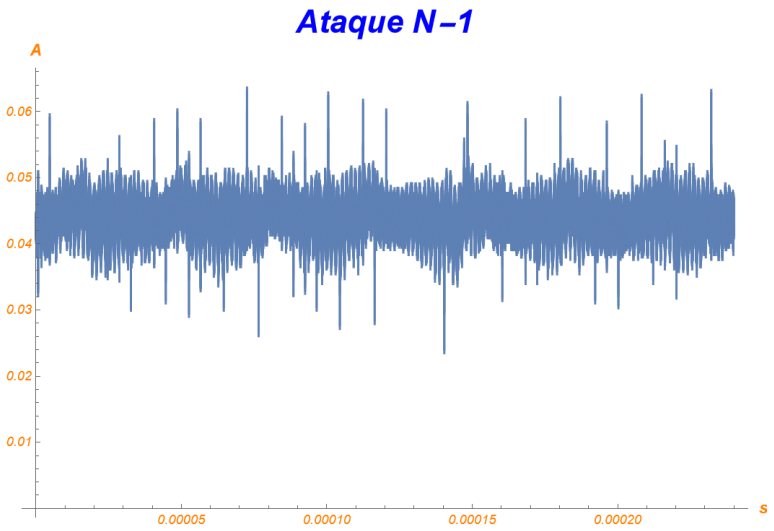


Fig. 3.10. Señal donde se observan los cambios pertenecientes al ataque $N-1$.

Para comprobar que efectivamente la señal observada corresponde a la clave secreta, se forzó una salida del FPGA que vaya mostrando el valor de dicha clave. En la Fig. 3.11 se muestra una medición que contiene tanto la corriente consumida durante el cifrado de datos como la clave mostrada.

Como se observa en las gráficas anteriores, el Algoritmo 3.2 muestra debilidad ante el ataque $N-1$. A continuación, se muestran el mismo ataque, pero ahora realizado sobre la propuesta de este trabajo, es decir, se realiza el ataque ahora sobre el Algoritmo 3.8.

En la Fig. 3.12, se muestra un ataque $N-1$ realizado al Algoritmo 3.8; como se observa en dicha figura, se logra apreciar (igual que en el caso anterior) cuando inicia y cuando termina la exponenciación modular.

Sin embargo, realizando un acercamiento a la figura. No se nota, como en el caso

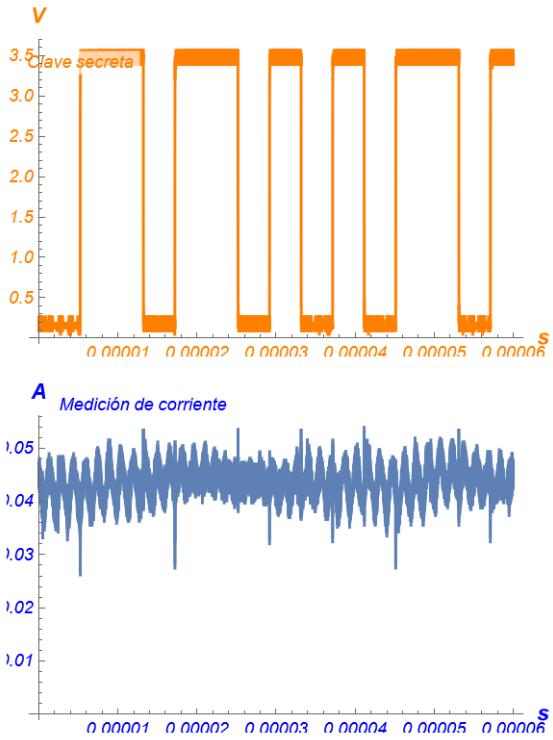


Fig. 3.11. Ataque $N-1$ a la exponenciación modular con salida del valor de la clave, utilizando el Algoritmo 3.2.

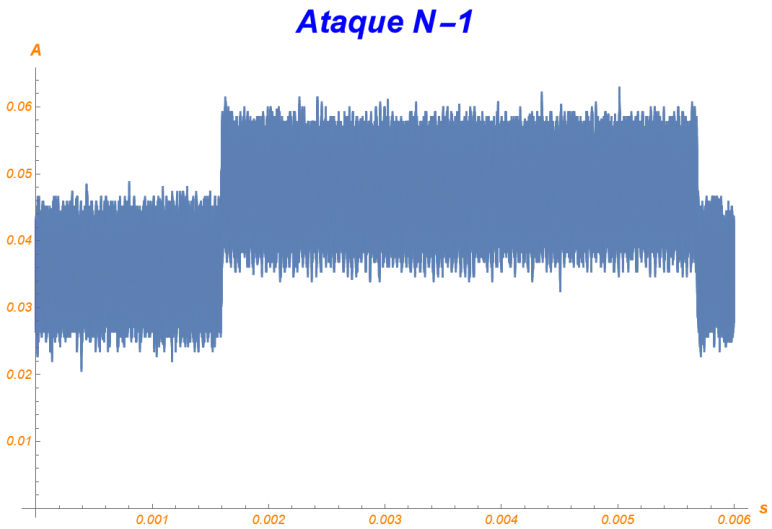


Fig. 3.12. Señal obtenida durante un ataque $N-1$ al Algoritmo 3.8.

anterior, las señales pertenecientes (o los patrones) a la clave secreta. Esto lo que se puede observar en la Fig. 3.13.

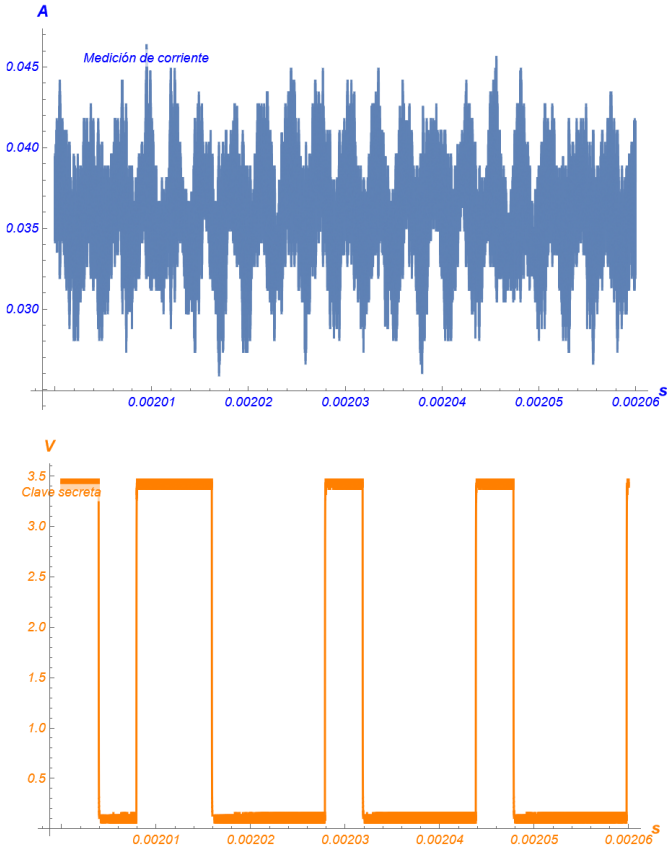


Fig. 3.13. Ataque $N-1$ a la exponenciación modular con salida del valor de la clave, utilizando el Algoritmo 3.2

Con esto se demuestra la efectividad del algoritmo propuesto contra los SCAs, específicamente los del tipo $N-1$.

3.6. Conclusiones

Como se pudo observar, se han desarrollado diversos ataques en contra de implementaciones que ejecutan diferentes algoritmos de cifrado. Además de éstos se observó que siguen surgiendo diferentes ataques a las implementaciones sobre los dispositivos programables. Es por esto, que es importante generar nuevos métodos que aseguren la protección de la información del usuario.

Se han propuesto diversas mejoras para evitar los ataques hacia los dispositivos criptográficos, sin embargo, estos generalmente aumentan la cantidad de recursos utilizados por el sistema criptográfico.

Como fue visto, durante este capítulo se propuso una mejora a la exponenciación modular, la cual es la base para diversos sistemas criptográficos; especialmente para RSA.

Empleando la propuesta realizada se comprobó que efectivamente evita los ataques $N-1$, como se mostró en las pruebas realizadas. Además de esto, la propuesta no aumenta significativamente el uso de recursos comparado con otros trabajos, como fue mostrado en la Tabla 3.3. Lo cual es importante para dispositivos con cantidad de recursos limitados.

Cabe mencionar que existen diversos ataques, sin embargo, esta tesis se enfoca sobre uno de los más importantes (el ataque $N-1$). Como parte del trabajo a futuro sería implementar otros tipos de ataques, como, por ejemplo, los Fault Attacks o los ataques basados en inteligencia artificial, para comprobar completamente la efectividad de la propuesta realizada.

Capítulo 4

Criptografía con Curva Elíptica y su implementación en FPGA

El uso de curvas elípticas en criptografía fue introducido por primera vez, de forma individual, por Miller [37] y Koblitz [38]. Estas son utilizadas junto a otros métodos para poder realizar el cifrado de datos, como lo es el uso de ElGamal, Diffie-Hellman o el uso combinado con RSA para mejorar la seguridad.

La seguridad en ECC radica en el problema del logaritmo elíptico de curvas elípticas (ECDLP de *Elliptic Curve Discrete Logarithm Problem*). ECC requiere una menor cantidad de bits para tener la misma seguridad que RSA, se dice que 256-bits de un ECC es equivalente a 3072-bits de RSA [59]. Una ventaja evidente es que esta disminución de bits produce un aumento en la velocidad de cifrado y además de requerir de una menor cantidad de recursos comparado con otros sistemas criptográficos de clave pública. Lo anterior es significativamente bueno para la implementación de este sistema criptográfico en sistemas embebidos, especialmente en aquellos con una cantidad reducida de recursos.

4.1. Introducción a Curvas Elípticas

Las curvas elípticas son curvas de género uno con un punto base específico. Tienen dos puntos definidos, el primero es el punto base y el segundo es el punto al infinito. Además, estas curvas no tienen un punto singular y no tienen intersecciones sobre ellas mismas.

Los puntos pertenecientes a la curva serán los que se utilizarán para representar la información que se usa en los protocolos criptográficos sobre curvas elípticas.

La cantidad de puntos que tiene una curva elíptica es llamada *orden*, denotado como $\#\varepsilon$. Dicho orden es importante en criptografía, sin embargo, su cálculo es difícil cuando se trabaja con curvas elípticas grandes.

La ecuación que define una curva elíptica está dada en (4.1), la cual es denominada *Forma Normal de Weierstrass* (FNW).

$$y^2 + uxy + vy = x^3 + ax^2 + bx + c \tag{4.1}$$

Con discriminante:

$$\Delta = -4a^3c + a^2b^2 + 18abc - 4b^3 - 27c^2 \tag{4.2}$$

El discriminante debe cumplir que sea diferente de cero. La condición $\Delta \neq 0$ asegura que no haya puntos que contengas dos o más rectas tangentes.

Como ejemplo, en la Fig. 4.1 se muestra una curva elíptica definida en los números racionales.

En criptografía y en especial cuando se desea utilizar en dispositivos programables, es necesario definir una curva que sea manejable por estos. Es decir, se requiere una curva definida sobre los cuerpos finitos.

En la Fig. 4.2 se muestra un ejemplo de dichas curvas, dada por $y^2 + y = x^3 + 996x$. En la Fig. 4.2 se muestra la curva sobre el cuerpo Z_{997}, es decir, es una curva $y^2 + y =$

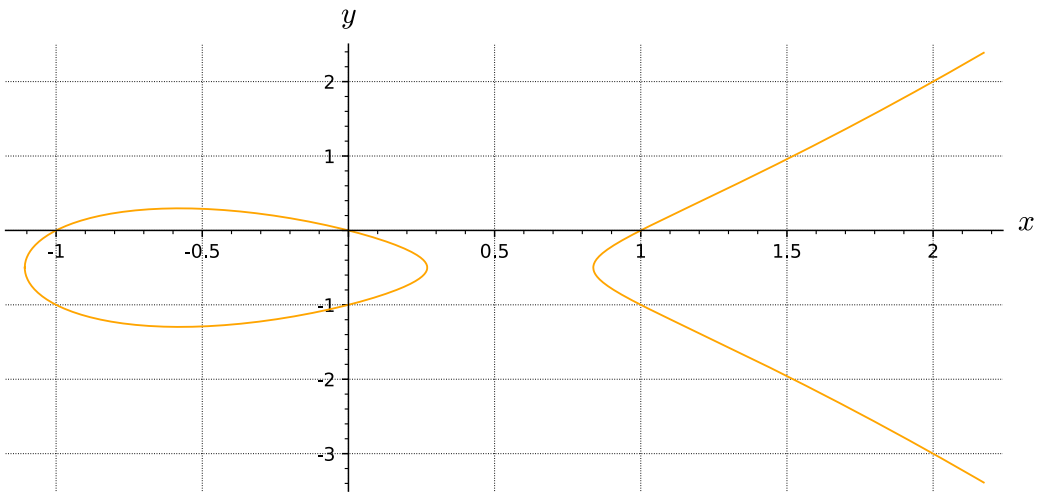


Fig. 4.1. Curva elíptica definida por $y^2 + y = x^3 - x$ sobre el campo de los racionales.

