

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

Programa de Maestría y Doctorado en Ciencias e Ingeniería



CRIPTOGRAFÍA CAÓTICA EN SISTEMAS EMBEBIDOS DE ALTO
DESEMPEÑO

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

DOCTOR EN CIENCIAS

presenta:

ABRAHAM FLORES VERGARA

Director de tesis

Dr. Enrique Efrén García Guerrero

Ensenada, Baja California, México. Junio del 2018.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

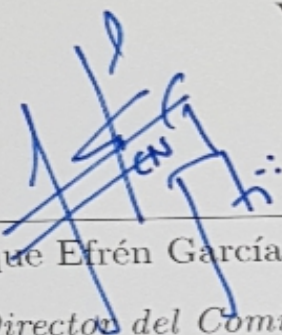
CRIPTOGRAFÍA CAÓTICA EN SISTEMAS EMBEBIDOS DE ALTO DESEMPEÑO

TESIS

que para obtener el grado de DOCTOR en CIENCIAS presenta:

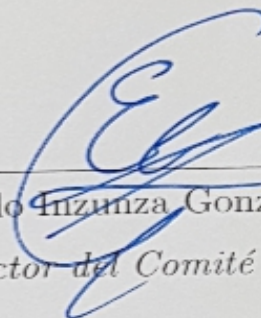
ABRAHAM FLORES VERGARA

Y aprobada por el siguiente comité:



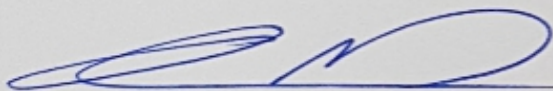
Dr. Enrique Efrén García Guerrero

Director del Comité



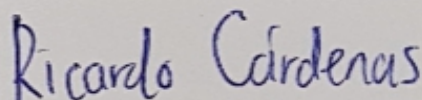
Dr. Everardo Inzunza González

Codirector del Comité



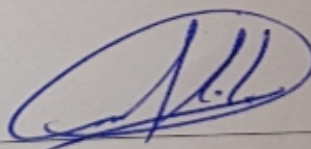
Dr. Oscar Roberto López Bonilla

Miembro del Comité



Dr. José Ricardo Cárdenas Valdez

Miembro de Comité



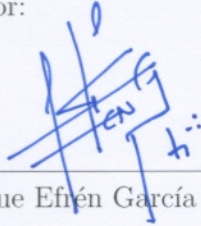
Dr. Esteban Tlelo Cuautle

Miembro del Comité

RESUMEN de la tesis de **Abraham Flores Vergara**, presentada como requisito parcial para obtener el grado de DOCTOR EN CIENCIAS, del programa de Maestría y Doctorado en Ciencias e Ingeniería de la UABC. Ensenada, B. C. México, Junio del 2018.

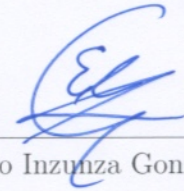
CRIPTOGRAFÍA CAÓTICA EN SISTEMAS EMBEBIDOS DE ALTO DESEMPEÑO

Resumen aprobado por:



Dr. Enrique Efrén García Guerrero

Director de tesis



Dr. Everardo Inzunza González

Codirector de tesis

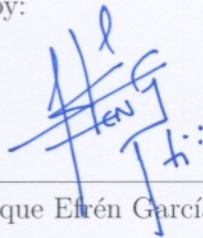
En este trabajo de tesis doctoral, se presenta un nuevo sistema criptográfico para el encriptado de imágenes digitales de muy alta resolución, basado en el diseño de un generador digital de caos utilizando aritmética de precisión arbitraria. Lo anterior, como una alternativa para reducir la degradación dinámica que presentan los modelos caóticos cuando son implementados en dispositivos digitales y para incrementar la seguridad del sistema criptográfico. Los resultados obtenidos, muestran que al utilizar aritmética de gran precisión, el generador de secuencias cifrantes proporciona altos niveles de aleatoriedad (muy alta seguridad) durante un mayor número de iteraciones del mapeo caótico implementado, en comparación con las secuencias que se generan al utilizar aritmética de punto flotante IEEE 754 con precisión simple o doble precisión. El método propuesto, no requiere de hardware de alto costo para el incremento de la precisión numérica. Como una ventaja frente a otros trabajos recientes, el uso de la aritmética de precisión arbitraria, en relación con los métodos que usan precisión simple o doble precisión, proporciona un aumento exponencial en el espacio clave. Se demuestra que usando la aritmética de precisión múltiple, se puede obtener un espacio de clave de $2^{33,268}$ o superior, dependiendo del nivel de gran precisión configurado. El análisis de seguridad confirma que el sistema criptográfico propuesto, es altamente seguro y robusto contra diferentes ataques bien conocidos en la literatura. Adicionalmente, se presenta un método con técnicas de procesamiento paralelo, con el propósito de optimizar los tiempos en el proceso de encriptado y desencriptado de imágenes de muy alta resolución. El método propuesto es potencialmente viable de implementarse en dispositivos modernos de telecomunicaciones, tal como un teléfono celular inteligente y en otros dispositivos de redes de telecomunicaciones, por ejemplo en sistemas de transmisión de TV o internet por satélite.

Palabras clave: Degradación digital, procesamiento en paralelo, aritmética de precisión arbitraria, criptografía caótica, sistema embebido.

ABSTRACT of the thesis of **ABRAHAM FLORES VERGARA**, presented as a partial requirement to obtain the degree of PhD in SCIENCES, of the program of MSc and PhD in Sciences and Engineering of UABC. Ensenada, B. C., Mexico, June 2018.

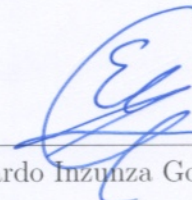
CHAOTIC CRYPTOGRAPHY IN HIGH PERFORMANCE EMBEDDED SYSTEMS

Abstract approved by:



Dr. Enrique Elrén García Guerrero

Thesis supervisor



Dr. Everardo Inzunza González

Thesis supervisor

In this doctoral thesis, a new chaotic cryptosystem for the encryption of very high-resolution digital images is proposed, based on the design of a digital chaos generator by using arbitrary precision arithmetic. The former one, as an alternative to reduce the dynamic degradation that chaotic models present when are implemented in digital devices and to increase the security of the cryptosystems. The results obtained show that when using high precision arithmetic, the generated sequences provide a high level of randomness (i.e. very high security) during a greater number of iterations of the chaotic maps implemented in comparison with the sequences generated by using the standard of simple precision or double precision according to the IEEE 754 Standard for Floating-Point Arithmetic. The proposed method does not require high-cost hardware for increasing numerical accuracy. As an advantage versus other recent works, using high precision, in relation to the methods that use simple precision or double precision, awards an exponential increase in the key space. It is demonstrated that using multiple-precision arithmetic, key space of $2^{33,268}$ or higher can be obtained, depending on the level of high precision configured. The security analysis confirms that the proposed cryptosystem is highly secure and robust against several known attacks. Additionally, a method with parallel processing techniques is presented in order to optimize the times in the process of encryption and decryption of very high-resolution images. The proposed method is potentially feasible to be implemented in modern telecommunications devices, such as a smart cell phone and in other telecommunications network devices, for example in TV/internet systems transmission by satellite.

Keywords: Digital degradation, parallel processing, arbitrary precision arithmetic, chaotic cryptography, embedded system.

A mi familia.

Agradecimientos

*A **DIOS**, por tenerme siempre bajo su protección y por todas las bendiciones que me ha dado todos los días de mi vida.*

*Al **Dr. Enrique Efrén García Guerrero**, por haberme dirigido en la realización del presente trabajo de tesis, por su amistad, la atención y el apoyo incondicional que me ha brindado.*

*Al **Dr. Everardo Inzunza González**, por su amistad, atención y por sus enseñanzas que me permitieron desarrollar con éxito el presente trabajo de tesis.*

*Al **Dr. Oscar Roberto López Bonilla**, un especial agradecimiento por sus comentarios muy acertados en las presentaciones y revisiones del presente trabajo de tesis de doctorado.*

*Al **Dr. José Ricardo Cárdenas Valdez**, un especial agradecimiento por su colaboración como integrante del comité para el presente trabajo de tesis de doctorado.*

*Al **Dr. Esteban Tlelo Cuautle**, un especial agradecimiento por su colaboración como integrante del comité para el presente trabajo de tesis de doctorado.*

*Al **CONACyT**, que con su programa de becas me proporcionaron los recursos necesarios durante esta etapa importante en mi vida profesional.*

*A nuestra alma mater: **Universidad Autónoma de Baja California**, que es como mi segundo hogar y donde se me ha permitido desarrollarme profesionalmente.*

*Al **coordinador de posgrado**, **Dr. Miguel Enrique Martínez Rosas**, por sus asesorías administrativas.*

Ensenada, B. C. México.
Junio de 2018.

ABRAHAM FLORES VERGARA

Tabla de Contenido

	Página
Tabla de Contenido	i
Resumen	iii
Abstract	iv
Agradecimientos	vi
Lista de figuras	ix
Lista de tablas	xi
I Introducción	1
I.1 Planteamiento del problema	4
I.1.1 Preguntas de investigación	4
I.1.2 Delimitación del problema	5
I.2 Objetivos del proyecto	6
I.2.1 Objetivo general	6
I.2.2 Objetivos específicos	6
I.3 Organización de la tesis	7
II Generador de caos de gran precisión (HPG)	8
II.1 Introducción	8
II.2 Metodología	9
II.2.1 Programa para generar caos de gran precisión	11
II.2.2 Condiciones iniciales y parámetros de control	15
II.2.3 Generador de condiciones iniciales pseudoaleatorias	16
II.2.4 Generador de serie de bits pseudoaleatorios	18
II.3 Conclusiones	23
III Encriptado y desencriptado de imágenes a color de alta resolución.	24
III.1 Introducción.	24
III.1.1 Seguridad criptográfica y el secreto perfecto de Shannon	25
III.1.2 Espacio de claves y ataque de fuerza bruta	28
III.2 Metodología.	30
III.2.1 Método de cifrado en flujo con clave simétrica basado en HPG propuesto.	30
III.2.2 Conformación de la clave de encriptado.	32
III.3 Conclusiones	35
IV Sistema criptográfico de imágenes a color con procesamiento en pa- ralelo	36
IV.0.1 Perfilación de programas en Python	36
IV.0.2 Parámetros de procesamiento en paralelo	42
IV.1 Método propuesto con procesamiento en paralelo	47

Tabla de Contenido (Continuación)

	Página
IV.1.1 Implementación de MPI utilizando comunicación colectiva en programa "Encriptado.py"	49
IV.1.2 Máquinas de procesadores Raspberry Pi 3 y Clúster FIAD . .	52
IV.2 Conclusiones	56
V Resultados y validación.	58
V.1 Resultados de Generador de caos de gran precisión (HPG)	58
V.1.1 Análisis estadístico con NIST Five-Test 140 – 2	58
V.1.2 Análisis estadístico con NIST 800 – 22 Rev 1a	61
V.2 Resultados de encriptado y desencriptado de imágenes de alta resolución	65
V.2.1 Espacio de claves.	65
V.2.2 Encriptado de imágenes RGB de alta resolución.	66
V.2.3 Prueba XOR en criptogramas con clave semejante.	70
V.2.4 Desencriptado de imagen RGB	71
V.2.5 Pruebas contra ataques diferenciales NPCR y UACI	71
V.2.6 Pruebas de criptogramas con entropía de información	72
V.3 Resultados del sistema criptográfico de imágenes a color con procesa- miento en paralelo	74
V.3.1 Resultados de implementación en máquina Raspberry Pi 3. .	74
V.3.2 Resultados de implementación en máquina Clúster FIAD . .	75
VI Conclusiones generales.	78
VI.1 Trabajos a futuro	80
Bibliografía	81

Lista de figuras

Figura	Página
1 Efecto de degradación digital del caos en el encriptado de imágenes de gran resolución.	10
2 Generador de caos de gran precisión (HPG) propuesto.	11
3 Diagrama de flujo: Generador de series caóticas con el HPG propuesto.	13
4 Trayectorias caóticas. Condiciones iniciales aplicando un Δx a x_0	15
5 Trayectorias caóticas con condiciones iniciales generadas pseudoaleatoriamente implementando HPG propuesto.	17
6 Diagrama de bloques del generador de serie pseudoaleatoria de bits mediante HPG.	18
7 Diagrama de flujo de la subrutina Convertidor a secuencia binaria. . . .	22
8 Comunicación segura mediante criptografía.	24
9 Método propuesto de encriptamiento de imágenes con criptografía caótica.	31
10 Método de cifrado en flujo con operador XOR.	31
11 Efecto de utilizar la misma serie cifrante en el encriptado de dos imágenes diferentes.	33
12 Composición de la clave de cifrado utilizando gran precisión numérica. .	34
13 Imagen y criptograma obtenido al utilizar HPG propuesto con el mapeo de Tinkerbell con ambos estados caóticos en el método de encriptado. .	41
14 Curva característica de la eficiencia del sistema $E(n)$ del programa "Encriptado.py" con $\alpha = 0.3\%$	45
15 Curvas del factor de rendimiento S_n para diferentes valores α en comparación con la curva de S_n para el programa "Encriptado.py" con $\alpha = 0.3\%$.	46
16 Diagrama de bloques de método de encriptado de imágenes con procesamiento paralelo.	48
17 Máquina Raspberry Pi 3.	53
18 Recurso computacional asignado a máquina virtual Raspberry Pi 3 con $n = 1$ procesador.	53
19 Recurso computacional asignado a máquina virtual Raspberry Pi 3: a) con $n = 4$ procesadores y b) con $n = 20$ procesadores.	55
20 Máquina Clúster FIAD empleando Raspberry Pi 3.	56
21 Gráfica de porcentajes de efectividad que proporcionan las series de bits con longitudes desde 1,000 bits y hasta 150,000 bits generadas con los diferentes niveles de precisión numérica.	62
22 a) Imagen "Mandrill.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbell con $np = 99$	67