$x^3 + 996x \mod 997$.

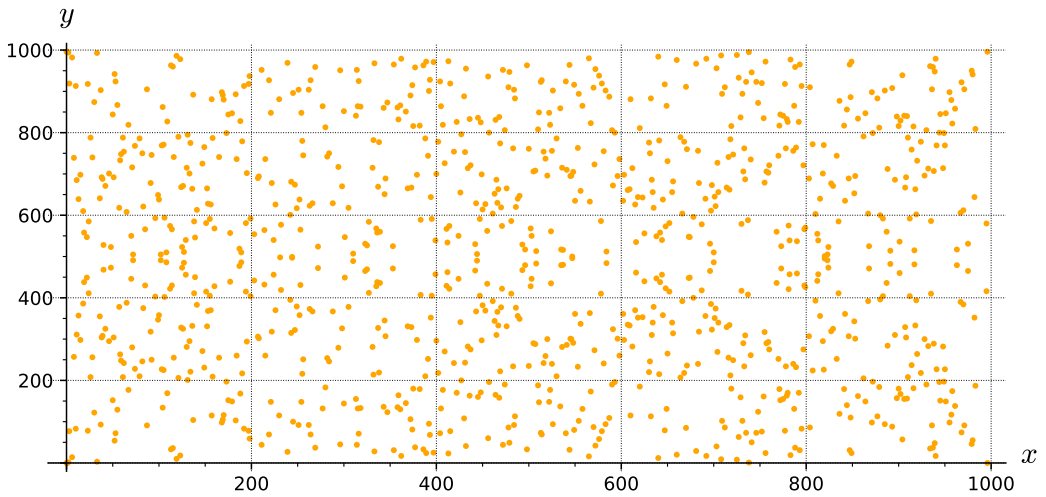


Fig. 4.2. Curva elíptica definida por $y^2 + y = x^3 + 996x$ sobre Z_{997}.

Cabe mencionar que, como ejemplo, la curva mostrada es pequeña y no es útil para criptografía, para lo cual se requiere buscar curvas que sean útiles.

Se pueden definir las curvas elípticas sobre tres campos principalmente. Primero, se definen sobre el *campo de los números reales*, sin embargo, éstos tienen el inconveniente de que sus puntos pueden llegar al infinito, lo cual no es práctico ni desde el punto de vista de software ni del hardware. El segundo, se pueden definir sobre el *campo de los números primos*; estas curvas son muy útiles desde el punto de vista

de software, ya que éstas están dentro de cuerpos finitos. Finalmente, en el tercero, las definimos sobre el *campo de los binarios*, los cuales son muy útiles para las implementaciones desde el punto de vista de hardware, ya que éstas están basadas en números que pueden ser representados en base 2, además de estar sobre cuerpos finitos [3,40,41].

Este trabajo se enfoca en las curvas elípticas definidas en el campo de los binarios. Además, se toman en cuenta las curvas elípticas recomendadas por el ***NIST*** (de *National Institute of Standards and Technology*) [60] y ***SEC 2*** (de *Securities and Exchange Commission*) [61].

4.2. Curvas Elípticas en Criptografía

Como se mencionó anteriormente, la información a ser cifrada por un sistema criptográfico utilizando curvas elípticas es representada dentro de los puntos que satisfacen la ecuación de dicha curva.

Primero, sea una curva elíptica E definida en el campo de los binarios $GF(2^m)$ cuyos puntos son las soluciones a la ecuación de Weierstrass dada en (4.3) y un punto extra ∞ (llamado punto al infinito).

$$E(F_p) : y^2 + xy = x^3 + ax^2 + b \tag{4.3}$$

Defínase la suma de dos puntos P_1 y P_2 diferentes, es decir ($P_1 \neq P_2$), que satisfacen (4.3). Se define un tercer punto, $P_1 + P_2 = (x_3, y_3)$, (el cual también se encuentra dentro de la curva) que pasa por dichos puntos como:

$$(x_3, y_3) = (\lambda^2 + \lambda + x_1 + x_2 + a, \lambda(x_1 + x_3) + x_3 + y_1) \tag{4.4}$$

Donde:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \tag{4.5}$$

Ahora, se define el doblado de un punto, como $2P = (x_3, y_3)$, con $P \neq -P$, de esta manera se tiene:

$$(x_3, y_3) = (\lambda^2 + \lambda + a, x_1^2 + \lambda x_3 + x_3) \tag{4.6}$$

Donde:

$$\lambda = x_1 + \frac{y_1}{x_1} \tag{4.7}$$

La operación fundamental para el cifrado de datos utilizando ECC es la multiplicación escalar, mostrada en (4.8), donde $k \in E$ es un entero y P es un punto de la curva elíptica. Al igual que en RSA, existen diversos algoritmos para realizar esta operación. En el Algoritmo 4.1 se muestra la versión más básica.

$$Q = kP \tag{4.8}$$

Algoritmo 4.1 Doblar y sumar de izquierda a derecha

Entrada: Un punto $P \in E(\mathbb{F}_p)$, $k = (k_{n-1}, \cdots, k_0)_2 \in \mathbb{N}$
Salida: $Q = kP$

1: Hacer $R = P$
2: Hacer $Q = \begin{cases} O & k_0 = 0 \\ P & k_0 = 1 \end{cases}$
3: **para** i desde 1 hasta t **hacer**
4: Hacer $R = 2R$
5: **si** $n_i = 1$ **entonces**
6: $Q = Q + R$
7: **fin si**
8: **fin para**
9: **devolver** Q

El Algoritmo 4.1 presentado anteriormente utiliza las ecuaciones mostradas en (4.4) y (4.6), es decir, la suma de puntos y el doblado del punto.

4.2.1. Puntos proyectivos

Del apartado anterior, realizar las operaciones de suma y doblado de puntos requiere el cálculo de inversos durante la ejecución del Algoritmo 4.1. Esta operación del cálculo del inverso requiere una gran cantidad de operaciones. Para evitar este inconveniente, es apropiado representar los puntos en otro tipo de coordenadas.

La representación de los puntos en el apartado anterior es denotada como puntos en el plano afín. Una representación de los puntos sobre una curva elíptica que evita el cálculo de inversos es a través de la representación de puntos en el plano proyectivo.

Este subapartado es tomado de [40, 41]. Para definir el plano proyectivo, primero sea K un cuerpo. Sean dos ternas (x_1, y_1, z_1) y (x_2, y_2, z_2) se dice que son equivalentes, $\sim$, en el conjunto $K^3 \setminus (0,0,0)$ sí existe un elemento no nulo $\lambda \in K$ tal que $(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_2)$. A esta equivalencia la denotaremos como $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$. La equivalencia de (x, y, z) se denota como $(X : Y : Z)$ y se dice que es un punto proyectivo.

Si en la terna $(X : Y : Z)$ se tiene que $Z \neq 0$, entonces, $(X : Y : Z) = (X/Z : Y/Z : 1)$; la equivalencia anterior es el único representante con $Z = 1$. Es así, que se tiene una correspondencia uno a uno entre el conjunto de puntos proyectivos y los puntos afines. En caso contrario, si $Z = 0$, ese conjunto de puntos, $(X : Y : 0)$, en el plano proyectivo se nombran recta del infinito ya que no hay correspondencia con el plano afín. Esta representación de puntos proyectivos es llamada *coordenadas proyectivas estándar*.

Además, de la representación de coordenadas proyectivas estándar existen dos representaciones más: la representación de *coordenadas proyectivas Jacobianas* y las *coordenadas proyectivas López-Dahab*. Las coordenadas proyectivas López-Dahab se han utilizado recientemente de manera extensiva debido a la característica de reducción de recursos durante la implementación de ECC en dispositivos programables.

4.2.2. Protocolos criptográficos que usan Curvas Elípticas

Dentro de la literatura existen diversos protocolos que utilizan las Curvas Elípticas, sin embargo, la mayoría se basan en sistemas anteriores. A continuación, se

mencionan algunas de ellas:

El *protocolo ECDH* (ECDH de *Elliptic Curve Diffie-Hellman*) es un sistema para intercambio de claves entre dos entidades que utilizan el sistema Diffie-Hellman [60]. El protocolo funciona de la siguiente manera: Dos entidades, A y B, eligen una curva sobre $\mathbb{F}_p$ y un punto P perteneciente a dicha curva. La entidad A elige un valor secreto n_A perteneciente a $\mathbb{F}_p$ y calcula el valor $n_A P$ y lo hace público. De igual manera, la entidad B hace lo mismo, publicando el valor $n_B P$. Ahora, ambas entidades toman el valor público de su contraparte y calculan el valor $n_A n_B P$, la cual será la clave secreta. Nótese que ambas partes obtendrán el mismo valor secreto, es decir, $n_A n_B P$.

El *criptosistema ElGamal* se utiliza para el cifrado de información teniendo las claves secretas. Este criptosistema funciona de la siguiente manera: Para cifrar, se elige una curva sobre $\mathbb{Z}/p$ y un punto de la curva de orden grande N. Primero, si la entidad A quiere enviar un mensaje cifrado a B, A escoge un valor al azar k. Luego, A calcula un punto P_m que está asociado al mensaje m a ser cifrado. Se cifra el mensaje como $C = P_m + kP_B$. Así, A envía a B el valor de (C, kP). Así, B, para descifrar el mensaje deberá hacer: $P_m = C - n_B(kP)$ y de esta manera encuentra el mensaje m asociado al punto P_m.

El *criptosistema RSA* para curvas elípticas (esquema KMOV) es muy parecido al visto anteriormente en este trabajo. Su funcionamiento es como sigue: se toma una curva elíptica $E_n(b)$ de la forma $y^2 = x^3 + b$ en $\mathbb{Z}_n$. Para la generación de claves en este criptosistema, la entidad B escogerá dos números primos grandes (p, q) (como en RSA clásico) de tal manera que $p = q = n \mod 3$ y al igual que en RSA clásico, calculará los valores de $n = pq$, $\varphi(n) = (p-1)(q-1)$, y tomará un valor e tal que $1 < e < \varphi(n)$ con $mcd(e, \varphi(n)) = 1$, y finalmente calculará $d = e^{-1} \mod \varphi(n)$. De esta manera, se hacen públicos los valores (e, n) y se mantienen privados los valores restantes.

Para el cifrado de datos utilizando RSA con Curva Elíptica se procede de la siguiente manera: El usuario A divide su mensaje, m, en dos partes de tal manera que $m = (m_1, m_2)$ con la condicionante de que estén dentro de $\mathbb{Z}_n$. Ahora calcula $b = m_2^2 - m_1^3 \mod n$, esto produce que m esté dentro de la curva, es decir, $m \in E_n(b)$.

Ahora, cifra el punto m calculando el valor $E(m) = em$ sobre $E_n(b)$ y envía el texto cifrado como $c = (c_1, c_2)$. Para descifrar el mensaje, la entidad B realiza lo siguiente: se calcula $b = c_2^1 - c_1^3 \mod n$ y construye la curva $y^2 = x^3 + b$. Finalmente calcula $m = dc$ en $E_n(0, b)$.

Los anteriores criptosistemas son solo algunos de los propuestos, además, se puede utilizar una curva definida sobre los números primos o sobre los binarios.

Como se podrá observar, los anteriores criptosistemas utilizan el cálculo de la multiplicación escalar, vista en (4.8), de un valor entero por el punto perteneciente a una curva elíptica. Por lo tanto, se hace necesario el Algoritmo 4.1 para generar claves, cifrar y/o descifrar la información utilizando Curvas Elípticas.

4.3. SCA sobre ECC

Nótese que el Algoritmo 4.1 es muy parecido algorítmicamente al presentado en el Algoritmo 3.1 del cifrado de datos con RSA. Lo cual denota las mismas debilidades en ECC que en RSA cuando es atacado con SCAs, por lo cual conlleva a una forma de atacar ECC muy similar a RSA. Es decir, el Algoritmo 4.1 en sus trazos de potencia, muestra cuando se está realizando una suma de puntos y cuando se realiza una doble suma de puntos (o duplicación de punto) dentro de la curva elíptica. Como fue mencionado anteriormente, esto conlleva a poder realizar una medición, ya sea de potencia o de tiempo, del valor de la clave privada, k.