Lista de figuras (Continuación)

Figura		Página
23	a) Imagen "Paisaje.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbelle con $np = 99$	68
24	a) Imagen "Loto.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbelle con $np = 99$	69
25	Prueba XOR en dos criptogramas diferentes obtenidos con clave simétrica semejante.	70
26	Imágenes recuperadas considerando la metodología propuesta; a) Recuperación del criptograma de "Mandrill.png", b) Recuperación del criptograma de "Paisaje.png" y c) Recuperación del criptograma de "Loto.png".	71
27	Gráfica de rendimiento del método criptográfico propuesto con procesamiento en paralelo en máquina Raspberry Pi 3.	75
28	Gráfica de rendimiento del método criptográfico propuesto con procesamiento en paralelo.	76
29	Resultado de encriptado con procesamiento en paralelo: a) Imagen de entrada, b) Criptograma y c) Imagen recuperada.	77
30	Ejemplos de recuperación de la imagen en el proceso de desencriptado con un error en el parámetro correspondiente al número de procesadores.	77

Lista de tablas

Tabla		Página
I	Longitud de la secuencia pseudoaleatoria de bits obtenida de x_n	20
II	Valores de entropía implementando HPG con x_{n+1} e y_{n+1} del mapeo de Tinkerbell, $np = 99$ y considerando la imagen "Paisaje.png".	40
III	Resultados de pruebas Five-Test considerando x_{n+1} del mapeo de Tinkerbell.	59
IV	Tabla comparativa de porcentajes de efectividad Five-Test de NIST considerando los estados de los mapeos caóticos analizados y $np = 99$	59
V	Tabla comparativa de porcentajes de efectividad Five-Test de NIST considerando los estados de los mapeos caóticos analizados y $np = 999$. . .	60
VI	Tabla comparativa de ejemplos de porcentajes de efectividad Five-Test con los diferentes niveles de precisión en series binarias de diferente longitud.	61
VII	Tabla de porcentajes de éxito del conjunto de pruebas NIST 800 – 22 realizado al generador propuesto (HPG) considerando el mapeo de Tinkerbell y cálculo numérico con precisión de $np = 99$	63
VIII	Tabla comparativa de porcentajes de éxito de los diferentes generadores de números aleatorios y pseudoaleatorios y el generador HPG propuesto con el estado x_{n+1} del mapeo de Tinkerbell.	64
IX	Tabla comparativa de porcentajes de éxito de los diferentes mapeos caóticos implementados en el generador HPG propuesto con $np = 99$. .	65
X	Tabla comparativa de espacios de clave del método propuesto con diferentes niveles de precisión.	66
XI	Tabla de resultados de la prueba NPCR a criptograma resultante implementando HPG mapeo de Tinkerbell x_{n+1} con $np = 99$ y la imagen "Paisaje.png".	72
XII	Tabla de resultados de la prueba UACI a criptograma resultante implementando HPG con mapeo de Tinkerbell x_{n+1} con $np = 99$ y la imagen Paisaje.png.	72
XIII	Valores de entropía de la información del criptograma resultante usando precisión simple, doble precisión y gran precisión con $np = 99$ en el método criptográfico propuesto considerando la imagen "Paisaje.png". .	73
XIV	Valores de entropía de la información del criptograma resultante considerando $np = 99$ y distintos mapeos caóticos en el generador HPG propuesto con la imagen "Paisaje.png".	73
XV	Tabla de resultados de los tiempos de ejecución del método criptográfico propuesto en máquina Raspberry Pi 3.	74

XVI	Tabla de resultados de los tiempos de ejecución del método criptográfico propuesto en máquina Clúster FIAD.	76
-----	--	----

Capítulo I

Introducción

Con el avance tecnológico en el desarrollo de sensores inteligentes, tecnologías de comunicación y protocolos de internet, crece el número de servicios y múltiples aplicaciones electrónicas modernas que llevan a cabo intercambio de información confidencial por canales públicos de telecomunicación; lo anterior, utilizando dispositivos electrónicos o sistemas embebidos con tecnología digital avanzada (Morabito *et al.*, 2018). Con la implementación del internet de las cosas, se espera que para el año 2020 se conecten a internet más de 50,000 millones de dispositivos con tecnología de comunicación digital en sistemas embebidos (Al-Fuqaha *et al.*, 2015; Ng, 2017). Si bien es cierto que en la actualidad existen métodos que transmiten y almacenan información de manera confiable, la tecnología se encuentra en permanente evolución y con el avance en las capacidades operacionales de cómputo, se siguen concibiendo nuevas amenazas y vulnerabilidades que comprometen la seguridad de la información en los sistemas de telecomunicaciones (Kocamaz *et al.*, 2018). Bajo esta perspectiva, la seguridad criptográfica forma parte integral e indivisible de este proceso evolutivo, por lo tanto, es necesario el desarrollo constante de nuevos métodos y técnicas de protección de información confidencial, que garanticen una mayor seguridad criptográfica en los sistemas de telecomunicaciones modernos (Inzunza-González, 2013; Cerqueira Ferreira, 2017).

La criptografía caótica implementada en dispositivos digitales, propone métodos y técnicas potenciales para dar solución a problemas derivados de la seguridad criptográfica (Murillo-Escobar *et al.*, 2016; Inzunza-González, 2013; Cruz-Hernández *et al.*, 2010). Sin embargo, las limitantes o problemas importantes al implementar los modelos

caóticos en métodos criptográficos con tecnología digital en sistemas embebidos, derivan principalmente de la limitante en la precisión numérica con la que se trabaja (Shujun *et al.*, 2003; Li *et al.*, 2005) y de la complejidad en el diseño del método criptográfico con el propósito de incrementar la seguridad criptográfica. Es conocido el hecho de que debido al redondeo numérico y en relación a la complejidad en la dinámica del mapeo caótico implementado, en los generadores de caos digitales que utilizan en su mayoría aritmética de punto flotante con formato de precisión simple IEEE 754 de 32 bits (IEEE, 2008), después de un determinado número de iteraciones, el generador presenta cierto grado de periodicidad o degradación digital (Hanping *et al.*, 2014). Este efecto limita el cifrado caótico digital para grandes cantidades de información, tales como: imágenes de gran resolución, secuencia de imágenes, audio y video.

En la literatura, se reportan métodos que proponen soluciones al problema de la precisión numérica con aritmética de punto flotante, con el propósito de disminuir la degradación digital del caos con base a tres vertientes principales: *i*) la implementación de modelos caóticos cuyas dinámicas son muy complejas, o de múltiples enrollamientos (Yashuang *et al.*, 2015; Yu-Ming *et al.*, 2010; Tlelo-Cuautle *et al.*, 2015b), *ii*) el empleo de más de un modelo caótico en cascada (Inzunza-González, 2013; Azzaz *et al.*, 2013a; Heidari-Bateni y McGillem, 1994) y *iii*) el incremento en la precisión numérica utilizando supercomputadoras o sistemas embebidos basados en FPGA (Tlelo-Cuautle *et al.*, 2015a; Sadoudi *et al.*, 2013; Azzaz *et al.*, 2013b). No obstante, en todos los casos, la implementación depende directamente de los sistemas integrales de desarrollo, de los sistemas de compilación, o de los lenguajes exclusivos de programación que requiere el dispositivo o sistema digital empleado. Tal es el caso por ejemplo, de los sistemas embebidos basados en microcontrolador, en donde se requiere de los compiladores de programas de acuerdo al fabricante, o de lenguajes de descripción de hardware complejos como VHDL (IEEE Computer, 2008) para el diseño en sistemas embebidos basados

en FPGA. Por otra parte, se reportan investigaciones donde se proponen métodos con criptografía caótica para el incremento de la seguridad criptográfica, implementando técnicas de confusión y difusión, con lo cual los métodos se vuelven más complejos y en su mayoría no son viables para su implementación en sistemas digitales embebidos, debido a alto poder computacional que demandan.

En la actualidad y como parte de las tecnologías emergentes de nueva generación, se han desarrollado dispositivos digitales embebidos "System on a Chip" (SoC) basados en microprocesadores de alto rendimiento, que permiten el diseño de métodos con procesamiento computacional avanzado a muy bajo costo y de fácil adquisición (Wei *et al.*, 2017). Los SoC contemplan sistemas operativos de uso común y de código abierto adaptados a sistemas embebidos digitales, lo que incrementa las posibilidades de desarrollos e implementaciones diversas sin necesidad de compiladores o sistemas integrales de desarrollo particulares, integrando una gran diversidad de software de diseño. Así mismo, los lenguajes de programación de alto nivel que incluyen bibliotecas para cómputo científico, facilitan el desarrollo de aplicaciones al integrar cálculo numérico con precisión arbitraria, lo cual permite incrementar la precisión numérica tanto como se requiera (Smith, 2003).

Por lo anterior, el presente trabajo de tesis se enfoca principalmente en el incremento de la precisión numérica en un generador de caos digital implementado en un SoC Raspberry Pi 3. Lo anterior, con el propósito de proponer primero, una alternativa de bajo costo y de fácil integración en sistemas digitales embebidos, para disminuir significativamente la degradación digital del caos y generar series pseudoaleatorias cifrantes potencialmente utilizables en métodos con criptografía caótica para grandes cantidades de información; y en segundo, un método para incrementar el espacio de claves y elevar el nivel de seguridad criptográfica considerando un esquema sensillo en estructura

potencialmente viable para ser implementado en sistemas SoC de bajo costo. Específicamente, se propone i) un generador de caos de gran precisión para obtener secuencias pseudoaleatorias empleando diferentes mapeos caóticos documentados en la literatura, ii) un generador de valores pseudoaleatorios válidos para condiciones iniciales de acuerdo al sistema caótico implementado con base en el generador de caos propuesto, iii) se desarrolla un algoritmo de lógica operacional simple para encriptar y desencriptar imágenes a color en formato RGB de alta resolución como ejemplo de aplicación del generador propuesto para comprobar la efectividad y viabilidad del sistema criptográfico en el encriptado de imágenes de alta resolución y finalmente, se propone un método con técnicas de procesamiento en paralelo que implementa la metodología propuesta en el presente trabajo de tesis con el fin de presentar una alternativa para disminuir los tiempos de encriptado y desencriptado de imágenes a color al implementarse en dispositivos embebidos SoC.

I.1 Planteamiento del problema

Considerando los argumentos antes mencionados, se requiere de un sistema criptográfico que sea implementado en dispositivos digitales embebidos SoC de alto rendimiento y de bajo costo, que realice el encriptado y desencriptado de una imagen de alta resolución, utilizando criptografía caótica. Lo anterior, con el propósito de enviar con un alto nivel de seguridad, el criptograma resultante por un canal público de comunicación remota a un usuario receptor autorizado y considerando un método de procesamiento de datos en paralelo para optimizar el tiempo de encriptado y desencriptado de la imagen.

I.1.1 Preguntas de investigación

- ¿Cómo se implementa un generador de caos de gran precisión en un sistema SoC Raspberry Pi 3?

- ¿Cómo se disminuye significativamente la degradación digital del caos incrementando la precisión numérica?
- ¿Cómo se encripta con un alto nivel de seguridad una imagen a color de alta resolución utilizando un generador de caos de gran precisión?
- ¿Cómo se disminuye el tiempo de encriptado y desencriptado de una imagen a color de alta resolución, implementando dispositivos SoC Raspberry Pi 3?

I.1.2 Delimitación del problema

Delimitación geográfica

En el presente trabajo de tesis se considera una delimitación geográfica a nivel mundial, debido a que el sistema criptográfico propuesto (transmisor y receptor) esta remotamente ubicado para el envío y recepción de la información encriptada a través de Internet.

Delimitación tecnológica

El presente trabajo de tesis implementa experimentalmente la propuesta de solución en un Soc Raspberry Pi 3. El sistema Raspberry Pi 3, está construido empleando un SoC Broadcom 2837 con un microprocesador Quad-Core ARM v8 de 64 bits con reloj de 1.2 GHz y 1 GB de memoria RAM; puede ser controlado por una adaptación de un sistema operativo de uso común como Windows10, Ubuntu o Debian. En este caso, el sistema operativo instalado en el SoC es una adaptación para sistemas embebidos del sistema operativo Debian conocido como Raspbian. El sistema operativo Raspbian permite la instalación de una gran variedad de software de desarrollo de aplicaciones de carácter científico incluyendo un compilador para lenguaje de programación Python (Larsen *et al.*, 2017). El lenguaje de programación Python, proporciona un paquete de

librerías extenso para el desarrollo de aplicaciones de carácter científico, tal es el caso de la biblioteca **numpy** y la librería **decimal**.

Delimitación de usuarios

En este proyecto se considera un usuario que realiza la implementación del sistema criptográfico propuesto para encriptar una imagen o secuencia de imágenes como mensaje confidencial y envía mediante internet el criptograma resultante a un usuario receptor autorizado. El usuario receptor autorizado, recibe el criptograma e implementa el sistema criptográfico propuesto para descryptar y obtener la información original. Un tercer usuario realiza funciones de criptoanálisis utilizando el mismo sistema criptográfico, a este usuario comúnmente se le llama intruso o hacker del sistema.

I.2 Objetivos del proyecto

I.2.1 Objetivo general

Al finalizar el trabajo de investigación, se pretende obtener un sistema criptográfico para el encriptado y descryptado de imágenes a color de alta resolución, implementado mediante criptografía caótica en un sistema embebido SoC Raspberry Pi 3 con un alto nivel de seguridad criptográfica y optimizando el tiempo de encriptado y descryptado con procesamiento paralelo.

I.2.2 Objetivos específicos

- Implementar un generador de caos de gran precisión numérica utilizando mapeos caóticos documentados en la literatura, en un dispositivo embebido SoC Raspberry Pi 3, con el propósito de generar series pseudoaleatorias potencialmente viables para ser utilizadas como series cifrantes en métodos con criptografía caótica.
- Implementar un método para el encriptado y descryptado de imágenes a color

de alta resolución, utilizando el generador de caos de gran precisión propuesto, para obtener altos niveles de seguridad criptográfica mediante el incremento del espacio de claves.

- Implementar una metodología de procesamiento en paralelo, para optimizar el tiempo total de encriptado y desencriptado de imágenes a color de alta resolución empleando el método criptográfico propuesto.

I.3 Organización de la tesis

El presente trabajo de tesis se organiza de la siguiente manera. En el **capítulo II** se presenta la implementación de la metodología propuesta para la construcción de un generador de caos de gran precisión numérica desde la perspectiva de la electrónica digital y el uso del sistema embebido SoC Raspberry Pi 3. En el **capítulo III** se presenta una metodología para el encriptado y desencriptado de imágenes a color de alta resolución que incrementa exponencialmente el espacio de claves del sistema criptográfico mediante el uso del generador de caos de gran precisión propuesto. En el **capítulo IV** se presenta una metodología que integra técnicas de procesamiento en paralelo con el propósito de optimizar los tiempos de ejecución en el encriptado y desencriptado de imágenes a color, implementada en el sistema embebido Raspberry Pi 3. En el **capítulo V** se presentan los resultados experimentales, incluyendo pruebas de análisis estadístico y ataques diferenciales. Finalmente en el **capítulo VI** se presentan las conclusiones generales del trabajo de tesis y el planteamiento de posibles líneas de investigación para trabajos a futuro.

Capítulo II

Generador de caos de gran precisión (HPG)

II.1 Introducción

La criptografía caótica implementada en sistemas electrónicos digitales, se presenta como una alternativa para solucionar las necesidades de comunicarse de manera segura por canales públicos de telecomunicación hoy en día. En el estudio de los sistemas de comunicaciones seguras, la criptografía caótica utiliza como generador de series cifrantes, un sistema caótico determinístico, que a pesar de no tener entradas aleatorias, presenta un comportamiento sumamente complejo y aparentemente aleatorio. Los generadores de caos digitales que implementan aritmética de punto flotante con formato de precisión simple IEEE 754 de 32 bits (IEEE, 2008), cada ecuación que compone al sistema caótico, produce elementos pseudoaleatorios cuyos valores numéricos contienen hasta 9 decimales significativos por cada iteración del sistema de ecuaciones.

Es conocido el hecho que, debido al redondeo numérico y en relación a la complejidad en la dinámica del mapeo caótico implementado, después de un determinado número de iteraciones el generador presenta cierto grado de periodicidad o degradación digital de caos (Hanping *et al.*, 2014). Este efecto limita el cifrado caótico digital para grandes cantidades de información, tales como: imágenes de gran resolución, secuencia de imágenes, audio y video; ejemplo de ello se observa en la figura 1 donde debido a la degradación que se presenta, el criptograma de la imagen Paisaje.png de 1950×1200

pixeles, muestra vestigios de la imagen original. Por otra parte, en los sistemas de computadora y en algunos sistemas embebidos digitales, es posible implementar criptografía caótica utilizando precisión IEEE 754 de 64 bits o doble precisión (IEEE, 2008). Sin embargo, los valores numéricos pseudoaleatorios generados con doble precisión, contienen hasta 17 decimales significativos por cada iteración de su sistema de ecuaciones; por lo tanto, el efecto de degradación digital sigue estando presente, pero a un mayor plazo de tiempo discreto.

La degradación digital que se presenta en el sistema caótico, se debe a que la precisión del cálculo numérico depende de la longitud física de los registros de memoria en el microcontrolador/microprocesador utilizado; es decir, el estándar IEEE 754 de precisión simple almacena los valores numéricos de punto flotante en una composición de 32 bits en los registros de memoria, mientras que el estándar IEEE 754 de doble precisión almacena los valores numéricos en una composición de 64 bits (IEEE, 2008), por lo que los valores numéricos se redeondean. En esta investigación se considera, bajo un esquema sencillo en estructura, el incremento de la precisión numérica en un generador de caos digital. Lo anterior, con el propósito de proponer una alternativa de bajo costo y de fácil integración en sistemas embebidos SoC de última generación, para disminuir significativamente la degradación digital del caos y generar series pseudoaleatorias cifrantes potencialmente utilizables en métodos con criptografía caótica digital para grandes cantidades de información, es decir, para imágenes de gran resolución.

II.2 Metodología

La metodología propuesta, considera utilizar mapeos caóticos documentados en la literatura para el diseño de un generador de caos de gran precisión (HPG). El diagrama de bloques de la figura 2, muestra el sistema generador de series pseudoaleatorias propuesto con base en el diseño de un generador de caos de gran precisión (HPG). El

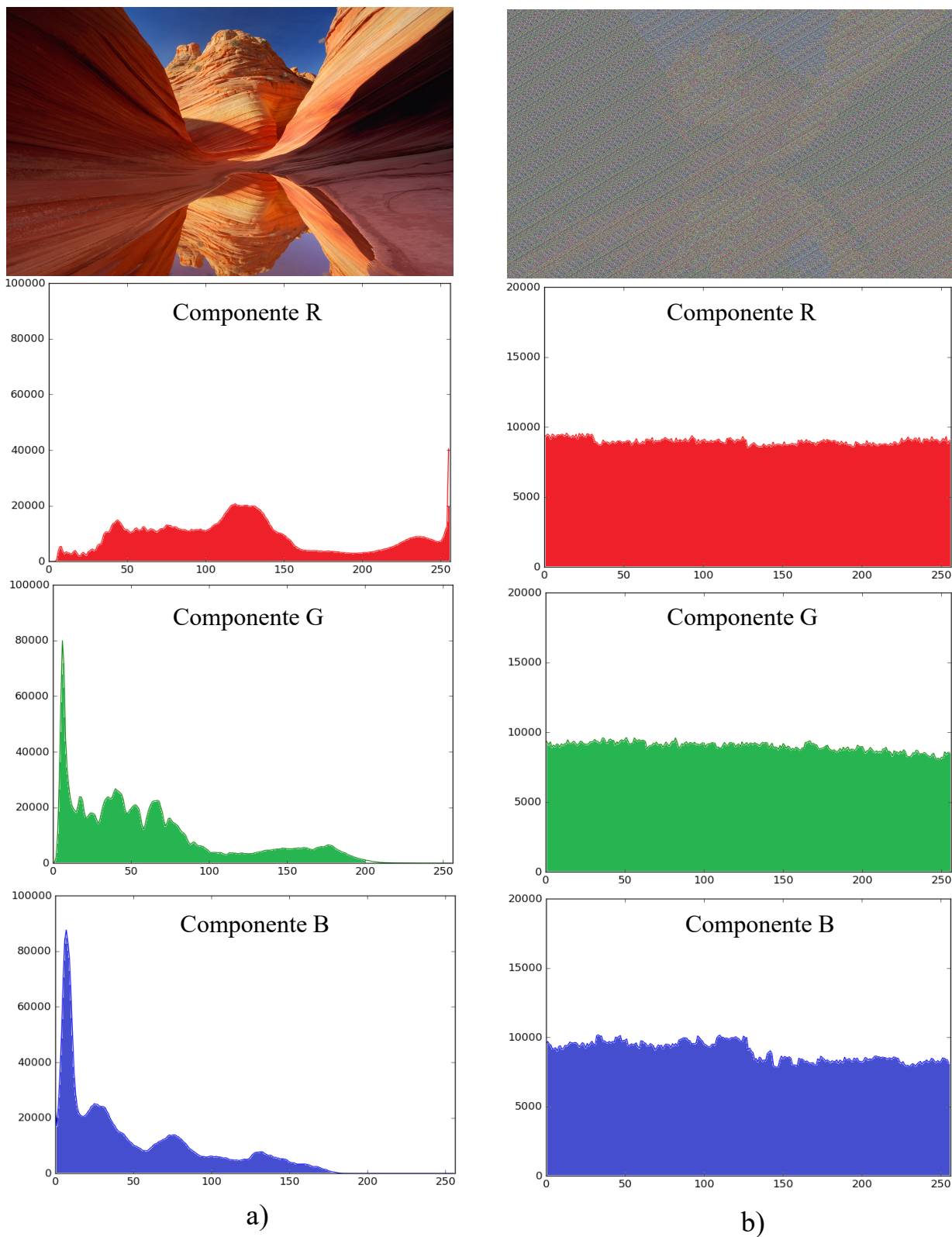


Figura 1: Efecto de degradación digital del caos en el encriptado de imágenes de gran resolución.

sistema generador se implementa en Raspberry Pi 3 a partir de tres entidades básicas: i) Parámetros de control, ii) Condiciones iniciales y iii) Mapeo caótico. Cada estado $X = \{x_0, x_1, \dots, x_n\}$, $Y = \{y_0, y_1, \dots, y_n\}$, ... , $Z = \{z_0, z_1, \dots, z_n\}$, relativo a un mapeo caótico, corresponde a una serie pseudoaleatoria con elementos de punto flotante con gran precisión numérica. En la entidad Mapeo caótico, se implementan las dinámicas de: Tinkerbell (Shaoliang y Jiang, 2011), Chen (Chen *et al.*, 2004), Rössler (Itoh *et al.*, 2001) y Logistic 2D (Verhulst, 1845) ampliamente documentados en la literatura, con el propósito de comparar resultados y evaluar la seguridad criptográfica que proporciona cada uno en el sistema propuesto; estos mapeos se eligieron sin considerar algún criterio en específico.

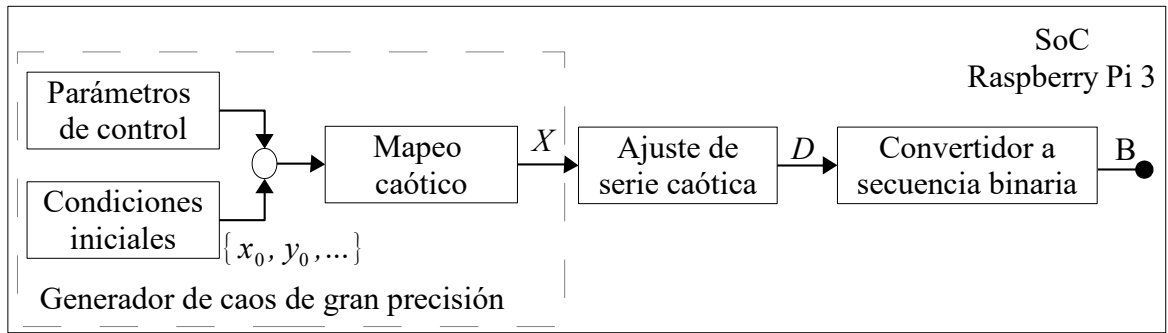


Figura 2: Generador de caos de gran precisión (HPG) propuesto.

II.2.1 Programa para generar caos de gran precisión

Considerando como ejemplo al mapeo de Tinkerbell (Shaoliang y Jiang, 2011), compuesto por dos ecuaciones en diferencias como las que se muestran en el sistema de ecuaciones 1, su comportamiento dinámico depende de cuatro parámetros de control a, b, c y d . Se conoce que los valores para $a = 0.9$, $b = -0.6013$, $c = 2$ y $d = 0.5$, lo mantienen en un régimen caótico (Goldszsztejn *et al.*, 2011) y que los valores para las condiciones iniciales x_0 e y_0 corresponden a puntos que se encuentran dentro del espacio del atractor caótico, por ejemplo $x_0 = -0.72$ e $y_0 = -0.64$.

$$\begin{aligned}
x_{n+1} &= x_n^2 - y_n^2 + ax_n + by_n \\
y_{n+1} &= 2x_ny_n + cx_n + dy_n
\end{aligned} \tag{1}$$

El mapeo caótico se implementa mediante programación en el SoC Raspberry Pi 3, utilizando lenguaje de programación Python. La cantidad de elementos de la serie pseudoaleatoria necesarios para encriptar una imagen digital en su totalidad, se representa por la letra k . Por lo tanto, si se considera la salida de una de las ecuaciones del mapeo caótico para producir S , entonces se necesita realizar k iteraciones en el sistema de ecuaciones del mapeo caótico a utilizar. En la figura 3 se ilustra el algoritmo propuesto para la generación de series caóticas empleando gran precisión numérica. El primer paso corresponde en definir la cantidad de decimales significativos a considerar en el cálculo numérico y que producirá las series caóticas con la gran precisión deseada. Posteriormente, se establecen los valores de los parámetros de control y de las condiciones iniciales relativos al mapeo caótico que lo mantengan en una dinámica compleja, finalmente, se ejecutan las k iteraciones necesarias del sistema de ecuaciones del mapeo caótico para producir la serie $X = \{x_0, x_1, \dots, x_k\}$.