Para solventar el problema anterior, al igual que en RSA, se han propuesto diversos algoritmos que evitan las vulnerabilidades vistas en RSA para ECC.

Una primera medida consiste en utilizar una *representación regular del valor escalar*, k, [62]. Una segunda medida consiste en utilizar el algoritmo *double-and-add-always* presentado por Coron [47]. Una tercera medida es utilizar *Montgomery-Ladder* sobre campos de los números primos [63]. Finalmente, se han propuestos diversas curvas elípticas que tienen la característica de ser resistentes a los SCAs, por ejemplo, las curvas *Edwards* [64].

Una propuesta interesante para evitar los SCAs es utilizar la Multiplicación Montgomery para puntos en una Curva Elíptica. Dicha propuesta tiene la ventaja de utilizar un algoritmo regular para evitar especialmente los TAAs. En Algoritmo 4.2 se muestra el algoritmo mencionado.

Algoritmo 4.2 Multiplicación Montgomery para ECC.

Entrada: Un entero $k \geq 0$ y un punto $P = (x, y) \in E$.
Salida: $Q = kP$

1: **si** $k = 0$ o $x = 0$ **entonces**
2: **devolver** $(0,0)$ y detener.
3: **fin si**
4: Hacer $k = (k_{l-1}, \cdots, k_1, k_0)_2$
5: Hacer $x_1 \leftarrow x$, $x_2 \leftarrow x^2 + b/x^2$
6: **para** i desde $l-2$ hasta 0 **hacer**
7: Hacer $t \leftarrow \frac{x_1}{x_1+x_2}$
8: **si** $k_i = 1$ **entonces**
9: Hacer $x_1 \leftarrow x + t^2 + t$, $x_2 \leftarrow x_2^2 + b/x_2^2$
10: **si no**
11: $x_1 \leftarrow x_1^2 + b/x_1^2$, $x_2 \leftarrow x + t^2 + t$
12: **fin si**
13: **fin para**
14: Hacer $r_1 \leftarrow x_1 + x$, $r_2 \leftarrow x_2 + x$
15: Hacer $y_1 \leftarrow \frac{r_1(r_1 r_2 + x^2 + y)}{x} + y$
16: **devolver** $Q = (x_1, y_1)$

Nótese que el anterior algoritmo utiliza coordenadas afines, lo cual, representa el calcular inversos durante su ejecución. El cálculo de dichos valores inversos, como fue mencionado anteriormente, conlleva a un gran consumo en los recursos del sistema donde será implementado.

Debido a lo anterior, se ha propuesto utilizar el plano proyectivo de la curva elíptica. Existen diversas formas de representar los puntos de una curva elíptica en el plano proyectivo. Uno muy interesante, ya que elimina el uso del cálculo de los inversos es la representación de los puntos a través de la propuesta realizada por López-Dahab y utilizando la multiplicación Montgomery vista en el Algoritmo 4.2. En el Algoritmo 4.3 se muestra dicha propuesta [40].

Como en el caso de RSA, el algoritmo anterior se puede atacar con DPA, sin embargo, existen diversas contramedidas para evitar estos ataques, a continuación,

Algoritmo 4.3 Multiplicación de puntos de Montgomery (para curvas elípticas sobre $F_(2^m)$).

Entrada: $k = (k_{t-1}, \cdots, k_1, k_0)_2$ con $k_{t-1} = 1$, $P = (x, y) \in E(\mathbb{F}_{2^m})$
Salida: $Q = kP$

1: $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$ ▷ Calcula $(P, 2P)$
2: **para** i desde $t-2$ hasta 0 **hacer**
3: **si** $k_i = 1$ **entonces**
4: $T \leftarrow Z_1$, $Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_1 \leftarrow xZ_1 + X_1 X_2 T Z_2$
5: $T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + bZ_2^4$, $Z_2 \leftarrow T^2 Z_2^2$
6: **si no**
7: $T \leftarrow Z_2$, $Z_2 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_2 \leftarrow xZ_2 + X_1 X_2 T Z_1$
8: $T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + bZ_1^4$, $Z_1 \leftarrow T^2 Z_1^2$
9: **fin si**
10: **fin para**
11: $x_3 \leftarrow X_1 / Z_1$
12: $y_3 \leftarrow (x + X_1/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1 Z_2)](xZ_1 Z_2)^{-1} + y$
13: **devolver** $Q = (x_3, y_3)$

mencionaremos algunos [40, 47]:

- La primera contramedida consiste en calcular un valor aleatorio $d \in E$ y calcular $k' = k + d.\#E$. Al momento de calcular el punto $Q = k'P$ se obtendrá $Q = kP$ esto debido a que $\#E.P = \infty$. Otra medida sería calcular un nuevo valor de k cada vez que se cifre/descifre un mensaje nuevo. De esta manera siempre se tendrá un nuevo valor de k, evitando así los DPA.

- Una segunda medida es blindar el punto P. Esto se realiza calculando un punto aleatorio $R \in E$ y agregándolo al punto P. Sin embargo, en esta medida se necesita tener guardado los punto P y R, lo cual incrementa los recursos y tiempo de ejecución debido a los cálculos extras que requiere el algoritmo.

- Una tercera medida es realizar las operaciones de suma y multiplicación dentro del algoritmo utilizando diferentes representaciones de los puntos, es decir, elegir de manera aleatoria una representación de los puntos.

- Finalmente, entre otras, se ha propuesto el uso de formas especiales de curvas elípticas como lo son las propuestas por Montgomery, Jacobianas, Hessianas, entre otras.

4.3.1. Ataques a la multiplicación de puntos de Montgomery

Aunque anteriormente se mencionaron algunas contramedidas para evitar los ataques por medición de potencia, existen diversos ataques como llamado *Ataque Doble*, (DA de *Doubling Attack*), el *Ataque de Potencia Refinado* (RPA de *Refined Power Attack*) y el *Ataque de Puntos de Valor Cero* (ZPA de *Zero-Value Point Attack*).

Para evitar los ataques anteriores Mayima *et al*. Propusieron un algoritmo llamado *BRIP* (de *Binary Random Initial Point*), al igual que lo hicieron con el algoritmo anteriormente mencionado en el capítulo anterior. En el Algoritmo 4.4 se muestra dicha propuesta.

Algoritmo 4.4 BRIP para ECC.

Entrada: $k = (k_{t-1}, \cdots, k_1, k_0)_2$ con $k_{t-1} = 1$, $P = (x, y) \in E(\mathbb{F}_{2^m})$
Salida: $Q = kP$

1: Elegir un punto aleatorio $R \in E$
2: $T[0] \leftarrow R$, $T[1] \leftarrow -R$, $T[2] \leftarrow P - R$
3: **para** i desde $t-1$ hasta 0 **hacer**
4: $\quad T[0] \leftarrow 2T[0]$
5: $\quad T[0] \leftarrow T[0] + T[d_i + 1]$
6: **fin para**
7: **devolver** $Q = T[0] + T[1]$

Existe un ataque análogo al Ataque $N-1$ para curvas elípticas llamado *2-torsion Attack*, 2TT. El comportamiento es similar al antes mencionado ataque $N-1$, el comportamiento toma el hecho de que durante la ejecución del algoritmo BRIP siempre se calcula un punto más el punto aleatorio R. Si un atacante encuentra un punto G dentro de la curva de tal manera que al calcular $2G = \infty$, se tendrá el siguiente comportamiento durante cada iteración del algoritmo cuando es ingresado dicho punto:

- Si $t_i = 1$ entonces el valor $T[0] = 2(G + R)$
- De lo contrario si $t_i = 0$ el valor obtenido es $T[0] = 2R$

De esta forma durante la ejecución del algoritmo siempre se obtendrán dos patrones en las medidas de potencia. Llegando así a la analogía con el ataque $N-1$.

4.3.2. Algoritmo de Multiplicación Escalar que Oculta el Mensaje

Una propuesta muy interesante para evitar no solo los ataques 2TT sino además los ataques DPA, DA, RPA y ZPA es el propuesto por JaeCheol *et al*., llamado *Algoritmo de Multiplicación Escalar que Oculta el Mensaje* (MBSM de *Message-Blinding Scalar Multiplication*) [65].

Aunque se han propuesto mejoras al algoritmo MBSM donde se evitan no solo ataques pasivos sino también ataques activos, por ejemplo, los FA, éstos tienen el inconveniente de requerir más recursos (no solo por el requerir una mayor cantidad de registros sino también por la necesidad de requerir hacer comparaciones) por parte del dispositivo que implemente dichos algoritmos. Realizar comparaciones dentro de dispositivos programables como lo son los FPGAs podría conllevan a un gran consumo de sus recursos, especialmente en dispositivos con poca cantidad de estos.

En el Algoritmo 4.5 se muestra el algoritmo MBSM.

Algoritmo 4.5 MBSM.

Entrada: $k = (k_{t-1}, \cdots, k_1, k_0)_2$ con $k_{t-1} = 1$, $P = (x, y) \in E(\mathbb{F}_{2^m})$
Salida: $Q = kP$

1: Precálculo: $d \leftarrow \omega \cdot \#\varepsilon + k - (2^{t-1} - 1)$, $s \leftarrow \#\varepsilon - k$
2: Escoger un punto aleatorio $R \in E$ y dos bits u y v
3: $T[00 \oplus uv] \leftarrow P + R$, $T[01 \oplus uv] \leftarrow P + 2R$, $T[10 \oplus uv] \leftarrow 2P + 2R$ y $T[11 \oplus uv] \leftarrow 2P + 3R$
4: Evaluar: $Q \leftarrow T[d_{n-1}s_{n-1} \oplus uv]$
5: **para** i desde $n-2$ hasta 0 **hacer**
6: $\quad Q \leftarrow 2Q$
7: $\quad Q \leftarrow Q + T[d_i s_i \oplus uv]$
8: **fin para**
9: **devolver** Q

Un breve análisis de seguridad del algoritmo anterior se muestra en seguida: Si un atacante ingresa un mensaje con valor $2P$ o G (2-torsion punto de la curva elíptica) no observará dos patrones de potencia como se espera (como fue mostrado en el ataque $N-1$). Durante las operaciones intermedias ejecutadas por el algoritmo se mostrarán los valores relacionados con T y R, observando el algoritmo se nota que dichos valores son aleatorios por lo cual, aun ingresando el punto 2-torsion es difícil obtener dos patrones. Lo anterior hace difícil obtener los patrones para encontrar el valor de k,

haciendo así, resistente el algoritmo a los SCAs.

Aunque este algoritmo es muy interesante, tiene varias desventajas para ser implementado dentro de un FPGA. El primero de ellos es debido a que no se ha comprobado si efectivamente es resistente a los 2-torsion attacks. El segundo es debido al gran consumo de recursos que conlleva la cantidad de registros necesarios para su implementación. El segundo es la generación de números aleatorios, aun siendo dos bits.

4.4. Implementación en FPGA de la Multiplicación Escalar

Primero se mostrará los diferentes componentes para la implementación del algoritmo de cifrado con curvas elípticas. Como se ha mencionado anteriormente, nuestro estudio está enfocado en las curvas definidas sobre los campos binarios módulo $f(z)$.

Para calcular la multiplicación escalar se debe definir entonces las operaciones adición, multiplicación, cuadrado e inversión en campos finitos modulo $f(z)$.

4.4.1. Operaciones en campos finitos

Adición

Esta operación es la más sencilla de realizar. Para esta operación únicamente se utiliza el operador binario XOR bit a bit entre los dos operandos.

Multiplicación

Para realizar la multiplicación se toma como base el algoritmo mostrado en [40], dicho algoritmo se muestra en el Algoritmo 4.6.

Donde, la operación $b \leftarrow b \cdot z \mod f(z)$ representa corrimientos a la izquierda del

Algoritmo 4.6 Multiplicación en campo en F_{2^m}, Derecha a Izquierda, Rotar y Sumar.

Entrada: Polinomio $a(z)$ y $b(z)$ de grado al menos $m-1$
Salida: $c(z) = a(z) \cdot b(z) \mod f(z)$

1: **si** $a_0 = 1$ **entonces**
2: $\quad c \leftarrow b$
3: **si no**
4: $\quad c \leftarrow 0$
5: **fin si**
6: **para** $i = 1$ hasta $m-$ **hacer**
7: $\quad b \leftarrow b \cdot z \mod f(z)$
8: $\quad$ **si** $a_i = 1$ **entonces**
9: $\quad\quad c \leftarrow c + b$
10: $\quad$ **fin si**
11: **fin para**
12: **devolver** (c)

vector $b(z)$, seguido de una la suma de $r(z)$ a $b(z)$ si el bit más significativo de b es 1. Estas operaciones internas del bucle en el algoritmo anterior se pueden implementar en hardware utilizando un solo ciclo máquina, como es mostrado en el Algoritmo 4.7.