En términos de programación, la directiva que carga al módulo de cálculo numérico de precisión arbitraria en el programa principal, se compone de **from * import decimal**. El parámetro que determina la precisión numérica a utilizar se establece mediante la función **getcontext ().prec = np**, donde np es el número de decimales significativos a considerar en los cálculos matemáticos. A partir de estas ejecuciones, los valores numéricos generados por las ecuaciones del mapeo caótico respectivo en cada iteración, tienen el nivel de precisión con np decimales significativos. Por otra parte, la asignación de los valores numéricos con la precisión definida, se ejecuta mediante la función *Decimal()*. La función *Decimal()* regresa el valor numérico del parámetro numérico

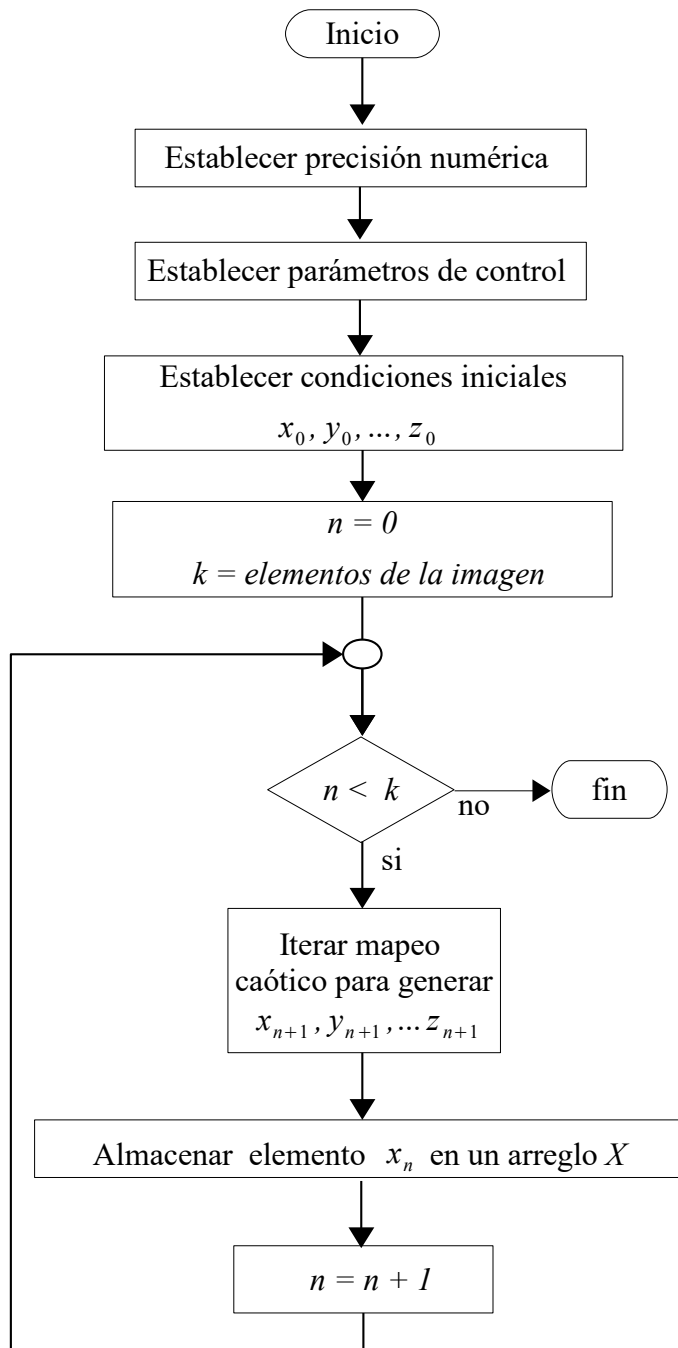


Figura 3: Diagrama de flujo: Generador de series caóticas con el HPG propuesto.

con la cantidad de np decimales significativos. El código del programa Generador.py que se muestra a continuación, se implementa en lenguaje de programación Python el procedimiento de la figura 3 para generar $k = 1000$ elementos y almacenar series pseudoaleatorias implementando el mapeo de Tinkerbell (Ec 1), estableciendo una precisión numérica de $np = 99$ decimales significativos.

```
# Programa Generador.py
-----
from decimal import *      #Carga la libreria decimal
getcontext().prec=99      #Establece la precision a utilizar
                           #Establecimiento de parámetros de control
a=Decimal('0.9')
b=Decimal('-0.6013')
c=Decimal('2')
d= Decimal('0.5')

                           #Establecimiento de condiciones iniciales
x0= Decimal('-0.72')
y0= Decimal('-0.64')
X=[x0]                    #Arreglo para almacenar la serie X
Y=[y0]                    #Arreglo para almacenar la serie Y

n = 0
k = 1000                  #Elementos a generar

while(n < k):              #Realiza k iteraciones del mapeo caótico
    x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
    y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
    X += [x1]              #Almacena el elemento generado en X
    Y += [y1]              #Almacena el elemento generado en Y
    x0 = x1
    y0 = y1
    n= n+1
-----
```

De acuerdo al código anterior, por cada k iteraciones, se generan elementos pertenecientes a una serie de números cuya pseudoaleatoriedad depende del comportamiento caótico del mapeo implementado, pero que en este caso, se le adiciona una alta precisión numérica predefinida con $np = 99$ decimales significativos. La dinámica de un mapeo caótico, tiene relación directa con los valores de sus parámetros de control implícitos en sus sistemas de ecuaciones y de los valores en sus condiciones iniciales, por lo tanto, es imprescindible establecer valores para el conjunto de estas variables, que garanticen

su comportamiento caótico según sea el caso.

II.2.2 Condiciones iniciales y parámetros de control

Una forma de conocer el rango de valores para los parámetros de control que garantizan una dinámica caótica, es mediante sus diagramas de bifurcación (Shaoliang y Jiang, 2011). En relación a las condiciones iniciales $(x_0, y_0, \dots, z_0)$ de cada mapeo caótico, es necesario establecer valores que se encuentren dentro del espacio del atractor caótico. Una forma de establecer condiciones iniciales, tiene su fundamento en la propiedad de sensibilidad a condiciones iniciales (Moon, 1992); así, al aplicar un cambio mínimo o incremento Δx en una condición inicial dada, se obtiene una nueva condición inicial que genera una secuencia dinámica diferente. Por ejemplo, si $x_0 = -0.72$ es una condición inicial válida para el mapeo de Tinkerbell, considerando un $\Delta x = -1 \times 10^{-8}$, la nueva condición inicial se obtiene mediante

$$x'_0 = x_0 + \Delta x = -0.72000001.$$

Siguiendo este mismo procedimiento para establecer $y'_0 = y_0 + \Delta y$.

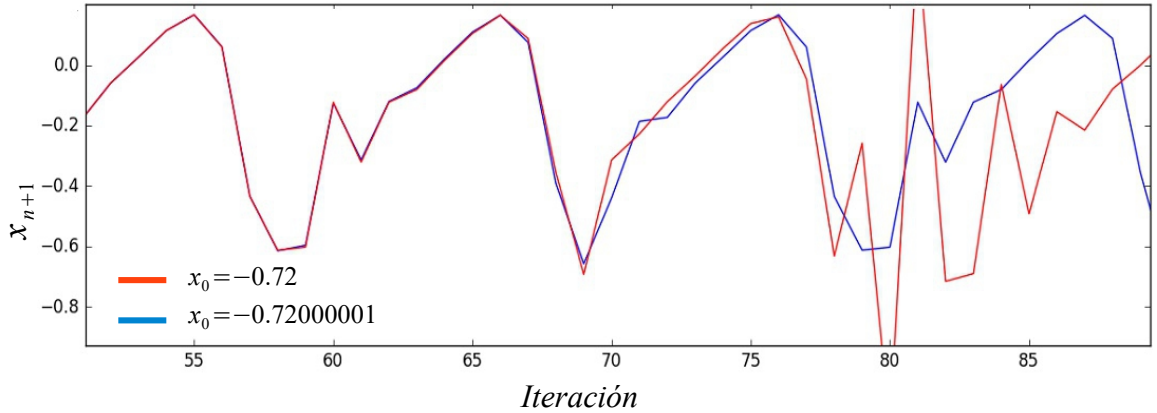


Figura 4: Trayectorias caóticas. Condiciones iniciales aplicando un Δx a x_0 .

La figura 4 muestra las trayectorias caóticas del estado x_{n+1} que se obtienen considerando el método propuesto para establecer condiciones iniciales. Se observa que, debido

a la sensibilidad a condiciones iniciales que presentan los sistemas caóticos, se generan trayectorias caóticas distintas. Sin embargo, ambas trayectorias son muy semejantes en las primeras iteraciones, lo cual para propósitos criptográficos, no es recomendable. Adicionalmente, es necesario asegurar que el incremento en la condición inicial no resulte en un valor fuera del atractor caótico, esto para garantizar que la trayectoria sea caótica. Bajo esta perspectiva, el presente trabajo de investigación, propone el siguiente método para establecer condiciones iniciales válidas para un determinado mapeo caótico, que consiste en utilizar el mismo mapeo caótico para generar nuevas condiciones iniciales.

II.2.3 Generador de condiciones iniciales pseudoaleatorias

El método propuesto para establecer condiciones iniciales válidas para un determinado mapeo caótico que se describe a continuación, tiene su fundamento en el establecimiento y control de la precisión numérica.

- Paso 1: Se establecen las primeras condiciones iniciales eligiendo valores numéricos para x_0 e y_0 dentro del espacio del atractor caótico, por ejemplo: $x_0 = -0.72$ e $y_0 = -0.64$ para el caso del mapeo de Tinkerbell.

- Paso 2: Se hace iterar el mapeo caótico con las condiciones iniciales establecidas considerando gran precisión para obtener los valores x_{n+1} e y_{n+1} . Ejemplo, con $np = 99$ decimales significativos, se obtiene:

$$x_1 = -0.15436800000000002343059080089915315923538115187293955020...$$

$$y_1 = -0.83839999999999996829203041670552991547589425823679674555...$$

- Paso 3: Los valores obtenidos se redondean a un número menor de decimales significativos np_{x_0} para obtener las nuevas condiciones iniciales x'_0 e y'_0 . Ejemplo, redondeo a $np_{x_0} = 4$ decimales significativos.

$$x'_0 = -0.1544$$

$$y'_0 = -0.8384.$$

Con lo anterior, se garantiza que los nuevos valores de condiciones iniciales se encuentran dentro del espacio del atractor caótico y son válidos para nuevas condiciones iniciales. De esta forma, el método propuesto genera valores numéricos para las condiciones iniciales muy diferentes entre si, lo que produce series caóticas con comportamientos dinámicos distintos sin perder sus propiedades caóticas, lo cual las hace idóneas para emplearse en el encriptado de imágenes de alta resolución. La figura 5 muestra las trayectorias caóticas del estado de fase x_{n+1} que se obtienen considerando el método propuesto. Se puede observar que debido a la sensibilidad a las condiciones iniciales que presentan los sistemas caóticos, se generan trayectorias distintas desde la primer iteración.

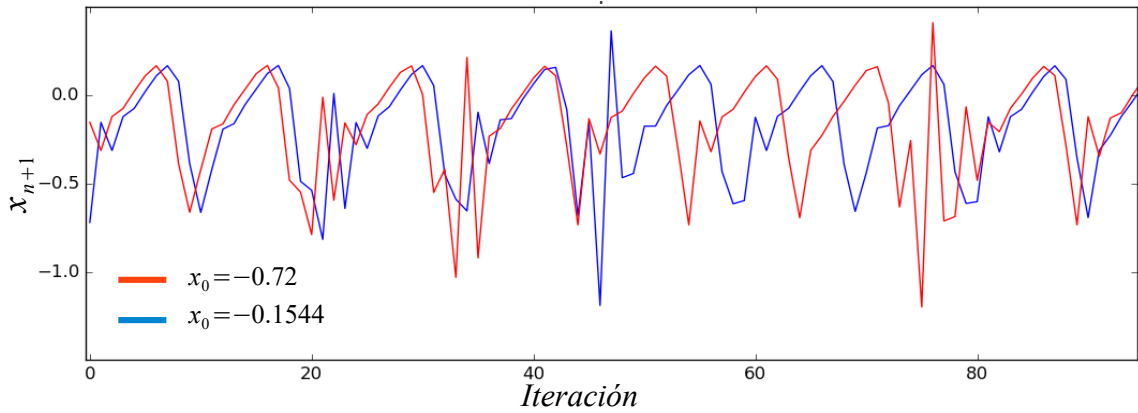


Figura 5: Trayectorias caóticas con condiciones iniciales generadas pseudoaleatoriamente implementando HPG propuesto.

Con el propósito de validar las series pseudoaleatorias obtenidas mediante pruebas estándares de análisis de series aleatorias de bits, en el presente trabajo se propone un método para generar series pseudoaleatorias de bit, a partir del generador de caos HPG propuesto. La validación de las series caóticas generadas, se realiza mediante el conjunto

de pruebas estadísticas aprobadas por el Instituto Nacional de Estándares y Tecnología (FIPS 140-2 NIST) (NIST, 2015) y NIST 800-22 Rev 1a (Rukhin *et al.*, 2010). Estas pruebas estadísticas analizan una secuencia de bits generada con un número mayor a 100,000 elementos, incluso algunas pruebas requieren mas de 1,000,000 de bits, para determinar si corresponden a una fuente generadora de bits verdaderamente aleatoria.

II.2.4 Generador de serie de bits pseudoaleatorios

El diagrama de bloques de la figura 6, muestra el método propuesto para generar una serie pseudoaleatoria de bits, con base en el generador de caos de gran precisión (HPG) propuesto. El método propuesto produce una secuencia de bits por cada valor de x_{n+1} o de y_{n+1} en cada una de las iteraciones del sistema de ecuaciones implementado en el bloque Mapeo caótico. Lo anterior se realiza convirtiendo cada elemento de la serie generada en un equivalente binario mediante los bloques **Ajuste de serie caótica** y **Convertir a secuencia binaria**. El equivalente binario que resulta en cada iteración es concatenado cada vez y se almacena en una cadena de bits resultante identificada con la letra **B**, tal y como se muestra en la figura 6. De esta manera, se puede generar una cadena **B** pseudoaleatoria de bits, por cada una de las ecuaciones que contenga el sistema caótico implementado.

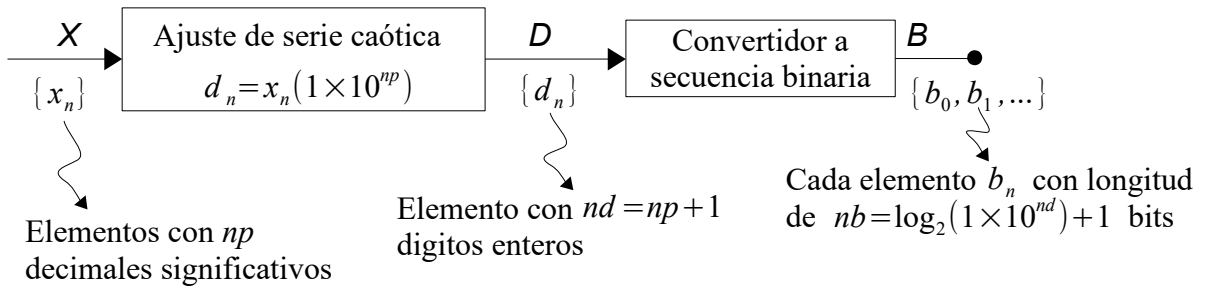


Figura 6: Diagrama de bloques del generador de serie pseudoaleatoria de bits mediante HPG.

En relación a la figura 6 y considerando solamente la salida del estado **X** del mapeo

caótico, la serie $X = \{x_0, x_1, \dots, x_{k-1}\}$, cuyos elementos tienen una precisión numérica de np decimales significativos, generados por la entidad **Generador de caos de gran precisión** de la figura 2, ingresa a la entidad **Ajuste de serie caótica**, para obtener una serie de elementos de tipo entero $D = \{d_0, d_1, \dots, d_{k-1}\}$; la serie **D** resultante ingresa a la entidad **Convertidor a secuencia binaria** de la cual se obtiene la secuencia binaria $B = \{\{b_{0,0}, b_{0,1}, \dots, b_{0,nb}\}, \{b_{1,0}, b_{1,1}, \dots, b_{1,nb}\}, \dots, \{b_{k-1,0}, b_{k-1,1}, \dots, b_{k-1,nb}\}\}$, tal y como se indican en la figura 6. Si se define a nd como la cantidad de dígitos que contiene cada elemento d_n y a nb como la cantidad de bits que se obtienen al convertir el elemento d_n en una secuencia de bits; entonces, en relación a la 6, cada elemento x_n de la serie caótica de gran precisión obtenido, es multiplicado por 1×10^{np} para obtener un número entero d_n representado por un máximo de $nd = np + 1$ dígitos enteros. Si el número de bits que compone una representación binaria de un número representado por un entero decimal esta determinado por $\log_2(a)$, donde a es un número entero decimal; entonces, la longitud máxima de la secuencia de bits que se obtiene de cada elemento d_n en la subentidad **Convertidor a secuencia binaria** es de $nb = \log_2(1 \times 10^{nd}) + 1$ bits incluyendo el bit de signo.

La tabla I, muestra los valores asociados para nd y nb en relación a algunos valores predefinidos para np y a manera de comparación, los niveles de los estándares para simple y doble precisión implementados en el generador propuesto. Al incrementar el número de dígitos significativos, se incrementa el número de bits que se obtienen en cada iteración del mapeo caótico. Es decir, al utilizar precisión simple, el valor numérico que se obtiene del generador de caos es de $np = 9$ decimales significativos, del cual se obtiene d_n con un máximo de $nd = 10$ dígitos, por lo tanto, la secuencia de bits b_n tiene una longitud máxima de $nb = \log_2(1 \times 10^{10}) + 1 = 34.21$ bits en cada iteración; mientras que al utilizar gran precisión con **np** = 999 se obtienen hasta 3,322.92 bits por cada iteración.

Tabla I: Longitud de la secuencia pseudoaleatoria de bits obtenida de x_n .

Nivel de precisión	Valor np	Número de dígitos $nd = np + 1$	Longitud $nb = \log_2(1 \times 10^{nd}) + 1$ bits
Precisión simple	np = 9	10	34.21
Doble precisión	np = 17	18	60.79
Gran precisión	np = 99	100	333.1928
Gran precisión	$np = 999$	1000	3322.928

En la subrutina **Convertidor a secuencia binaria** que se muestra en la figura 6, se ejecuta una adaptación del método de divisiones sucesivas para convertir el elemento d_n en una secuencia pseudoaleatoria de bits. En la figura 7 se presenta el diagrama de flujo de esta subentidad. Como primer paso se almacena el bit b_0 que corresponde al signo del elemento x_n , donde 0 es para positivo y 1 es para negativo. Posteriormente a partir del valor d_n , se obtienen los valores sucesivos de la secuencia binaria $b_{n,1}, b_{n,3}, \dots$ donde cada bit corresponde a los residuos de las divisiones sucesivas por 2, hasta que el cociente de la división sea 0. El programa "generador_bits".py que se muestra a continuación, implementa la metodología propuesta considerando el estado de fase x_{n+1} del mapeo de Tinkerbell, se crea un archivo de texto con el nombre data_Tinkerbell_X.hpg que contiene 100 secuencias con elementos 0 y 1 de tipo caracter, con una longitud de 1,000,000 de bits cada una para el correspondiente análisis estadístico FIPS-NIST.

```
#Programa generador_bits.py
from decimal import *
sfile=open('data_Tinkerbell_X.hpg','a')    #crea el archivo de datos.
nstream=100                                #número de series a generar.
long_stream=1000000                        #longitud de cada serie a generar.

np=0                                        #contador de cadenas
ret=500                                    #retardo de iteraciones
np_prec=99                                #establece la precisión de la variable np
npx0=8                                     #establece la precisión de la variable npx0.
xini=Decimal('-0.721')                    #condición inicial
yini=Decimal('-0.64')                     #condición inicial

while(np<nstream):
    getcontext().prec=np_prec              # define la precisión a considerar
    aa= Decimal(0.9)                       #establecimiento de parámetros de
    bb= Decimal(-0.6013)                    #control del sistema caótico
```

```

cc= Decimal(2.0)
dd= Decimal(0.5)
x0= Decimal(xini)      #cond. inicial X0
y0= Decimal(yini)      #cond. inicial Y0
i=0      #contador de numero de bits
nb=0     #contador de número de bits por cada elemento de X
j=0     #variable de incremento auxiliar
while (i<long_stream):
    x1 = Decimal((x0*x0)-(y0*y0)+(aa*x0)+(bb*y0)) #sistema
    y1 = Decimal((2*x0*y0)+(cc*x0)+(dd*y0))      #de ecuaciones
    if j>ret:
        if x0<0:
            x0=Decimal(x0*(-1))
        sfile.write('1')
    sfile.write('0')
    x0=Decimal(x0**np_prec)
    #_____convertidor a secuencia binaria_____
    while(x0>1):
        bit=int(x0%2)
        dato=str(bit)
        sfile.write(dato)
        x0=Decimal(x0/2)
        i=i+1 #incrementa el contador de bits
        if(i==long_stream):break
    #_____
    j =j+1
    x0=x1
    y0=y1
    #_____generador de condiciones iniciales pseudoaleatorias_____
    getcontext().prec=np_x0    #define precisión para redondeo
    xini =Decimal((x0*x0)-(y0*y0)+(aa*x0)+(bb*y0))
    yini =Decimal((2*x0*y0)+(cc*x0)+(dd*y0))
    np=np+1
sfile.close()
#fin del programa

```

En el presente trabajo de investigación y con el propósito de analizar y validar el nivel de aleatoriedad que presenta la serie de bits generada con el método propuesto, se generan 100 secuencias binarias con una longitud de 1000 bits, 100 secuencias binarias con una longitud de 2000 bits y así sucesivamente, hasta generar 100 secuencias binarias de 150,000 bits de longitud y utilizando en cada una de ellas precisión simple, doble precisión, gran precisión con $np = 99$ y gran precisión con $np = 999$. El análisis y validación de cada una de las secuencias binarias se realiza mediante las pruebas Five-Test de (FIPS 140-2 NIST) (NIST, 2015) y el conjunto de pruebas NIST 800-22 Rev

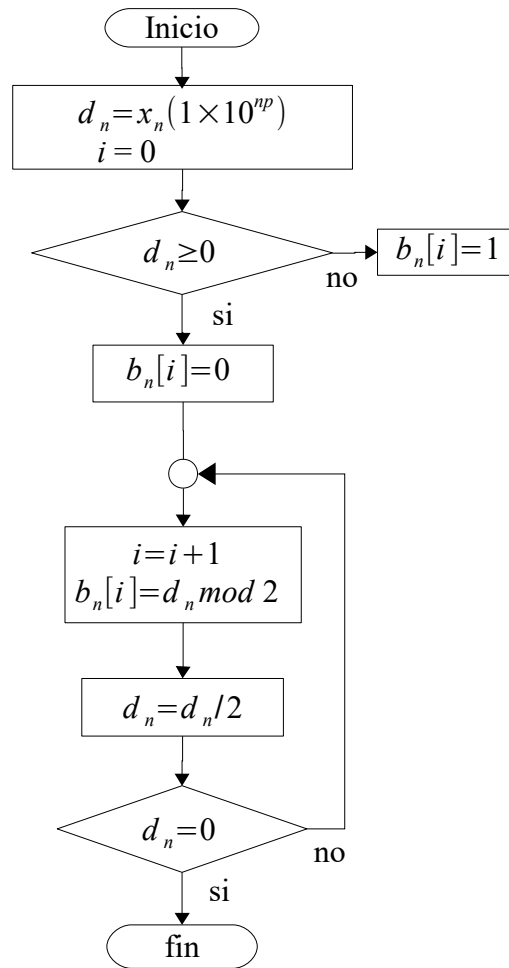


Figura 7: Diagrama de flujo de la subrutina Convertidor a secuencia binaria.

1a (Rukhin *et al.*, 2010) con los siguientes parámetros de ajuste:

- Block Frequency test - block length $(M) = 128$.
- Non Overlapping Template test - block length $(m) = 9$.
- Overlapping Template test - block length $(m) = 9$.
- Approximate Entropy test - block length $(m) = 10$.
- Serial test - block length $(M) = 16$.
- Linear Complexity test - block length $(M) = 500$.
- All test except Universal - Stream length $= 150,000$.
- Universal test - stream length $= 1000,000$.
- Bit streams $= 100$.

II.3 Conclusiones

Con el incremento en la precisión numérica, se disminuye significativamente la degradación digital del caos al ser implementado el sistema caótico en el dispositivo Raspberry Pi 3. Con la metodología propuesta, es sencillo realizar la experimentación con varios modelos caóticos documentados en la literatura, debido a que el modelo caótico se implementa por programación. De los resultados obtenidos con el análisis de las series pseudoaleatorias obtenidas bajo este contexto, se concluye que el generador es viable y altamente competitivo con otros generadores de números aleatorios y pseudoaleatorios documentados en la literatura. Por otra parte, las series generadas pueden ser utilizadas como series cifrantes con aproximación a una serie verdaderamente aleatoria, proporcionando una mayor seguridad criptográfica por el espacio de claves resultante.

Capítulo III

Encriptado y desencriptado de imágenes a color de alta resolución.

III.1 Introducción.

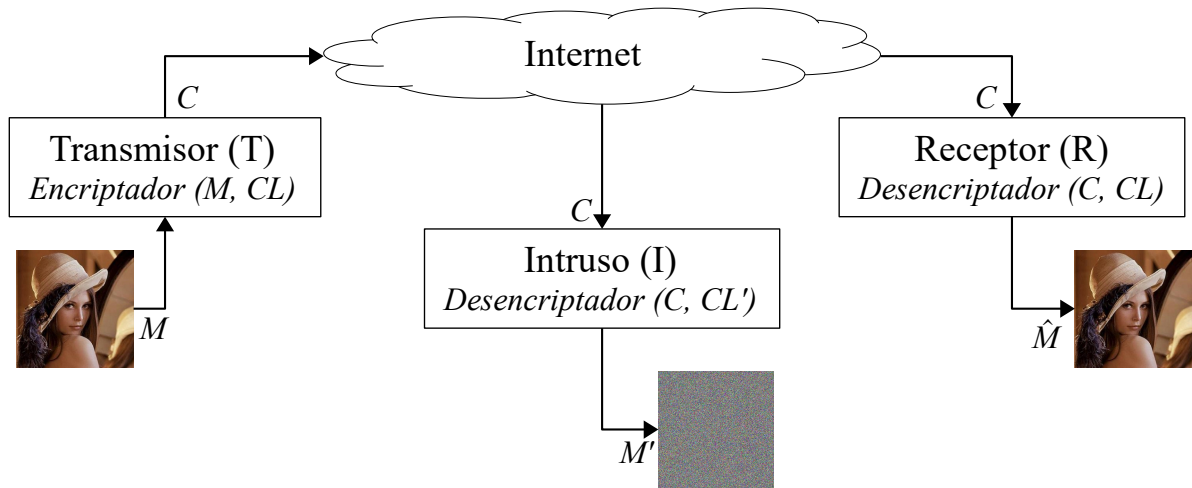


Figura 8: Comunicación segura mediante criptografía.