Algoritmo 4.7 Multiplicador MSB (Most significant bit first) para F_{2^m}.

Entrada: $a = (a_{m-1}, \cdots, a_1, a_0)$, $b = (b_{m-1}, \cdots, b_1, b_0)$, y un polinomio de reducción $f(z) = z^m + r(z)$
Salida: $a \cdot b$

1: Hacer $c \leftarrow 0$
2: **para** i desde $m-1$ hasta 0 **hacer**
3: $\quad c \leftarrow (c \ll 1) \oplus (c_{m-1} \& r)$
4: $\quad c \leftarrow c \oplus (b_i \& a)$
5: **fin para**
6: **devolver** (c)

En donde los operadores $\ll$, $\oplus$ y & representan las operaciones binarias de corrimiento a la izquierda, or exclusiva y and, respectivamente.

Cuadrado

Desde el punto de vista de hardware, elevar el cuadrado un número en el campo de los binarios es fácil de realizar. Esta operación se realiza solo insertando 0s después de cada bit en el dato de entrada, como es mostrado en (4.9) y (4.10).

$$a(z) = a_(m-1)z^(m-1) + \cdots + a_2 z^2 + a_1 z + a_0 \tag{4.9}$$

$$a(z)^2 = a_(m-1)z^(2m-2) + 0 + \cdots + a_2 z^4 + 0 + a_1 z^2 + 0 + a_0 \tag{4.10}$$

Reducción

La reducción en campos finitos se realiza después de cada elevación al cuadrado y/o multiplicación. Para realizar esta operación se ha utilizado el algoritmo mostrado en [40]. Sin embargo, dicho algoritmo puede ser simplificado tomando en cuenta las curvas propuestas por el NIST. Por ejemplo, en este trabajo trabajaremos con la curva 2^{163}. Para dicha curva, el algoritmo de reducción se muestra en el Algoritmo 4.8.

Algoritmo 4.8 Reducción rápida módulo $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$.

Entrada: Polinomio $a(x)$ con 325 bits
Salida: $r(x)$ con 163 bits

1: $M \leftarrow a[i] \oplus a[i+163] \oplus a[i+319]$
2: $W \leftarrow a[i] \oplus a[i+157] \oplus a[i+160]$
3: **para** $i = 0$ hasta 162 **hacer**
4: Si $0 \leq i \leq 1$ entonces: $r[i] \leftarrow M \oplus a[i+320] \oplus a[i+323]$
5: Si $i == 2$ entonces: $r[i] \leftarrow M \oplus a[i+320]$
6: Si $3 \leq i \leq 5$ entonces: $r[i] \leftarrow M \oplus a[i+160] \oplus a[i+316] \oplus a[i+317]$
7: Si $i == 6$ entonces: $r[i] \leftarrow W \oplus a[i+163] \oplus a[i+313] \oplus a[i+314] \oplus a[i+316]$
8: Si $7 \leq i \leq 10$ entonces: $r[i] \leftarrow W \oplus a[i+156] \oplus a[i+163] \oplus a[i+312] \oplus a[i+314]$
9: Si $11 \leq u \leq 12$ entonces: $r[i] \leftarrow W \oplus a[i+156] \oplus a[i+163] \oplus a[i+312]$
10: Si $13 \leq i \leq 161$ entonces: $r[i] \leftarrow W \oplus a[i+156] \oplus a[i+163]$
11: Si $i = 162$ entonces: $r[i] \leftarrow W \oplus a[i+156]$
12: **fin para**

Este algoritmo no fue implementado en el diseño final, en su lugar se ha implementado el multiplicador en campos finitos visto en el Algoritmo 4.7. De esta manera, se reducen la cantidad de recursos utilizados en el FPGA. El inconveniente de este proceso es el aumento en el tiempo de procesamiento.

4.4.2. Implementación de la aritmética en campos finitos binarios

Los componentes anteriores se han implementado en el FPGA, para lo cual se ha propuesto el diagrama a bloques mostrado en la Fig. 4.3.

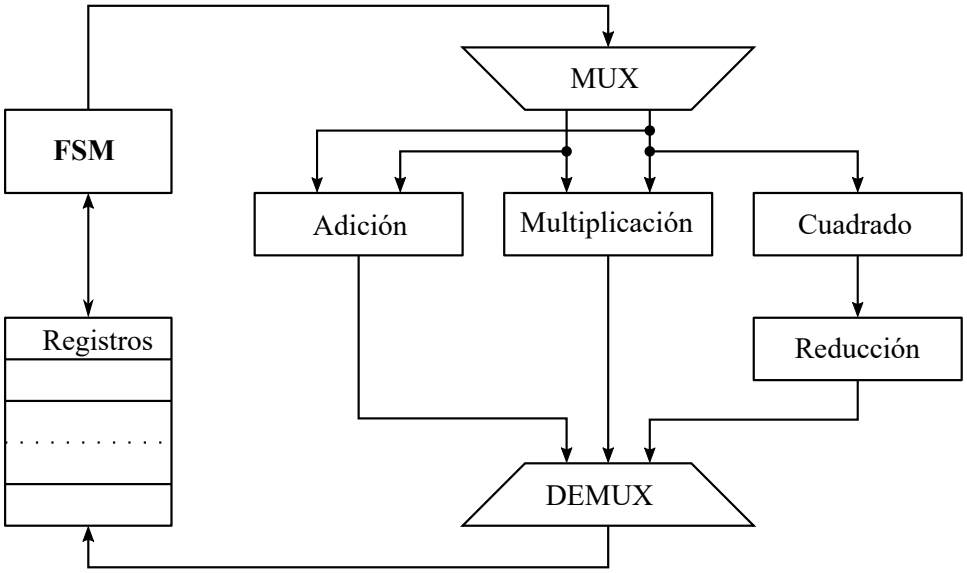


Fig. 4.3. Diagrama a bloques del controlador para la multiplicación escalar.

Donde el FSM es el bloque encargado de controlar las operaciones que se van a realizar.

Los parámetros utilizados, para la implementación de este trabajo, son los recomendados por el NIST. Se ha utilizado la curva elíptica B-163, cuyos parámetros se muestran en la Tabla 4.1.

Tabla. 4.1. Parámetros de la curva elíptica utilizada B-163

B-163: $m = 163$, $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, $a = 1$, $h + 2$
$S = 0 \times 85E25BFE5C86226CDB12016F7553F9D0E693A268$
$b = 0 \times 000000020A601907B8C953CA1481EB10512F78744A3205FD$
$n = 0 \times 00000004000000000000000000292FE77E70C12A4234C33$
$x = 0 \times 00000003F0EBA16286A2D57EA0991168D4994637E8343E36$
$y = 0 \times 00000000D51FBC6C71A0094FA2CDD545B11C5C0C797324F1$

Donde:

m: Es el grado de extensión del campo binario $\mathbb{F}_{2^m}$.

$f(z)$: Es el polinomio de reducción de grado m.

S: Es la semilla seleccionada aleatoriamente que genera los coeficientes de la curva elíptica.

a y b: Son los coeficientes de la curva elíptica $y^2 + xy = x^3 + ax^2 + b$.

n: Es el orden (primo) del punto base P.

h: Es el cofactor. x, y: Son las coordenadas de P.

Para realizar las operaciones sobre los puntos en el Algoritmo 4.4 se han utilizado los puntos en su representación en coordenadas proyectivas utilizando Lopez-Dahab. Para dicha representación se ha modificado el Algoritmo 4.4 para que funcione con las coordenadas proyectivas Lopez-Dahab (LD), como se muestra en el Algoritmo 4.9. Como se mencionó anteriormente, el uso de coordenadas proyectivas LD representa una ventaja debido a que reduce la cantidad de recursos en la implementación; esto lo podemos observar debido a que en comparación con las coordenadas afines no se requiere el cálculo de inversos. El cálculo del inverso de un número módulo $f(z)$, desde el punto de vista de hardware, conlleva al consumo de una gran cantidad de recursos y tiempo de procesamiento.

Algoritmo 4.9 BRIP con coordenadas LD propuesto para implementación en FPGA.

Entrada: $k = (k_{t-1}, \cdots, k_0)_2$ con $k_{t-1} = 1$, un punto $P = (x, y) \in E$
Salida: $Q = kP$

1: Elegir un punto aleatorio $R \in E$ en coordenadas LD
2: Hacer $P = (X_1, Z_1)$ donde $X_1 = x$ y $Z_1 = 1$
3: $T[0] = R$, $T[1] = -R$, $T[2] = P + R$
4: **para** i desde $t-1$ hasta 0 **hacer**
5: $\quad T[0] = Mdouble[T[0]]$
6: $\quad T[0] = Madd[T[0] + T[d_i + 1]]$
7: **fin para**
8: **devolver** $Q = Madd[T[0] + T[1]]$

Las funciones Mdouble y Madd se definen en los Algoritmo 4.10 y Algoritmo 4.11.

El consumo de recursos utilizado de la implementación realizada se puede observar en la Tabla 4.2.

Los tiempos para realizar el cálculo de la multiplicación escalar para el cifrado y descifrado de los datos utilizando ECC B-163, utilizando el FPGA con una frecuencia

Algoritmo 4.10 Mdouble. Doble de un punto en coordenadas LD.

Entrada: Elementos del campo a y $c = b^{2m-1}$ $(c^2 = b)$ definidos sobre la curva E; la coordenada X/Z para el punto P.

Salida: La coordenada x/Z para el punto $2P$

1: $T_1 \leftarrow c$
2: $X \leftarrow X^2$
3: $Z \leftarrow Z^2$
4: $T_1 \leftarrow Z \times T_1$
5: $Z \leftarrow Z \times X$
6: $T_1 \leftarrow T_1^2$
7: $X \leftarrow X^2$
8: $X \leftarrow X + T_1$

Algoritmo 4.11 Madd. Suma de puntos en coordenadas LD.

Entrada: Los elementos a y b definidos sobre la curva E, la coordenada $X_1.Z_1$ y X_2/Z_2 parra los puntos P_1 y P_2 sobre E.

Salida: La coordenada X_1/Z_1 para el punto $P_1 + P_2$.

1: $T_1 \leftarrow x$
2: $X_1 \leftarrow X_1 \times Z_2$
3: $Z_1 \leftarrow Z_1 \times X_2$
4: $T_2 \leftarrow X_1 \times Z_1$
5: $Z_1 \leftarrow Z_1 \times +X_1$
6: $Z_1 \leftarrow Z_1^2$
7: $X_1 \leftarrow Z_1 \times T_1$
8: $X_1 \leftarrow X_1 + T_2$

Tabla. 4.2. Recursos utilizados para la implementación de la ECC B-163

	LUTs (63,400)	**Registers** (126,800)	**Slice** (15,859)	**BRAM** (135)	**DSP** (240)
ECC B-163	2,525	2,446	954	0	0
▷ Multiplicador MSB	778	362	335	0	0

de reloj de 100MHz, son los siguientes: La multiplicación escalar completa lleva un tiempo de $8,870,720ns$, la función Madd lleva un tiempo de $24,660ns$, y finalmente la función Mdouble consume un tiempo total de $29,599ns$.

Como se ha mencionado anteriormente, la operación del inverso, desde el punto de vista de hardware, conlleva a un gran consumo de recursos; debido a lo anterior se ha establecido no implementar la operación del inverso en el FPGA. Para poder convertir las coordenadas LD a coordenadas Afines se ha de realizar por software.

4.5. Ataques a la implementación ECC

Como fue explicado anteriormente, se implementó la curva elíptica B-163. El cifrado de datos fue atacado utilizando dicha curva. Sin embargo, para atacarla solo se realizó la medición de potencia simple, debido a que no se conoce un punto de torsión que genere el ataque 2-torsion.

En la Fig. 4.4 se observa el inicio y final de los cálculos realizados por la multiplicación escalar sobre la curva B-163.

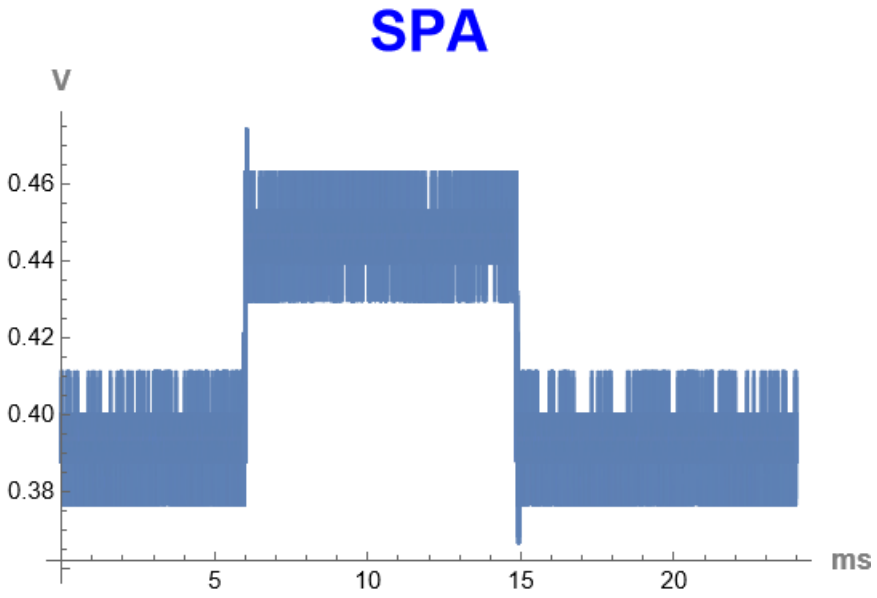


Fig. 4.4. SPA sobre B-163.

En la Fig. 4.5 se muestra un acercamiento a la medición. Como se puede notar en

la gráfica, se observa el inicio de los cálculos.

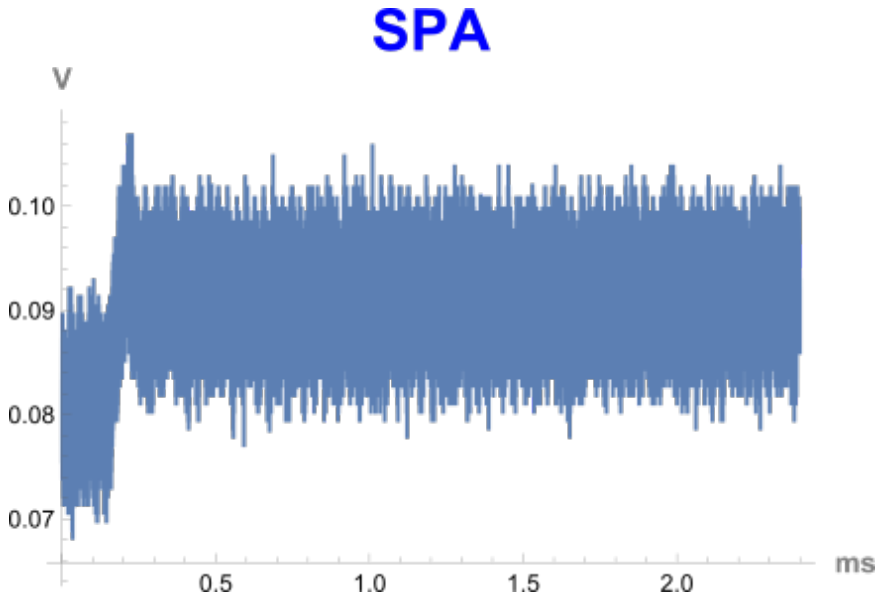


Fig. 4.5. Inicio del cálculo de la multiplicación escalar para B-163.

Para hacer notar las operaciones realizadas en el algoritmo de cifrado, se realizaron diversas mediciones. En la Fig. 4.6 se muestra un acercamiento con los tiempos adecuados para observar las operaciones de Mdouble y Madd, empero, no se observa una distinción en la gráfica de ambas operaciones.

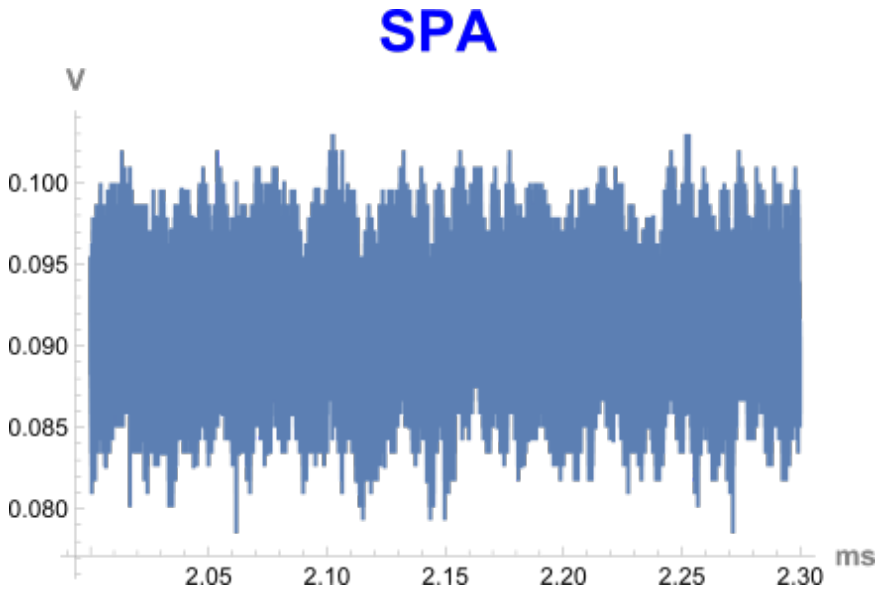


Fig. 4.6. Acercamiento al SPA sobre ECC B-163.

4.6. Conclusiones

Durante el desarrollo de este capítulo observamos las ventajas de utilizar curvas elípticas para el cifrado de datos. No únicamente estas curvas se utilizan para el cifrado, sino también, como esquemas de intercambio de claves.

Hemos observado además que en la literatura existen ataques similares, en operación y metodología, al ataque $N-1$. Sin embargo, no se logró realizar dichos ataques debido a que no se conoce un punto de torsión adecuado para poder generarlo. En la literatura ya existen algoritmos adecuados para evitar este tipo de ataques llamados 2-torsion attack.

La implementación de curvas elípticas en FPGA, comparado con RSA, es sencilla de realizar y requiere menos recursos (esto se analizará en las secciones de resultados de esta tesis). Cabe aclarar que las curvas implementadas en este trabajo son sobre el campo binario. Para poder realizar criptografía con curvas elípticas en otros campos se requiere de la implementación adecuada de los multiplicadores, aunque bien puede utilizarse el multiplicador Montgomery con sus mejoras de implementación en FPGA como el mostrado en este trabajo.

Finalmente, a pesar de que solo se han mostrado resultados de la implementación de la curva B-163, también se desarrollaron las curvas B-233 y B-571. La implementación para dichas curvas fue posible, debido a la modularidad de utilizar el FPGA, ya que permite realizar cambios rápidos, esto debido a que básicamente (con algunos detalles) se cambian los tamaños de los vectores

Capítulo 5

Análisis del cifrado de imágenes utilizando RSA y ECC

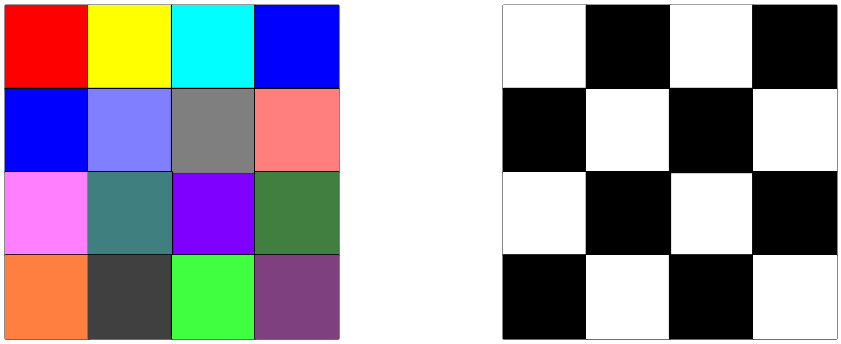
Como fue mencionado en capítulos anteriores el procesado y cifrado de imágenes son tareas importantes en la comunicación entre diferentes identidades, como lo son entre los gobiernos, entidades empresariales y personas en general. En este capítulo examinaremos el cifrado de imágenes con los métodos de procesamiento de imágenes y cifrado expuesto en capítulos anteriores.

5.1. Representación de una imagen para el cifrado

Aunque en la literatura existen diferentes propuestas para el cifrado de imágenes como los mostrados en los trabajos [11, 12, 66–69], pocos de ellos tratan del cifrado de imágenes utilizando redes neuronales, específicamente aquellos implementados en FPGA.

Generalmente el cifrado de imágenes se realiza utilizando los tres canales de color, es decir, el Verde, Rojo y Azul; sin embargo, el manejo de los tres canales conlleva a una gran cantidad de datos a ser cifrados. Una ventaja de utilizar un procesador de imágenes con CeNN es justamente la reducción de datos a ser cifrados.

El utilizar la CeNN para procesar imágenes es interesante, ya que son capaces de entregar imágenes procesadas en color o en escalas de grises. En este trabajo hemos utilizado la CeNN para procesar imágenes en escala de grises entregando a su salida imágenes en escala de blanco y negro. Justamente, las imágenes en escala de blanco y negro representan una reducción en los datos a ser cifrados. Para ejemplificar lo anterior suponga que se tiene una pequeña imagen a color como la mostrada en la Fig. 5.1a , y la misma imagen, pero ahora después de haberla procesada con la CeNN en escala en blanco y negro es mostrada en la Fig. 5.1b.



(a) Imagen a color **(b)** Imagen en blanco y negro

Fig. 5.1. Representación de los pixeles de una imagen.

Una figura en blanco y negro, como la mostrada en la Fig. 5.1b, la podemos representar como una matriz de datos, como se muestra a continuación.

$$\begin{bmatrix} -1 & +1 & -1 & +1 \\ +1 & -1 & +1 & -1 \\ -1 & +1 & -1 & +1 \\ +1 & -1 & +1 & -1 \end{bmatrix} \tag{5.1}$$

Donde un bit en blanco toma el valor de -1 y uno en negro toma el valor de +1, como fue mostrado en capítulos anteriores. De esta forma, observamos que una imagen procesada con una CeNN la podemos representar con únicamente dos valores numéricos, -1 y +1. Estos dos valores los podemos entonces codificar como un 1 o un 0.

5.2. Codificación de imágenes para cifrado

Tomando en cuenta la implementación de la CeNN realizada en este trabajo, se tienen bloques de 8×8 células, con esto podemos representar 64 pixeles de información de la imagen en blanco y negro. Considerando lo mencionado en la subsección anterior, podemos entonces tener una matriz de información de 8×8 bits. Con esto observamos nuevamente que en lugar de representar un bloque de 8×8 células con 64 bytes de información, donde cada byte representa 1 píxel de la imagen, se tiene únicamente 64 bits u 8 bytes de información a ser cifrada. Esta reducción de datos permite aumentar la velocidad del cifrado de la imagen.

Para aumentar la seguridad en el cifrado de la imagen se propone utilizar como método de codificación la permutación de Confusión en Zigzag [68,70]. Dicho método consiste en tomar un bit al azar dentro de la matriz, a partir de dicho bit se van tomando el resto de los bits en forma de zigzag, con esto se forma un vector que representa dicha matriz; como se muestra en la Fig. 5.2. Este método es simple de implementar y satisface el criterio de Shannon de confusión y difusión [70], aumentando así la seguridad en la implementación.

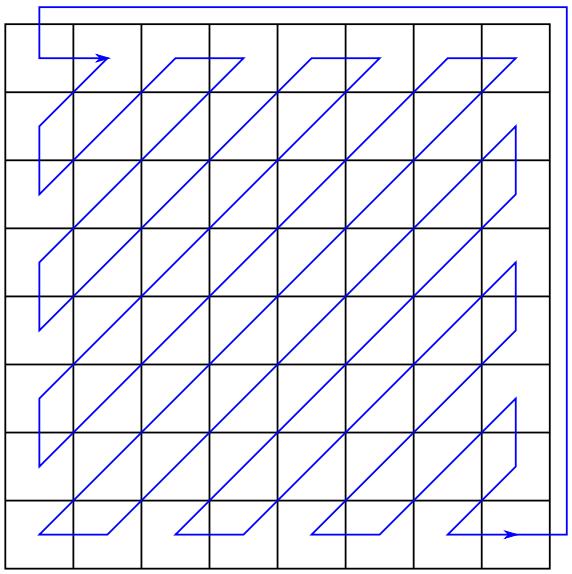


Fig. 5.2. Patrón de zigzag para la codificación de un bloque de pixeles.

Para el análisis se tomará como punto de partida el primer bit de la matriz de pixeles, evitando así la generación de números aleatorios dentro del FPGA. Además, hacer

esto nos permite realizar mediciones y observar el comportamiento de los algoritmos de cifrado durante la medición de potencia.

De esta forma, el vector obtenido es de 64 bits. Este vector tiene un orden de bits según lo mostrado en la Fig. 5.2. De este vector se tiene que el primer bit, es decir, el que representa al primer píxel dentro de la matriz, es el bit de mayor peso dentro de dicho vector, y el último bit, que representa al último píxel dentro de la matriz, es el de menor peso. Sin embargo, como fue mencionado anteriormente, este orden puede ser cambiado utilizando un bit de inicio aleatoriamente dentro de la matriz.