La figura 8, muestra básicamente la representación del concepto de comunicación segura mediante algún método criptográfico. El trasmisor (T), envía un mensaje (M) al receptor (R) remotamente ubicado, con el propósito de que el envío sea de carácter confidencial. Para ello, el mensaje (M) antes de ser enviado, pasa por un proceso de cifrado o encriptado, lo cual en principio, lo hace incomprensible para cualquier receptor no autorizado (I). Al mensaje encriptado, se le conoce como **criptograma** (C) y es el que viaja por el canal público de comunicación, por ejemplo internet. En el receptor (R), el criptograma (C) pasa por un proceso de desencriptado (proceso inverso al de

encriptado) del cual se obtiene o se recupera el mensaje original ($\hat{M}$), para que finalmente se interprete la información enviada por parte del receptor (R). En el diagrama de la figura 8, se muestra también al receptor no autorizado (I), cuya función es la de obtener el mensaje (M') contenido en el criptograma (C). A la acción del receptor no autorizado (I) de intentar obtener el mensaje, se le conoce como Criptoanálisis. El objetivo de la comunicación segura es que $M = \hat{M}$ y que $M \neq M'$ durante todo el proceso propio de la comunicación. Es importante mencionar que en la criptografía, para la etapa de cifrado y la de descifrado, se lleva a cabo a partir de un algoritmo y un elemento fundamental conocido como llave o clave de cifrado (CL). De esta forma, podemos plantear la función $Encriptado(M, CL)$, donde *Encriptado*, es la función de cifrado del mensaje (M) a partir de la clave (CL); y una función inversa de descifrado $Desencriptado(C, CL)$, donde *Desencriptado* es la función de descifrado inversa a la función *Encriptado* del criptograma (C) con la clave (CL).

III.1.1 Seguridad criptográfica y el secreto perfecto de Shannon

Una primera clasificación de los métodos criptográficos con base en las claves utilizadas se puede describir como sigue:

- **Criptografía con clave simétrica:** es el método criptográfico que utiliza la misma clave (CL) en el cifrado y en el descifrado. Luego entonces, la clave tiene que permanecer secreta, lo cual presupone que tanto el emisor como el receptor se han puesto de acuerdo previamente en la determinación de la misma. Este tipo de métodos son propios de la criptografía clásica (Diffie y Hellman, 1976).
- **Criptografía con clave asimétrica:** es el método criptográfico que utiliza claves (CL) diferentes en el cifrado y en el descifrado. Generalmente, la clave para cifrar es conocida libremente (pública), mientras que la clave para descifrar es

conocida solamente por el receptor. Estos métodos son propios de la criptografía de clave pública (Diffie y Hellman, 1976).

La principal diferencia entre los métodos clásicos y los métodos que se utilizan en la actualidad radica en el concepto de seguridad. En la antigüedad, los métodos criptográficos contaban con una seguridad “probable”, a diferencia de los métodos criptográficos modernos, que deben tener una seguridad matemáticamente demostrable; de lo anterior se establece una primera clasificación de seguridad criptográfica (Fuster *et al.*, 2001):

- **Seguridad incondicional (teórica):** el sistema es seguro frente a un atacante con tiempo y recursos computacionales ilimitados, ejemplo: cifrado Vernam (Fuster *et al.*, 2001).
- **Seguridad computacional (práctica):** el sistema es seguro frente a un atacante con tiempo y recursos computacionales limitados, ejemplo: sistemas de clave asimétrica basados en problemas de alta complejidad de cálculo (Fuster *et al.*, 2001).
- **Seguridad probable:** no se puede demostrar su integridad, pero el sistema no ha sido violado (Fuster *et al.*, 2001).
- **Seguridad condicional:** los sistemas que son seguros mientras que el criptoanalista no alcance los medios computacionales para violar la seguridad (Fuster *et al.*, 2001).

Los expertos en criptografía creyeron en la seguridad total del método de Vernam, pero el primero en proporcionar pruebas teóricas sobre la seguridad total fue Claude

Shannon en 1994. Shannon definió sus condiciones de secreto perfecto partiendo de dos hipótesis básicas:

1. La clave secreta se utiliza solamente una vez.
2. El enemigo criptoanalista tiene acceso sólo al criptograma, después, se limita a atacar la información cifrada únicamente.

A partir de la cuales, se enuncian sus condiciones que se describen a continuación. Un sistema criptográfico verifica las condiciones de secreto perfecto si la información original X es estadísticamente independiente del criptograma Y , lo que en lenguaje de probabilidad puede expresarse como:

$$P(X = x|Y = y) = P(X = x), \quad (2)$$

para todos los posibles textos fuente $x = (x_1, x_2, \dots, x_M)$ y todos los posibles criptogramas $y = (y_1, y_2, \dots, y_N)$, es decir, la probabilidad de que la variable aleatoria X tome el valor de x es la misma, con o sin conocimiento del valor tomado por la variable aleatoria Y . De la misma manera y basado en el concepto de entropía, la longitud de la clave (k) necesaria para que se pueda verificar la condición de secreto perfecto, debe ser igual o mayor a la longitud de la información original.

$$K \geq M. \quad (3)$$

Esta desigualdad se convierte en igualdad para el caso del cifrado Vernam. Se verifica entonces de lo anterior que, el cifrado Vernam cumple con las condiciones de secreto perfecto definidas por Shannon y se clasifica como el único procedimiento de cifrado incondicionalmente seguro. Sin embargo, presenta el inconveniente de que se requiere de un bit de enmascaramiento por cada bit de mensaje. Por lo que, teniendo en cuenta las exigencias actuales de información a cifrar, se tiene que el método no es

viable; debido a que al hacer llegar tal cantidad de bits del emisor al receptor por un canal de comunicación se desbordaría la propia capacidad del canal. En la práctica se utiliza el método de cifrado en flujo (Fuster *et al.*, 2001), el cual es una aproximación al cifrado Vernam. En el cifrado en flujo se utilizan generadores pseudoaleatorios de secuencias cifrantes, es decir, mediante la implementación de algoritmos determinísticos, que a partir de una clave secreta (CL) conocida únicamente por el emisor y el receptor autorizado, generan simultáneamente en uno y otro una determinada secuencia cifrante de una longitud deseada, haciendo las veces de la clave en el cifrado Vernam. Debido a que estas secuencias son generadas por una maquina de estados finitos, a partir de un algoritmo determinístico, nunca podrán ser verdaderamente aleatorias (Fuster *et al.*, 2001). A estas secuencias que se intentan sean lo más posiblemente semejantes a secuencias aleatorias, se les denomina secuencias pseudoaleatorias cifrantes.

III.1.2 Espacio de claves y ataque de fuerza bruta

Las posibilidad de que un criptoanálisis tenga éxito en el ataque o ruptura de un sistema de cifrado, tiene una fuerte dependencia de la información con la que dispone el criptoanalista y de las circunstancias que lo rodean. Luego entonces, el atacar un sistema de cifrado desconocido, demanda un mayor esfuerzo que el ataque a un sistema conocido. Si el sistema es conocido, existe un primer ataque que se enfoca en probar el número total de claves posibles que admite el sistema de cifrado; al total de claves posibles que admite el sistema de cifrado se le conoce como **espacio de claves** y al ataque del espacio de claves se le conoce como **Ataque de fuerza bruta** (Fuster *et al.*, 2001). Por lo anterior, es de vital importancia el desarrollo de métodos criptográficos que proporcionen un amplio espacio de claves, con el propósito de incrementar la seguridad del método criptográfico.

En la literatura se encuentran diversos métodos que realizan la transmisión y recuperación de imágenes, utilizando criptografía caótica e implementando sistemas digitales, como por ejemplo (Mazloom y Eftekhari-Moghadam, 2009; Murillo-Escobar *et al.*, 2015; Tlelo-Cuautle *et al.*, 2015a; Sadoudi *et al.*, 2013). Sin embargo, el avance tecnológico en el desarrollo de sistemas embebidos basados en microprocesadores y la utilización de software de desarrollo de aplicaciones de alto nivel, brindan la posibilidad de desarrollar nuevos métodos y técnicas para resolver los problemas derivados de las necesidades de la comunicación segura por internet utilizando gran potencial computacional a muy bajo costo. Las aportaciones principales de este trabajo de tesis son: i) se presenta una metodología para el encriptado y desencriptado de imágenes a color en formato RGB, utilizando criptografía caótica con clave simétrica y el método de cifrado en flujo con base en el generador HPG propuesto y empleando diferentes mapeos caóticos para la generación de series cifrantes, ii) la metodología propuesta, se implementa en un sistema embebido Raspberry Pi 3 basado en microprocesador gestionado por un sistema operativo de uso común y de código abierto (Raspbian), iii) se utiliza un software de alto nivel para el desarrollo de aplicaciones científicas basado en lenguaje Python, con la finalidad de que el sistema propuesto, se pueda implementar en cualquier sistema digital que soporte aplicaciones con lenguaje Python. Con lo anterior, se pretende proponer un sistema criptográfico que proporcione un amplio espacio de claves y de fácil integración en diversos sistemas embebidos SoC, gestionados por un sistema operativo de uso común y de código abierto; lo anterior, con el propósito de que el método criptográfico tenga niveles elevados de seguridad y pueda ser ampliamente escalable en diferentes plataformas de hardware.

III.2 Metodología.

III.2.1 Método de cifrado en flujo con clave simétrica basado en HPG propuesto.

El método propuesto para el encriptado de imágenes a color de alta resolución, se basa en el generador de series pseudoaleatorias que se presentó en la figura 2 de la sección II.2. En el SoC Raspberry Pi 3, el sistema criptográfico se implementa tal y como se muestra en la figura 9. El método que se emplea para encriptar o desencriptar las imágenes o criptogramas según sea el caso, tiene su fundamento en el cifrado en flujo con clave simétrica (Menezes *et al.*, 1996). La figura 9 muestra que cada elemento (pixel) de una imagen $M = \{m_1, m_2, \dots, m_k\}$, al ser procesado con un elemento de la serie cifrante $S = \{s_1, s_2, \dots, s_k\}$, producen los elementos del criptograma $C = \{c_1, c_2, \dots, c_k\}$. La entidad Ajuste de la serie cifrante, convierte cada elemento de la serie pseudoaleatoria $D = \{d_1, d_2, \dots, d_k\}$ en un correspondiente elemento de la serie cifrante $S = \{s_1, s_2, \dots, s_k\}$, con el propósito de que S sea del mismo tipo que M . Los elementos de M , son valores numéricos de 8 bits que corresponden al color de cada pixel en los componentes R, G y B de la imagen a encriptar. El ajuste de cada elemento $\{d_n\}$, consiste en obtener un valor de 8 bits $\{s_n\}$ a partir de $s_n = \text{mod}(d_n, 255)$. Por otra parte, en la entidad Método de encriptado, se ejecuta la operación binaria *XOR* entre M y S como operación de cifrado, tal y como se muestra en la figura 10.

En la figura 10, se muestra una de las ventajas de utilizar el operador *XOR* en el proceso de encriptado o desencriptado. Esto es debido, a que se mantiene invariante en software y hardware el sistema de encriptado, al ejecutar sin distinción el cifrado de la imagen o la recuperación de la imagen original, a partir de su criptograma respectivo. Dicho de otra forma, en la figura 10a, se representa el encriptado de una imagen digital "Lena.png" a color de 256×256 pixeles, encriptada a partir de las secuencias caóticas binarias $S = \{s_1, s_2, \dots, s_i\}$ generadas por el procedimiento mostrado en la figura 9 y

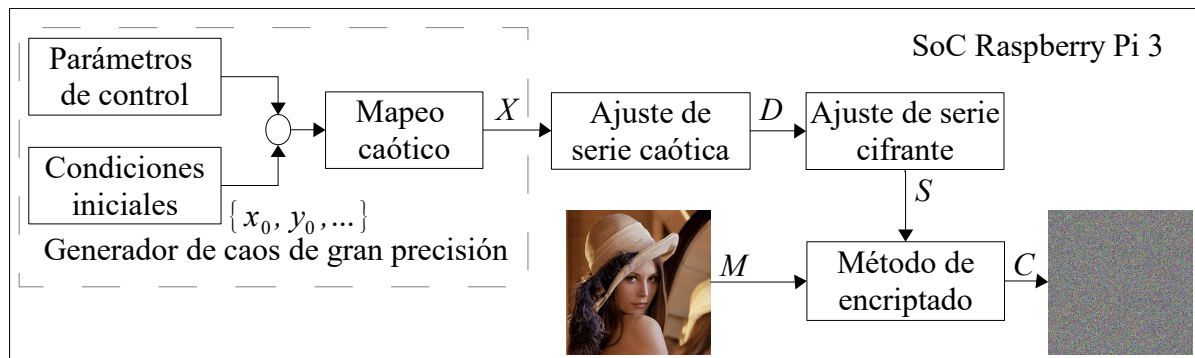


Figura 9: Método propuesto de encriptamiento de imágenes con criptografía caótica.

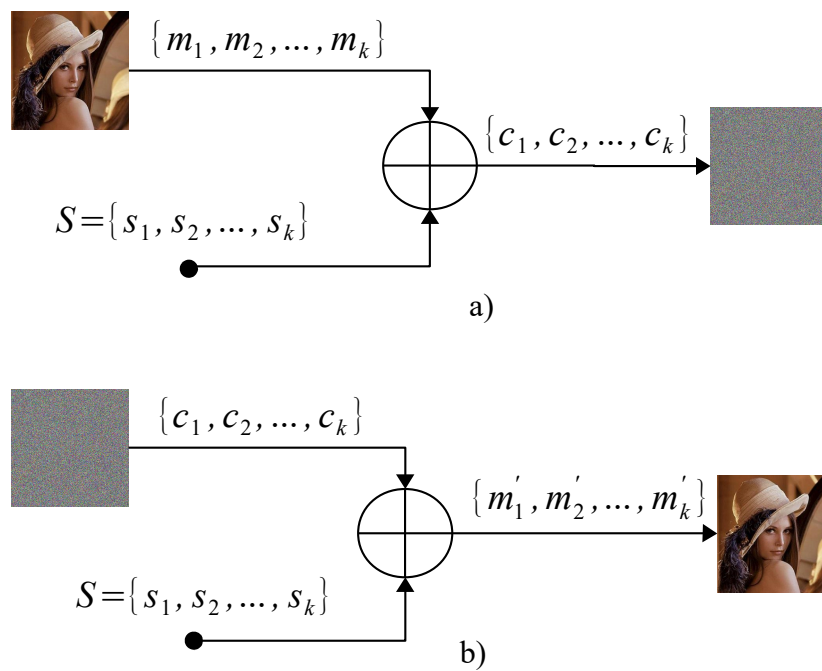


Figura 10: Método de cifrado en flujo con operador XOR.

que a manera de ejemplo, se emplea el mapeo caótico de Tinkerbell descrito por el sistema de ecuaciones 1 de la sección II.1. Por su parte, la figura 10b muestra el proceso inverso que recupera la imagen original a partir del criptograma $C = \{c_1, c_2, \dots, c_k\}$. La condicionante relevante para que este procedimiento no se vea alterado, es utilizar el mismo conjunto de secuencias caóticas binarias $S = \{s_1, s_2, \dots, s_k\}$ para ambos procesos.

Esta condicionante, es a su vez, una potencial desventaja para el encriptado de imágenes diferentes bajo el mismo conjunto de series cifrantes $S = \{s_1, s_2, \dots, s_k\}$. Al considerar por ejemplo dos imágenes digitales a color distintas: $M_a = \{m_1a, m_2a, \dots, m_ia\}$, "Lena.png" de 256×256 pixeles y $M_b = \{m_1b, m_2b, \dots, m_ib\}$, "Loto.png" de 256×256 pixeles encriptadas cada una bajo el mismo conjunto de secuencias cifrantes $S = \{s_1, s_2, \dots, s_k\}$ se obtienen los criptogramas $C_a = \{c_1a, c_2a, \dots, c_ia\}$ y $C_b = \{c_1b, c_2b, \dots, c_ib\}$ como se muestran en la figura 11a y 11b respectivamente. Cuando a estos criptogramas se les aplica la operación binaria $C_a XOR C_b$ como se representa en la figura 10, se puede inferir información relativa a las imágenes originales como se evidencia en la figura 11c. Bajo este esquema, se propone un método de encriptado en el que se elimine la vulnerabilidad del método al utilizar series cifrantes diferentes generadas a partir de claves de cifrado compuestas que utilicen condiciones iniciales con valor pseudoaleatorio, tal y como se describió en la sección II.2.3.

III.2.2 Conformación de la clave de encriptado.

La clave de cifrado definida por (CL) utilizada en el método propuesto, se compone de los valores numéricos que corresponden a los parámetros de control del sistema caótico, los valores en sus condiciones iniciales, el valor de la precisión a establecer en el cálculo numérico np , el valor de la precisión con la que se redondea a las nuevas condiciones iniciales np_{x_0} y el valor ret . El valor ret , es una cantidad de iteraciones del mapeo caótico que no se consideran para los valores de la serie a generar, debido

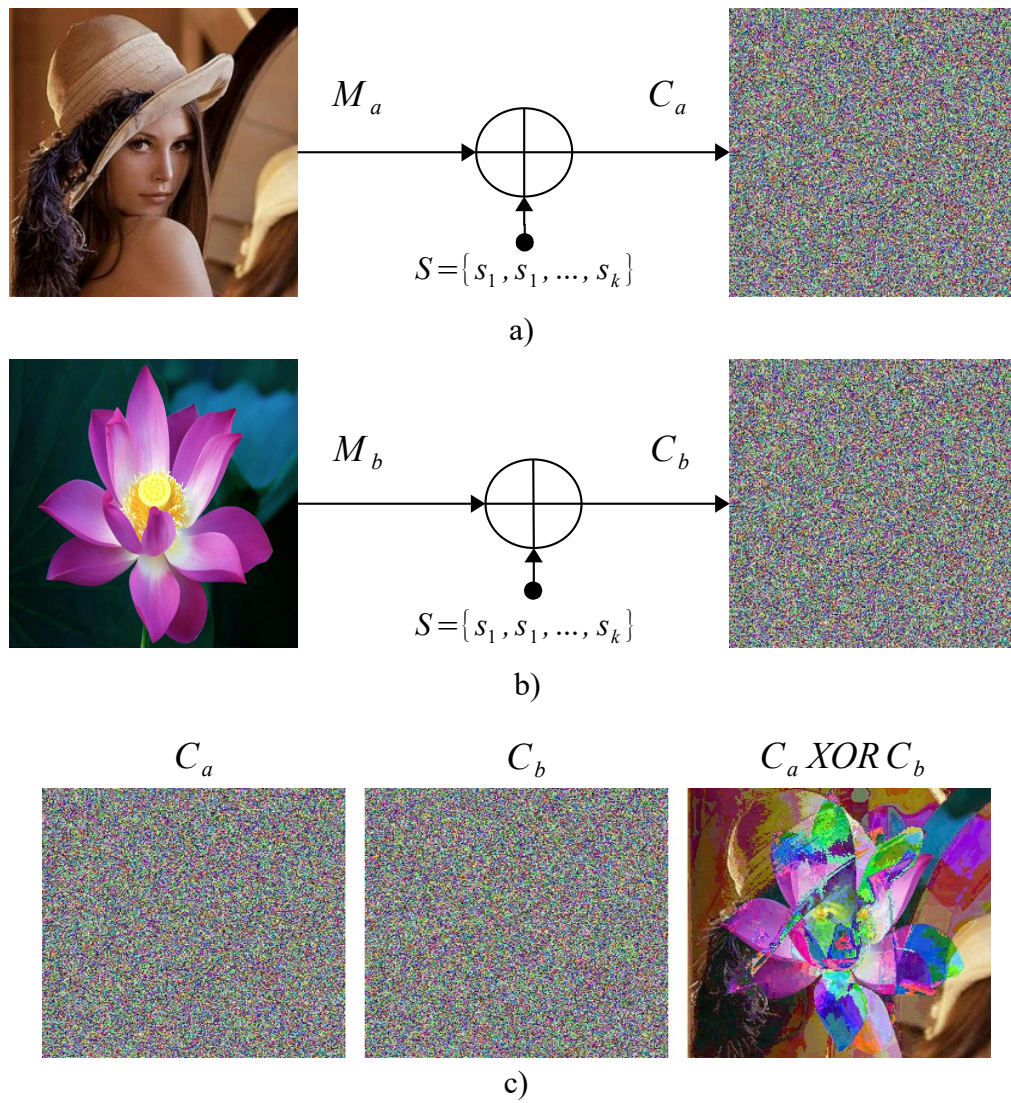
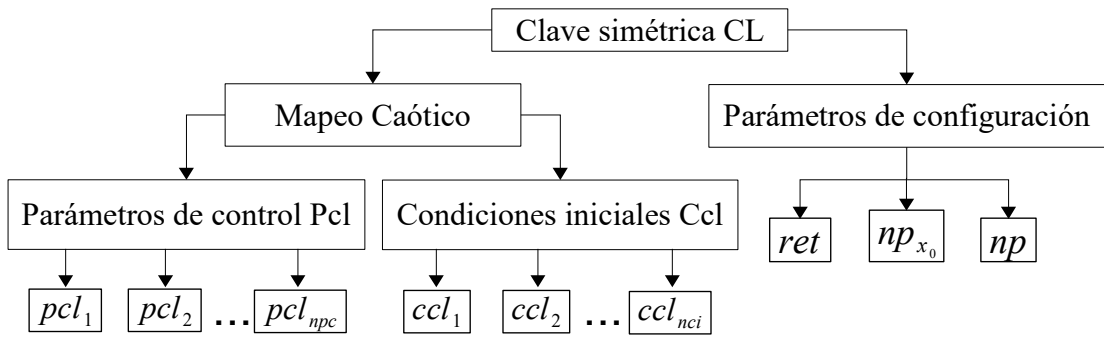


Figura 11: Efecto de utilizar la misma serie cifrante en el encriptado de dos imágenes diferentes.

a que las trayectorias caóticas alcanzan una mayor pseudoaleatoriedad después de un cierto número de iteraciones, tal y como se demuestra en la sección II.2.2. La figura 12, muestra un esquema de la conformción de la clave de cifrado CL , donde el conjunto de valores de los parámetros de control está representado por $Pcl = \{pcl_1, pcl_2, \dots, pcl_{npc}\}$ con npc = número de parámetros de control del mapeo caótico. De la misma manera, el conjunto de condiciones iniciales está representado por $Ccl = \{ccl_1, ccl_2, \dots, ccl_{nci}\}$ con nci = número de condiciones iniciales.



$$CL = \{ pcl_1, pcl_2, \dots, pcl_{npc}, ccl_1, ccl_2, \dots, ccl_{nci}, np_{x_0}, ret, np \}$$

Figura 12: Composición de la clave de cifrado utilizando gran precisión numérica.

La longitud de los valores numéricos de precisión np , np_{x_0} y del valor de retardo de iteraciones ret que se consideran en este caso, son de tipo entero de 16 bits. La longitud de los valores en los parámetros de control y de los valores en las condiciones iniciales considerando gran precisión, está determinada por $\log_2(1 \times 10^{nd})$, por lo tanto, la longitud máxima de la clave de cifrado, es la suma del número de bits resultantes. El número de bits ncl_b en la clave CL se puede representar mediante la ecuación 4.

$$ncl_b = (npc + nci)\log_2(1 \times 10^{nd}) + 48 \quad (4)$$

De esta forma, el espacio de claves (S_{cl}) en términos de bits que proporciona el método de encriptado propuesto, considerando un mapeo caótico y el establecimiento

de la precisión numérica en np decimales significativos, queda determinado mediante la ecuación 5.

$$S_{cl} = 2^{ncl_b} = 2^{(npc+nci)\log_2(1 \times 10^{n_d})+48} \quad (5)$$

III.3 Conclusiones

El método para encriptado y desencriptado de imágenes propuesto, permite utilizar el mismo hardware para ambos procesos, dependiendo solamente de los datos de entrada. Al implementar el generador de gran precisión propuesto en el método de encriptado y desencriptado de imágenes, se observa que el espacio de claves que proporciona se incrementa exponencialmente en comparación con los espacios de clave que proporcionan los métodos criptográficos que implementan precisión simple y doble precisión.

Capítulo IV

Sistema criptográfico de imágenes a color con procesamiento en paralelo

Un objetivo fundamental de este trabajo de tesis, es realizar un encriptamiento de secuencia de imágenes de alta resolución, por lo tanto, es necesario que el encriptado de cada imagen se realice en el menor tiempo posible. Por lo anterior se propone un método que realiza el proceso de encriptado y desencriptado de una imagen de alta resolución, implementando técnicas de perfilación y de procesamiento en paralelo, con el propósito de optimizar el tiempo total de encriptado en imágenes de alta resolución.