Tomando en cuenta que se están utilizando bloques de 64 bits, el cifrado de la imagen completa será entonces tomada en bloques de 8×8 bits.

Para el cifrado utilizando RSA de 1024 bits se tiene que se necesitan un total de 16 bloques, sin embargo, debido a que el valor binario puede ser mayor que el valor del módulo N, se ha decido tomar un total de 15 bloques. Esto evita, el realizar el cálculo del valor $(\textit{valor binario}) \mod N$. Teniendo de esta forma un total de 960 pixeles a ser cifrados. Para el estudio de este trabajo, se ha decido rellenar el resto de los bits (para completar los 1024 bits) con un valor de 0s.

Para el cifrado utilizando ECC B-163 y siguiendo con la misma idea implementada en RSA con una longitud de 163 bits se requerirían 2.54 bloques. Sin embargo, como se explicará más adelante, se propondrá un método diferente para la representación de los bloques como puntos de la curva elíptica.

5.2.1. Codificación de los bloques de imagen de la CeNN

Tomando en cuenta la estructura entregada por la CeNN y además los valores para la representación de los pixeles, se tiene que numéricamente el resultado entregado está entre los valores de $[-1, +1]$. Esta representación numérica está conformada por 18 bits, donde el bit más significativo representa el signo, de tal manera que, un bit de 1 representa un valor negativo y un 0 representa un valor positivo; justamente, estos valores corresponden a la representación de los pixeles blancos y negros.

Considerando el bit de signo mostrado anteriormente, podemos, entonces, almacenar en la RAM interna del FPGA únicamente el bit del signo. Reduciendo así la cantidad de recursos utilizados para la implementación.

Para la implementación dentro del FGPA se utiliza la memoria interna. El uso de dicha memoria interna permite que no se utilice una externa. Esto es ventajoso debido a que evita ataques a la memoria externa o que un ente externo lea dicha memoria obteniendo así los valores del mensaje original.

Se ha divido el vector de 64 bits en 4 bloques de un ancho de palabra de 16 bits. Estos cuatro bloques de palabra son los ingresados a la memoria RAM. Para esto se ha adecuado la CeNN para poder guardar los datos según lo mostrado anteriormente.

En la Fig. 5.3 se muestra el controlador final para la realización de este trabajo.

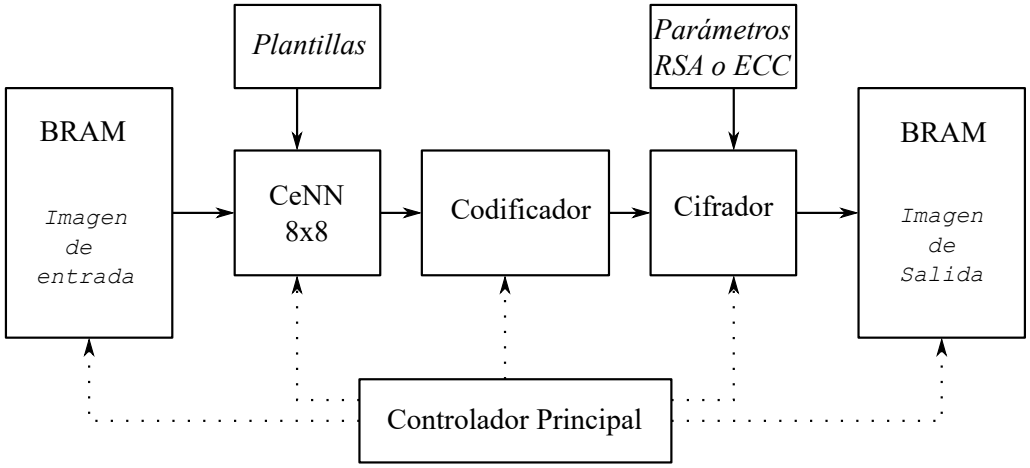


Fig. 5.3. Control de cifrado de imágenes.

Donde el primer bloque *BRAM* contiene la imagen a ser cifrada, el bloque *CeNN* contiene la red neuronal encargada de procesar la imagen, el *Codificador* es el encargado de la codificación de la imagen entregada por la CeNN, además de contener los bloques de imagen, entregados por la CeNN, necesarios para completar los bits necesarios para el cifrado, el bloque *Cifrador* es el encargado de cifrar la imagen ya sea utilizando RSA o ECC, y el segundo bloque BRAM es el que contiene la imagen cifrada.

Finalmente, el bloque *Controlador Principal* es el encargo de, como su nombre

indica, controlar el flujo de datos entre los diferentes bloques constitutivos.

5.3. Cifrado de imágenes usando RSA

Como se mencionó anteriormente se ha decidido tomar vectores de 960 bits de la imagen final (después de ser procesada por la CeNN) y después rellenar el resto de los 1024 bits con 0*s*. Para imágenes pequeñas, donde se entreguen imágenes con valores menores a 960 bits, se puede rellenar el resto de los bits con 0s o utilizar algún método de relleno como es mostrado en la literatura. Con esto se obtiene el vector mostrado en la Fig. 5.4. Este vector es el utilizado como mensaje para el cifrado.

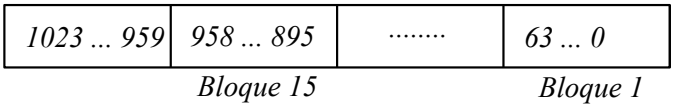


Fig. 5.4. Vector creado con los bloques CeNN para cifrado con RSA.

Una ventaja de utilizar RSA para el cifrado de la imagen es el no requerir hacer la conversión de los datos a ser cifrados, como veremos es el caso para curvas elípticas. Esto nos permite tomar los valores directamente de la imagen procesada y pasarlos al Codificador y después al Cifrador.

5.4. Cifrado de imágenes usando ECC

Para el cifrado de las imágenes procesadas con la CeNN, a diferencia del cifrado con RSA, se requiere que el dato a ser cifrado este dentro de un punto de la curva $E(\mathbb{F}_{2^m})$. Existen diversos algoritmos para la representación de un dato como un punto dentro de la curva, sin embargo, estos algoritmos no tienen un tiempo fijo para dicha conversión y requiere del cálculo de diversas funciones. El no tener un tiempo fijo para el mapeo entre el dato y el punto de la curva puede conllevar un riesgo en la seguridad de la información.

5.4.1. Cifrado de imágenes. Método 1: Utilizando mapeo de puntos

Existe en la literatura un método el cual realiza un mapeo del dato con puntos dentro de la curva, como, por ejemplo, el mostrado en [71]. Se toma como base dicho método en este trabajo para adaptarlo al cifrado de imágenes, y el funcionamiento de dicha propuesta es como sigue:

Primero se establece un punto G aleatorio en la curva elíptica, dicho punto debe conocerse tanto por el ente que cifra la información como por el ente que descifra la información; este punto puede considerarse como la clave privada. Teniendo dicho punto G se calcula el valor kG donde k está comprendido entre los valores de 2 y $2^m + 1$, con $m = 8$. Con esto se pueden crear 256 puntos sobre la curva elíptica, los puntos obtenidos se pueden acomodar en un bloque de RAM donde se tienen 256 palabras con un ancho de dirección de 8 bits, y con un ancho de palabra de 163 bits.

El mapeo, entre el bloque de RAM y el valor de cada bloque entregado por la CeNN, se realiza de la siguiente manera: se divide el vector de 64 bits en bloques más pequeños de 8 bits, obteniendo así 8 bloques de 8 bits. Estos 8 bits representan el valor k, es decir, la dirección del bloque RAM calculado previamente. Para ejemplificar esta propuesta observe la Tabla 5.1.

Tabla. 5.1. Ejemplo de mapeo entre los puntos de una curva elíptica y la representación como dirección de RAM del procesado de imagen con CeNN

	Dirección l	**Punto lG sober la curva**
Valor decimal	Valor binario	Punto $(l+2)G$
0	0000 0000	$P_1 = 2G$
1	0000 0001	$P_2 = 3G$
⋮	⋮	⋮
255	1111 1111	$P_{255} = 257G$

Teniendo el mapeo de puntos se puede utilizar cualquier método de cifrado visto en el Capítulo 4.2.2. Como veremos existe la posibilidad de hacer un criptoanálisis de este método.

5.4.2. Cifrado de imágenes. Método 2: Utilizando ECIES

El *Esquema de Cifrado Integrado de Curva Elíptica* (ECIES de *Elliptic Curve Integrated Encryption Scheme*) es un estándar de cifrado que utiliza curvas elípticas. Este está estandarizado en ANSI X9.63, ISO/IEC 15946-3, IEEE 1363a y SECG SEC-1. Este esquema está basado en el esquema de cifrado ElGamal. Este método es considerado un sistema de cifrado híbrido debido a que utiliza ECC para el establecimiento de las claves y como método de cifrado se utiliza un esquema de cifrado simétrico, como lo es AES.

Para utilizar el esquema ECIES se requiere conocer la clave publica Q. Además, se tiene la ventaja de no requerir que el mensaje a cifrar sea codificado o mapeado como un punto dentro de la curva elíptica. En el Algoritmo 5.1 se muestra el método de cifrado utilizando este esquema, y en el Algoritmo 5.2 se muestra el método de descifrado.

Algoritmo 5.1 Cifrado ECIES

Entrada: Los parámetros de la curva $D = (q, FR, S, a, b, P, n, h)$, la clave pública Q, y el mensaje m.
Salida: Mensaje cifrado (R, C, t)

1: Seleccionar un valor aleatorio $r \in [1, n]$ y calcular $R = rP$
2: Calcular $Z = hkQ = (Z_x, Z_y)$ con $Z \neq \mathcal{O}$
3: Calcular $(k_1, k_2) \leftarrow KFD(Z_x, R)$
4: Cifrar los datos: $C = Enc_{k_1}(m)$ y $t = MAC_{k_2}(C)$
5: **devolver** (R, C, t)

Algoritmo 5.2 Descifrado ECIES

Entrada: Los parámetros de la curva $D = (q, FR, S, a, b, P, n, h)$, la clave privada d, y el texto cifrado (R, C, t).
Salida: El texto original m o rechazo del texto cifrado.

1: Validar la clave pública R, rechazar el texto cifrado si la validación fue fallida.
2: Calcular $Z = hdR = (Z_x, Z_y)$. Si $Z = \mathcal{O}$, entonces rechazar el texto cifrado.
3: Calcular $(k_1, k_2) \leftarrow KDF((Z_x mR))$.
4: Calcular $t' = MAC_{k_2}(C)$. Si $t' \neq t$, entonces rechazar el texto cifrado.
5: Calcular $m = DEC_{k_1}(C)$.
6: **devolver** m

Como se observa en los algoritmos anteriores se requiere del cálculo de las funciones KDF, MAC, ENC y DEC. Las funciones ENC y DEC, como fue mencionado

anteriormente, son funciones de cifrado simétrico. KFD (de *Key Derivation Function*) es una función que deriva claves a partir de una clave principal, para construir esta función se puede implementar por ejemplo las funciones HASH. La función MAC (de Message Authentication Code) es un método para autenticar los mensajes, se puede utilizar por ejemplo HMAC.

5.5. Implementación del Cifrado de Imágenes utilizando RSA y ECC

Se ha implementado el cifrado de imágenes tanto para RSA como para ECC, con lo cual podemos hacer una comparativa entre ambos métodos. Para este propósito se ha utilizado las implementaciones realizadas en el Capítulo 2.5 para la CeNN, Capítulo 3.4 para la implementación del cifrador RSA y en el Capítulo 4.4 para la implementación del cifrador ECC.

En la Fig. 5.3 se muestra el controlador general para el cifrado de la imagen, utilizando los conceptos anteriormente mencionados.

5.5.1. Implementación usando RSA

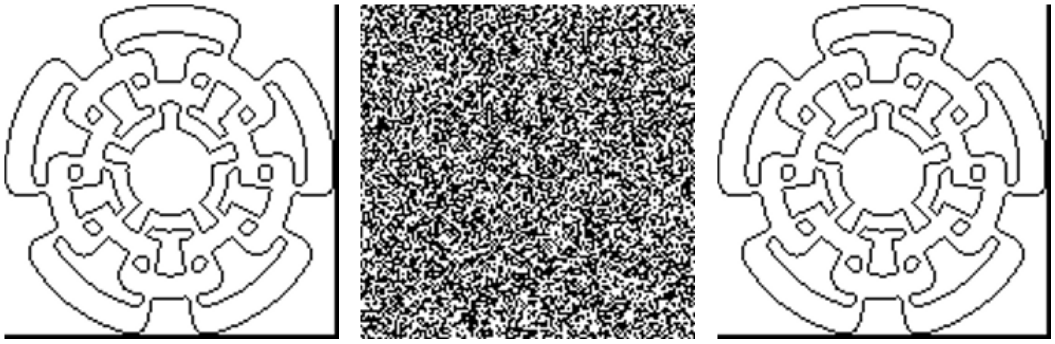
Los recursos utilizados para esta implementación se pueden observar en la Tabla 5.2 en dicha tabla se observan los diferentes componentes y sus recursos usados dentro del FPGA.

Para comprobar el funcionamiento correcto del cifrador RSA utilizando CeNN se ha procesado una imagen de prueba, la cual es una imagen procesada con la CeNN. Dicha imagen es de tamaño 136×136 pixeles, con dicha imagen se han obtenido 27 mensajes a cifrar de tamaño 960 bits. Los resultados obtenidos se pueden observar en la Fig. 5.5.

Para el descifrado de la imagen se puede utilizar la misma implementación. Para comprobar el resultado se ha utilizado Wolfram Mathematica®. Este mismo software

Tabla. 5.2. Recursos usados por el FPGA para el cifrado usando RSA para el procesamiento de imágenes.

	LUTs (63,400)	**Registers** (126,800)	**Slice** (15,850)	**BRAM** (135)	**DSP** (240)
CeNN	17,601	8,748	5,660	32	64
▷ Cell	215	114	111	0	1
Codificador	0	64	35	0	0
M SaMAL2R	10,803	10,557	4,548	0	66
▷ Multiplicación	10,429	5,353	3,904	0	66



(a) Imagen original **(b)** Imagen cifrada **(c)** Imagen descifrada

Fig. 5.5. Resultados del cifrado de imágenes usando RSA.

se ha utilizado para generar las claves utilizadas.

Durante el cifrado de las imágenes no se volvió a realizar SCA, específicamente SPA; esto debido a que en capítulos anteriores ya se ha realizado este estudio.

5.5.2. Implementación usando ECC

La implementación para nuestro caso de estudio se basa en el Cifrador de imágenes con base al Método 1. Permitiendo así obtener el Algoritmo 5.3 como método de cifrado.

Algoritmo 5.3 Propuesta para el cifrado de imágenes usando ECC B-163.

Entrada: Los parámetros de la curva, la clave pública $P = (x, y) \in E$, la clave privada G, el valor del bloque k de la CeNN, donde $k = (j_7, \cdots, k_0)$.
Salida: $Q = MG$.

1: Crear el vector $M = (M_1, \cdots, M_8)$.
2: **para** $i = 1$ hasta 8 **hacer**
3: Calcular $M_i \leftarrow k_i \cdot G$
4: **fin para**
5: **devolver** (M)

Donde k_i representa el vector codificado utilizando el Método 1, obteniéndose así un vector de 8 puntos en la curva elíptica, los cuales representan el cifrado del bloque dicha CeNN. El algoritmo propuesto permite realizar un esquema comparativo más adecuado con RSA, sin embargo, esto se discutirá más adelante.

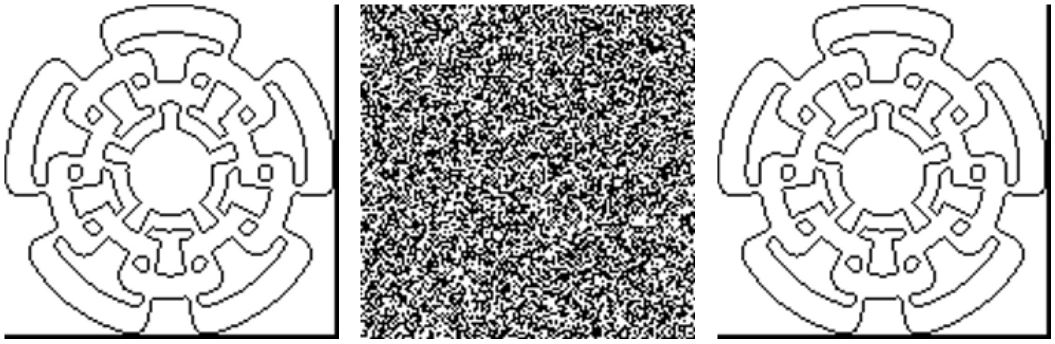
Los recursos utilizados de la implementación del cifrador ECC para imágenes procesadas con CeNN se muestran en la Tabla 5.3.

Para comprobar el correcto funcionamiento del sistema propuesto, se ha decido tomar la misma imagen que se utilizó en el subapartado anterior, la cual es una imagen procesada con la CeNN. Debido a que la imagen es de tamaño 136×136 pixeles, con lo cual obtenemos 2,312 mensajes codificados, estos se tienen que mapear a puntos de la curva elíptica según el método propuesto. En la Fig. 5.6 se muestran los resultados obtenidos.

Al igual que en caso de RSA, la implementación puede ser utilizada para descifrar

Tabla. 5.3. Recursos usados por el FPGA para el cifrado usando ECC para el procesamiento de imágenes.

	LUTs (63,400)	**Registers** (126,800)	**Slice** (15,850)	**BRAM** (135)	**DSP** (240)
CeNN	17,608	8,750	5,520	32	64
▷ Cell	215	114	111	0	1
Codificador	0	64	35	0	0
ECC B-163	2,500	2,393	967	0	66
▷ Multiplicación	777	362	359	0	66



(a) Imagen original **(b)** Imagen cifrada **(c)** Imagen descifrada

Fig. 5.6. Cifrado de imágenes usando ECC B-163.

las imágenes. Para comprobar los valores obtenidos por el cifrador se utilizado Wolfram Mathematica®. Además, con este mismo software se han generado las claves, al igual que en RSA.0

5.6. Comparativa

En la Tabla 5.4 se observa una comparativa de los recursos utilizados por cada una de las implementaciones realizadas. Se puede observar tanto los recursos utilizados por la implementación completa, como los componentes individuales.

Tabla. 5.4. Comparativa del cifrador de imágenes procesadas con la CeNN.

Implementación		**Slices** (15,850)	**BRAM** (135)	**DSP** (240)	**Tiempo** (*ns*)
RSA 1024	CeNN	5,741	32	64	-
	Cifrador	4,437	0	66	3,991,130
	Total	10,888	32	130	-
ECC B-163	CeNN	5,520	32	64	-
	Cifrador	967	0	0	7,742,000
	Total	7,958	32	64	0

Como podemos observar en la Tabla 5.4, la implementación realizada con curva elíptica requiere un menor consumo de recursos; sin embargo, como vemos, el tiempo de cifrado es mayor al requerido por el cifrador RSA.

El tiempo de procesamiento con la CeNN y su cifrado de una imagen dependerá del tamaño y del tipo de procesamiento de ésta. Sin embargo, podemos realizar un cálculo del cifrado utilizando como ejemplo la imagen mostrada en los ejemplos anteriores.

Primero se considera el tamaño de la imagen, el cual, en nuestro caso de estudio, es de 136×136 pixeles. Para el cifrador utilizando RSA de 1024 bits, tenemos: 20 mensajes de 960 bits cada uno, por lo cual requerimos para el cifrado del total de la imagen de $79.8226ms$.

Para el cifrador utilizando ECC B-163 tenemos un total de 2,312 bloques. Consi-

derando que se toma en cuenta el método 1, donde se calcula la multiplicación escalar por cada sub-bloque de 8 bits de la CeNN como procesador de imágenes, para el cifrado con curva elíptica, se tiene que la multiplicación escalar tiene una duración de $434\mu s$, esto da como total un tiempo de cifrado de $1.003408s$.

5.7. Conclusiones

Como se ha observado en este capítulo, se ha propuesto una metodología para el cifrado de imágenes. Esta metodología usa tanto RSA como ECC, en ambos sistemas propuestos se ve claramente que cumplen con su función de cifrado.

Como vimos al inicio de este capítulo, existen diversos métodos para el cifrado de imágenes, sin embargo, la gran mayoría de estos trabajos no tratan del cifrado desde el punto de vista de hardware.

En este capítulo pudimos observar una propuesta para el cifrado de las imágenes. El cual puede ser utilizado tanto para cifradores utilizando RSA como ECC, e inclusive para otro tipo de cifradores, como por ejemplo AES o 3DES; esto permite utilizar esta propuesta de forma modular.

Como hemos observado, para el cifrado, no importando que tipo de dato se va a cifrar, no se puede realizar directamente utilizando ECC; esto debido a la dificultad de representar el mensaje en claro como un punto o puntos dentro de la curva elíptica.

Aunque en este capítulo se ha establecido una propuesta de cifrado de imágenes, este puede tener inconvenientes; sin embargo, esto será discutido en la sección de Conclusiones y Discusión.

Conclusiones

Como hemos visto a través de este trabajo el realizar la transmisión segura de imágenes cifradas por medios no seguros, no es una tarea sencilla.

El cifrar una imagen completa, en especial imágenes a color, conlleva mucho tiempo de procesamiento; esto debido a la cantidad de bits requeridos para contener su información y del tamaño de clave privada o pública. La CeNN, como hemos visto en este trabajo, puede ser utilizada para el procesamiento de imágenes. Dicho procesamiento, entonces, puede reducir la cantidad de datos a ser cifrados y poder así ser transmitidos en un menor tiempo. Como fue mostrado, se propuso una técnica de utilizar los datos entregados por la CeNN para su cifrado, reduciendo la cantidad de estos datos, lo cual reduce los tiempos de cifrado y/o descifrado de las imágenes.

Para el cifrado de datos, se estudiaron los esquemas de cifrado RSA y ECC. Además, se propusieron métodos para que estos, al ser implementados en dispositivos programables (como los FPGAs) no puedan ser obtenidas las claves secretas a través de mediciones físicas. Se realizaron SCAs, en especial los SPA, sobre las implementaciones realizadas mostrando que dichas implementaciones eran seguras.

Se propuso un esquema para cifrar los datos, en específico los de imágenes, el cual es seguro, para los casos de los ataques estudiados en este trabajo. Esta propuesta fue implementada con cifradores RSA y ECC. Sin embargo, como se observó en el apartado de Resultados, es necesario realizar optimizaciones sobre la implementación ECC, con la finalidad de reducir los tiempos de cifrado de la información.

Discusión

La metodología para el cifrado de imágenes mencionada anteriormente permite mostrar que el tiempo de procesamiento con curva elíptica es mucho mayor que con RSA. Sin embargo, como fue mencionado anteriormente la cantidad de recursos es mucho menor. Para aumentar la velocidad de cifrado de dicha imagen se puede crear una memoria la cual contendrá los valores de la curva elíptica codificada según el valor de la CeNN, como es mostrado en la Tabla 5.1. Esto reduciría la cantidad de bloques a solo 256, con lo cual tendríamos un tiempo de $111.104ms$; por otra parte, el realizar esto aumentaría la cantidad de recursos necesarios por el FPGA para dicha implementación, ya que se requerirían 512 registros con un ancho de palabra de 163 bits para contener los puntos de la curva elíptica.

Otra metodología de reducir el tiempo de cifrado de la imagen utilizando ECC, es utilizando el algoritmo ECIES mostrado en el Algoritmo 5.1. Sin embargo, se debe considerar que se requiere un segundo método de cifrado como AES, además, de la creación de funciones extras.

Además, la implementación realizada tiene como base las operaciones sobre campos finitos, específicamente, se optó por no implementar las operaciones de elevar al cuadrado y reducción. Si se implementan las operaciones de elevar al cuadrado y reducción, se reduce significativamente el tiempo de procesamiento; sin embargo, por otra parte, aumenta la cantidad de recursos utilizados para su implementación dentro del FPGA.

Una desventaja, con respecto a la propuesta realizada en este trabajo sobre el cifrado de imágenes utilizando ECC, es que para imágenes grandes se puede realizar

un análisis de los datos cifrados, como, por ejemplo, medir la cantidad promedio de bloques de pixeles y utilizando un método probabilístico obtener la imagen. Como se observa en la metodología se tienen en total únicamente 256 datos cifrados, los cuales pueden ser visualizados fácilmente.

Trabajo a futuro

Como trabajo a futuro se propone realizar la optimización, tanto para reducir el consumo de recursos como para aumentar la velocidad de operación de la multiplicación escalar. Específicamente realizar la optimización sobre la aritmética en campos finitos.

Una segunda propuesta es utilizar la CeNN como generador de números aleatorios, específicamente para establecer los números aleatorios que se requieren en criptografía. Además, este serviría para poder implementar el Algoritmo 4.5.

Una tercera propuesta es realizar DPA sobre las propuestas e implementaciones realizadas en este trabajo; a partir de este punto se puede realizar, entonces, un estudio más detallado. Aunado a lo anterior, sería interesante realizar ataques electromagnéticos, y conocer el comportamiento de los FPGA y su resistencia a dichos ataques, específicamente sobre criptografía implementada sobre estos dispositivos.

Como fue mencionado anteriormente, las imágenes cifradas con los métodos propuestos en este trabajo pueden ser atacados con un análisis sobre los puntos obtenidos, ya que serían entonces pocos los cuales se tendrían. Para evitar este tipo de criptoanálisis, se propone dividir la imagen a ser cifrada en bloques más pequeños, además, de generar una clave privada para cada bloque de imagen utilizando el intercambio de claves Diffie-Hellman utilizando curvas elípticas. Para lo cual, se puede utilizar como base las implementaciones presentadas en este trabajo, además, de realizar el estudio correspondiente para asegurar su fortaleza contra diferentes ataques.

Finalmente, se propone realizar un estudio de las implementaciones RSA y ECC realizadas, utilizando diferentes frecuencias de reloj, debido a que la captura de las se-

ñales puede ser diferente utilizando frecuencias bajas, como, por ejemplo, con valores menores a 20MHz, como las manejadas por las tarjetas inteligentes (muy utilizadas actualmente).

Apéndice A

Códigos

Los códigos realizados en esta tesis pueden ser solicitados al grupo de VLSI, de la Sección de Electrónica del Estado Sólido, del Departamente de Ingeniería Eléctrica, del Centro de Investigación y de Estudios Avanzados del I.P.N., unidad Zacatenco o pueden ser descargados de los siguientes enalces:

Códigos de la Tesis

`https://github.com/JdJMR/Tesis_Doctorado`

Bibliografía

[1] L. O. Chua and L. Yang, "Cellular neural networks: Theory and applications," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257–1290, 1988.

[2] A. J. Menezes, J. Katz, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications, CRC Press, 1996.

[3] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, Inc., 2nd ed., 2015.

[4] M. H. T. Hagan, B. Demuth and O. de Jesus, *Neural Network Design*. 2nd ed., 2014.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[7] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," pp. 1237–1242, 07 2011.

[8] L. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*. Cambridge University Press, 2002.

[9] M. Csapodi, J. Vandewalle, and T. Roska, "High speed calculation of cryptographic hash functions by cnn chips," in *1998 Fifth IEEE International Workshop on Cellular Neural Networks and their Applications. Proceedings (Cat. No.98TH8359)*, pp. 186–191, 1998.

[10] G. Csaba, X. Ju, Z. Ma, Q. Chen, W. Porod, J. Schmidhuber, U. Schlichtmann, P. Lugli, and U. Rührmair, "Application of mismatched cellular nonlinear networks for physical cryptography," in *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, pp. 1–6, 2010.

[11] G. Hu, W. Kou, J. Dong, and J. Peng, "A novel image encryption algorithm based on cellular neural networks hyper chaotic system," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1878–1882, 2018.

[12] J. Lin, Y. Luo, J. Liu, J. Bi, S. Qiu, M. Cen, and Z. Liao, "An image compression-encryption algorithm based on cellular neural network and compressive sensing," in *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, pp. 673–677, 2018.

[13] R. Forgáč and M. Očkay, "Contribution to symmetric cryptography by convolutional neural networks," in *2019 Communication and Information Technologies (KIT)*, pp. 1–6, 2019.

[14] J. Roh, S. Cho, and S. Jin, "Learning based biometric key generation method using cnn and rnn," in *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 136–139, 2018.

[15] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology — CRYPTO '96* (N. Koblitz, ed.), (Berlin, Heidelberg), pp. 104–113, Springer Berlin Heidelberg, 1996.

[16] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), (Berlin, Heidelberg), pp. 388–397, Springer Berlin Heidelberg, 1999.

[17] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology — EUROCRYPT '97* (W. Fumy, ed.), (Berlin, Heidelberg), pp. 37–51, Springer Berlin Heidelberg, 1997.

[18] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security* (I. Attali and T. Jensen, eds.), (Berlin, Heidelberg), pp. 200–210, Springer Berlin Heidelberg, 2001.

[19] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing.," *IACR Cryptology ePrint Archive*, vol. 2005, p. 388, Jan 2005.

[20] G. B. Ratanpal, R. D. Williams, and T. N. Blalock, "An on-chip signal suppression countermeasure to power analysis attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 179–189, 2004.

[21] O. Mirmotahari and Y. Berg, "Low voltage design against power analysis attacks," in *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, pp. 545–548, 2008.

[22] H. Saini and A. Gupta, "Constant power consumption design of novel differential logic gate for immunity against differential power analysis," *IET Circuits, Devices Systems*, vol. 13, no. 1, pp. 103–109, 2019.

[23] V. Sundaresan, S. Rammohan, and R. Vemuri, "Defense against side-channel power analysis attacks on microelectronic systems," in *2008 IEEE National Aerospace and Electronics Conference*, pp. 144–150, 2008.

[24] M. Stöttinger, S. Malipatlolla, and Q. Tian, "Survey of methods to improve side-channel resistance on partial reconfigurable platforms," in *Design Methodologies*

for Secure Embedded Systems (A. Biedermann and H. G. Molter, eds.), (Berlin, Heidelberg), pp. 63–84, Springer Berlin Heidelberg, 2011.

[25] M. Juneja and s. Nagar, "Particle swarm optimization algorithm and its parameters: A review," pp. 1–5, 10 2016.

[26] A. Giaquinto and G. Fornarelli, "Pso-based cloning template design for cnn associative memories," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 20, pp. 1837–41, 09 2009.

[27] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.

[28] T. Kozek, T. Roska, and L. O. Chua, "Genetic algorithm for cnn template learning," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 6, pp. 392–402, 1993.

[29] D. Karaboga, "An idea based on honey bee swarm for numerical optimization, technical report - tr06," *Technical Report, Erciyes University*, 01 2005.

[30] S. Parmaksizoglu and M. Alçı, "A novel cloning template designing method by using an artificial bee colony algorithm for edge detection of cnn based imaging sensors," *Sensors (Basel, Switzerland)*, vol. 11, pp. 5337–59, 12 2011.

[31] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, pp. 308–313, 01 1965.

[32] K. Nakai and A. Ushida, "Design technique of cellular neural network," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 78, no. 3, pp. 97–107, 1995.

[33] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (abc) algorithm," *Applied Soft Computing*, vol. 8, pp. 687–697, 01 2008.

[34] M. Duraisamy and F. M. M. Jane, "Cellular neural network based medical image segmentation using artificial bee colony algorithm," in *2014 International*

Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), pp. 1–6, 2014.

[35] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[36] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, p. 120–126, Feb. 1978.

[37] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings* (H. C. Williams, ed.), (Berlin, Heidelberg), pp. 417–426, Springer Berlin Heidelberg, 1986.

[38] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp*, vol. 48, pp. 243–264, 01 1987.

[39] J. Daor, J. Daemen, and V. Rijmen, "Aes proposal: rijndael," 10 1999.

[40] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[41] L. Washington, *Elliptic Curves: Number Theory and Cryptography*. Discrete Mathematics and Its Applications, CRC Press, 2nd ed., 2008.

[42] D. T. Varela, *Algoritmos de Exponenciación Modular y su protección en sistemas criptográficos*. PhD thesis, Universidad Nacional Autónoma de México, Facultad de Estudios Superiores Cuautitlán, July 2014.

[43] P.-A. Fouque and F. Valette, "The doubling attack – why upwards is better than downwards," in *Cryptographic Hardware and Embedded Systems - CHES 2003* (C. D. Walter, Ç. K. Koç, and C. Paar, eds.), (Berlin, Heidelberg), pp. 269–280, Springer Berlin Heidelberg, 2003.

[44] S.-M. Yen, L.-C. Ko, S. Moon, and J. Ha, "Relative doubling attack against montgomery ladder," in *Information Security and Cryptology - ICISC 2005* (D. H.

Won and S. Kim, eds.), (Berlin, Heidelberg), pp. 117–128, Springer Berlin Heidelberg, 2006.

[45] S. M. Yen, W. C. Lien, S. Moon, and J. Ha, "Power analysis by exploiting chosen message and internal collisions – vulnerability of checking mechanism for rsa-decryption," in *Progress in Cryptology – Mycrypt 2005* (E. Dawson and S. Vaudenay, eds.), (Berlin, Heidelberg), pp. 183–195, Springer Berlin Heidelberg, 2005.

[46] Atsushi Miyamoto, Naofumi Homma, Takafumi Aoki, and Akashi Satoh, "Enhanced power analysis attack using chosen message against rsa hardware implementations," in *2008 IEEE International Symposium on Circuits and Systems*, pp. 3282–3285, 2008.

[47] J. S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Cryptographic Hardware and Embedded Systems* (Ç. K. Koç and C. Paar, eds.), (Berlin, Heidelberg), pp. 292–302, Springer Berlin Heidelberg, 1999.

[48] D. T. Varela, "How to avoid the n-1 attack without costly implementations," *International Journal of Network Security & Its Applications*, vol. 4, no. 4, p. 109, 2012.

[49] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology* (D. Chaum, R. L. Rivest, and A. T. Sherman, eds.), (Boston, MA), pp. 51–60, Springer US, 1983.

[50] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO' 86* (A. M. Odlyzko, ed.), (Berlin, Heidelberg), pp. 311–323, Springer Berlin Heidelberg, 1987.

[51] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[52] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 193–199, 1995.

[53] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[54] G. Perin, D. Gomes Mesquita, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: Systolic versus multiplexed implementation," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[55] P. Wang, Z. Liu, L. Wang, and N. Gao, "High radix montgomery modular multiplier on modern fpga," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1484–1489, 2013.

[56] W. Dai, D. Chen, R. C. Cheung, and C. K. Koc, "Fft-based mclaughlin's montgomery exponentiation without conditional selections," *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1301–1314, 2018.

[57] K. Venkatesh, K. Pratibha, S. Annadurai, and L. Kuppusamy, "Reconfigurable architecture to speed-up modular exponentiation," in *2019 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–6, 2019.

[58] S. Mondal and S. Patkar, "Hardware-software co-implementation of a high performance and light-weight scalable systolic-montgomery based modified rsa for portable iot devices," in *Proceedings of the 2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pp. 439–443, 2021.

[59] R. Abarzúa, C. Valencia, and J. López, "Survey on performance and security problems of countermeasures for passive side-channel attacks on ecc," *Journal of Cryptographic Engineering*, vol. 11, no. 1, pp. 71–102, 2021.

[60] E. Barker, L. Chen, S. Keller, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography," tech. rep., National Institute of Standards and Technology, 2017.

[61] D. R. Brown, "Sec 2: Recommended elliptic curve domain parameters," *Standars for Efficient Cryptography*, 2010.

[62] M. Joye, "Highly regular right-to-left algorithms for scalar multiplication," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 135–147, Springer, 2007.

[63] E. Brier and M. Joye, "Weierstraß elliptic curves and side-channel attacks," in *International workshop on public key cryptography*, pp. 335–345, Springer, 2002.

[64] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," tech. rep., 2017.

[65] J. Ha, J. Park, S. Moon, and S. Yen, "Provably secure countermeasure resistant to several types of power attack for ecc," in *International Workshop on Information Security Applications*, pp. 333–344, Springer, 2007.

[66] M. Roy, S. Chakraborty, K. Mali, S. Mitra, I. Mondal, R. Dawn, D. Das, and S. Chatterjee, "A dual layer image encryption using polymerase chain reaction amplification and dna encryption," in *2019 international conference on opto-electronics and applied optics (Optronix)*, pp. 1–4, IEEE, 2019.

[67] J. K. Saini and H. K. Verma, "A hybrid approach for image security by combining encryption and steganography," in *2013 IEEE second international conference on image information processing (ICIIP-2013)*, pp. 607–611, IEEE, 2013.

[68] F. Jie, P. Ping, G. Zeyu, and M. Yingchi, "A meaningful visually secure image encryption scheme," in *2019 IEEE fifth international conference on big data computing service and applications (BigDataService)*, pp. 199–204, IEEE, 2019.

[69] F. S. Hasan and M. A. Saffo, "Fpga hardware co-simulation of image encryption using stream cipher based on chaotic maps," *Sensing and Imaging*, vol. 21, no. 1, pp. 1–22, 2020.

[70] N. K. Pareek, V. Patidar, and K. K. Sud, "Diffusion–substitution based gray image encryption scheme," *Digital signal processing*, vol. 23, no. 3, pp. 894–901, 2013.

[71] O. Reyad, “Text message encoding based on elliptic curve cryptography and a mapping methodology,” *Information Sciences Letters*, vol. 7, no. 1, p. 2, 2018.