IV.0.1 Perfilación de programas en Python

Un perfilador, es un programa que analiza y recolecta información sobre el comportamiento de otro programa objeto durante su ejecución. El tipo de información que se analiza, incluye los tiempos de procesamiento de los procedimientos del programa y el número de veces que se llama a cada procedimiento. Lo anterior, con el propósito de permitir al desarrollador optimizar el código del programa para mejorar la velocidad en la ejecución del programa completo, ajustando el diseño del código implementando técnicas de ajuste y, si es posible, la implementación de técnicas de procesamiento en paralelo. La herramienta para perfilar que se utiliza en el presente trabajo de investigación es la herramienta de perfilación "línea por línea" conocida como **Line_Profiler** para Python. **Line_Profiler** se auxilia de la etiqueta **@profile** para identificar la función del programa a perfilar y realizar el análisis correspondiente durante la ejecución del programa. La función *encripta()*, definida en el programa "Encriptado.py",

corresponde al método de encriptado propuesto en la sección III.2.1. Se observa que la etiqueta **@profile**, se encuentra una línea antes de la definición de la función.

```
@profile
def encripta():
    foto = Image.open('entrada.png')
    foto = np.array(foto)
    ren = foto.shape[0]
    col = foto.shape[1]
    cap = foto.shape[2]
    vector=ren*col*cap
    foto=np.reshape(foto,(vector,1))
    a= Decimal('0.9')
    b= Decimal('-0.6013')
    c= Decimal('2.0')
    d= Decimal('0.5')
    x0= Decimal('-0.72')
    y0= Decimal('-0.64')
    ret=0
    npx0=8
    np_prec=99
    getcontext().prec=npx0
    while (ret<1):
        x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
        y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
        x0=x1
        y0=y1
        ret=ret+1
    getcontext().prec=np_prec
    j=0
    while(j<vector):
        x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
        y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
        x0=x1
        y0=y1
        x1=x1*Decimal(10**np_prec)
        x1=Decimal.to_integral(x1)
        if x1<0:
            x1=x1*(-1)
        s1=int(x1%255)
        foto[j]= foto[j]^s1
        j=j+1
    foto=np.reshape(foto,(ren,col,cap))
    foto=Image.fromarray(foto)
    foto.save('salida.png')
```

La ejecución del programa "Encriptado.py" con la herramienta de perfilado se realiza mediante el comando:

kernprof -l -v Encriptado.py

El parámetro **-v** del comando kernprof, permite observar el reporte del análisis de perfilado como respuesta inmediata a la ejecución del comando anterior en la terminal de comandos. De otra forma, la herramienta de perfilado guarda la información del análisis correspondiente en un archivo binario con extensión **.lprof** con el mismo nombre del archivo de código python. El reporte completo se puede observar ejecutando el archivo de reporte mediante:

python -m line_profiler Encriptado.py.lprof

El siguiente reporte de ejemplo, corresponde al análisis de perfilado de la función *encripta()* considerando como imagen objeto el archivo `"""Paisaje.png"""`. La imagen a encriptar es una imagen a color de alta resolución en composición de 32 bits de 2560×1600 pixeles, es decir, cada pixel de la imagen se compone de 4 elementos de 8 bits dando un total de 16,384,000 elementos de información por encriptar.

Wrote profile results to Encriptado.py.lprof
Timer unit: 3.79968e-07 s

Total time: 862.992 s
File: Encriptado.py
Function: encripta at line 4

Line #	Hits	Time	Per Hit	% Time	Line Contents
4					@profile
5					def encripta():
6	1	146078	146078.0	0.0	foto = Image.open('Pai...
7	1	761832	761832.0	0.0	foto = np.array(foto)
8	1	63	63.0	0.0	ren = foto.shape[0]
9	1	7	7.0	0.0	col = foto.shape[1]
10	1	6	6.0	0.0	cap = foto.shape[2]
11	1	8	8.0	0.0	vector=ren*col*cap
12	1	97	97.0	0.0	foto=np.reshape(foto,(v...
13	1	98	98.0	0.0	a= Decimal('0.9')
14	1	11	11.0	0.0	b= Decimal('-0.6013')
15	1	9	9.0	0.0	c= Decimal('2.0')
16	1	9	9.0	0.0	d= Decimal('0.5')
17	1	8	8.0	0.0	x0= Decimal('-0.72')

18	1	8	8.0	0.0	y0= Decimal('-0.64')
19	1	5	5.0	0.0	ret=0
20	1	4	4.0	0.0	npx0=8
21	1	4	4.0	0.0	np_prec=99
22	1	15	15.0	0.0	getcontext().prec=npx0
23	2	12	6.0	0.0	while (ret<1):
24	1	69	69.0	0.0	x1 = Decimal((x0*x0)...
25	1	14	14.0	0.0	y1 = Decimal((2*x0*y...
26	1	5	5.0	0.0	x0=x1
27	1	6	6.0	0.0	y0=y1
28	1	6	6.0	0.0	ret=ret+1
29	1	6	6.0	0.0	getcontext().prec=np_prec
30	1	6	6.0	0.0	j=0
31	16384001	78137141	4.8	3.4	while(j<vector):
32	16384000	439917303	26.9	19.4	x1 = Decimal((x0*...
33	16384000	351247125	21.4	15.5	y1 = Decimal((2*x...
34	16384000	87992069	5.4	3.9	x0=x1
35	16384000	79512478	4.9	3.5	y0=y1
36	16384000	300653689	18.4	13.2	x1=x1*Decimal(10**...
37	16384000	136258513	8.3	6.0	x1=Decimal.to_inte...
38	16384000	91137986	5.6	4.0	if x1<0:
39	10906047	83757897	7.7	3.7	x1=x1*(-1)
40	16384000	185584508	11.3	8.2	s1=int(x1%255)
41	16384000	337097363	20.6	14.8	foto[j]= foto[j]s1
42	16384000	93981721	5.7	4.1	j=j+1
43	1	77	77.0	0.0	foto=np.reshape(foto,(ren,c...
44	1	456	456.0	0.0	foto=Image.fromarray(foto)
45	1	5036187	5036187.0	0.2	foto.save('salida.png')

En el archivo de reporte, se observa que el tiempo total de ejecución de la función *encripta()* es de 862.992 segundos. Adicionalmente el reporte indica el tiempo de ejecución de cada instrucción por línea de código de programa, en este caso, la unidad de medida que considera el reporte es de $3.79968e-07$ segundos. De los resultados obtenidos en la sección V.1.2 se tiene que la serie cifrante obtenida del estado y_{n+1} presenta altos niveles de aleatoriedad y es potencialmente viable para usarse como serie cifrante, por lo tanto si en el método propuesto se incluye la serie cifrante de y_{n+1} se estima que la función *Encripta()* reduzca a la mitad, el porcentaje de tiempo correspondiente a las iteraciones del mapeo caótico. Los ajustes a realizar son los siguientes:

```
...
while(j<vector):
    x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
    y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
```

```

x0=x1
y0=y1
x1=x1*Decimal(10**np_prec)
x1=Decimal.to_integral(x1)
if x1<0:
    x1=x1*(-1)
s1=int(x1%255)
y1=y1*Decimal(10**np_prec)
y1=Decimal.to_integral(y1)
if y1<0:
    y1=y1*(-1)
s2=int(y1%255)
foto[j]= foto[j]^s1
foto[j+1]= foto[j+1]^s2
j=j+2
...

```

Al realizar el análisis con la herramienta de perfilado una vez hecho el cambio en la función *encripta()*, se tiene que el tiempo de ejecución total es de 639.005 segundos. La figura 13 muestra el criptograma resultante al implementar el metodo de encriptamiento con el generador HPG considerando los estados x_{n+1} y y_{n+1} del mapeo de Tinkerbell para componer la serie cifrante. Se observa que el criptograma es una imagen totalmente ininteligible y no muestra vestigios de la imagen original.

Con propósitos experimentales y de comparación, se realizó el encriptado de la imagen "Paisaje.png" con formato de color de 4 canales y ajustada a una resolución de 1024×1024 pixeles. La tabla II muestra los resultados de el valor de entropía de la información obtenidos del criptograma resultante utilizando el metodo criptográfico propuesto y considerando el estado x_{n+1} e y_{n+1} para generar la serie cifrante, comparados con los valores de entropía reportados en la tabla XIII.

Tabla II: Valores de entropía implementando HPG con x_{n+1} e y_{n+1} del mapeo de Tinkerbell, $np = 99$ y considerando la imagen "Paisaje.png".

HPG Tinkerbell estado caótico	Entopía por componente (bit/símbolo)		
	R	G	B
x_{n+1}	7.99979	7.99979	7.99983
x_{n+1} e y_{n+1}	7.99983	7.99978	7.99973

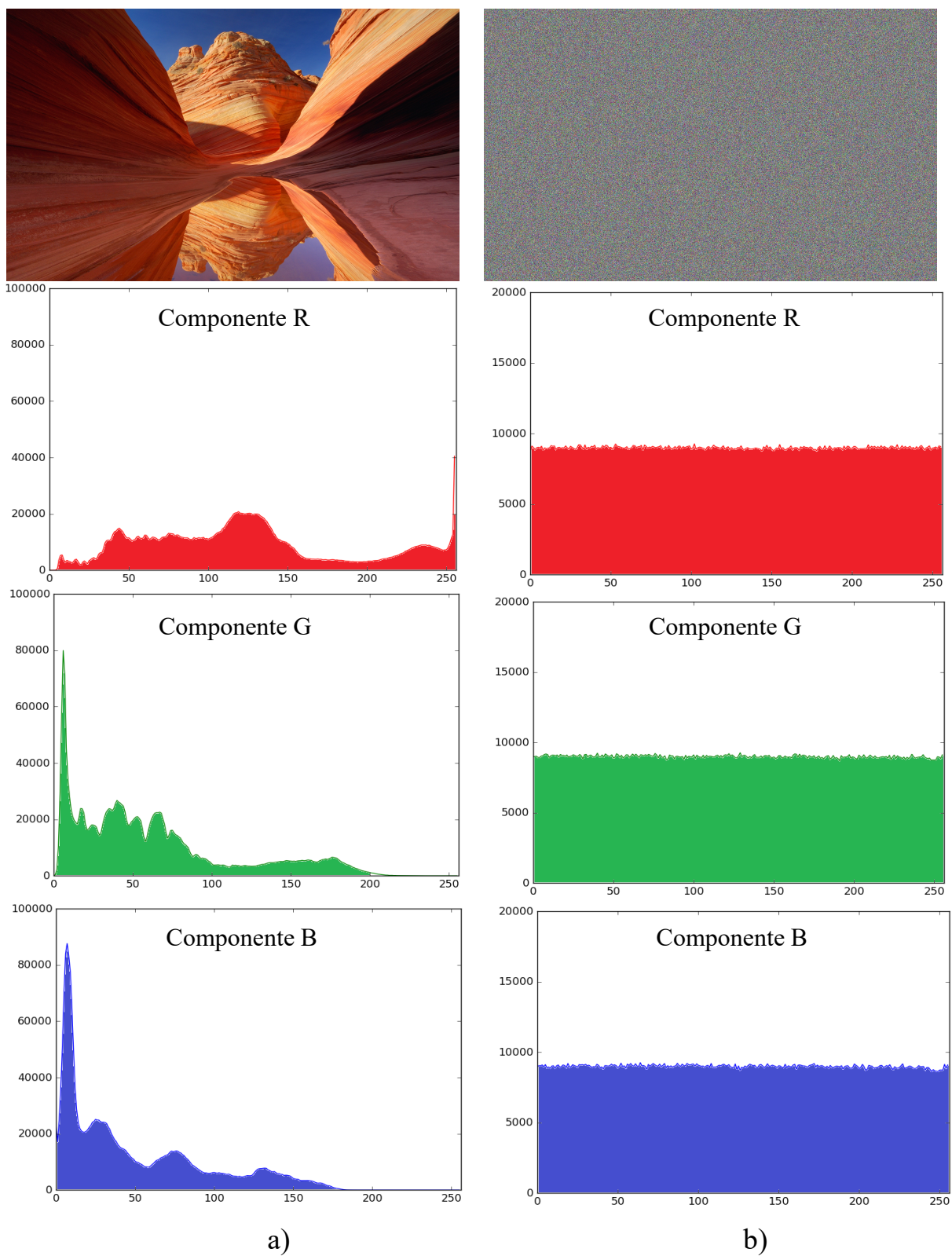


Figura 13: Imagen y criptograma obtenido al utilizar HPG propuesto con el mapeo de Tinkerbell con ambos estados caóticos en el método de encriptado.

Con la información obtenida mediante la herramienta de perfilado, se observa cuales instrucciones se realizan una mayor cantidad de veces y se ejecutan en una mayor cantidad de tiempo, con el propósito de implementar técnicas de ajuste de optimización de código y dividir la función en una parte con procesamiento secuencial y otra paralelizable.

IV.0.2 Parámetros de procesamiento en paralelo

Del análisis de perfilación se determina que la función se puede dividir en una parte secuencial y una parte paralelizable. Existen una serie de parámetros para evaluar el procesamiento en paralelo dentro de los cuales se encuentra, el factor de mejora de rendimiento o "sepeed-up" y la eficiencia del rendimiento. Si se define a $O(n)$ como el número de operaciones elementales realizadas por un sistema con n procesos y a $T(n)$ como el tiempo de ejecución en pasos unitarios de tiempo, entonces $T(n) < O(n)$, si los n procesadores realizan más de una operación por unidad de tiempo. El factor de mejora del rendimiento $S(n)$ para n procesadores se define mediante la ecuación 6,

$$S(n) = T(1)/T(n) \quad (6)$$

y la eficiencia del sistema $E(n)$, para un sistema con n procesadores, está determinada por la ecuación 7,

$$E(n) = S(n)/n = T(1)/n * T(n) \quad (7)$$

La eficiencia del sistema mas baja $E(n) \rightarrow 0$ sucede cuando todas las instrucciones en el programa objeto se ejecutan de manera secuencial en un sistema de un único procesador. La eficiencia del sistema máxima $E(n) = 1$ corresponde al caso en el que todos los procesadores del sistema estan siendo completamente utilizados durante la ejecución del programa. Otro parámetro de evaluación de sistemas con procesamiento

paralelo es la escalabilidad. Se dice que un sistema es escalable para un determinado número de procesadores n , si la eficiencia del sistema $E(n)$ es constante y en todo momento mayor a un factor de 0.5 (Hwang y Jotwani, 2011). En la práctica, los sistemas escalables se pueden dividir en un determinado número de procesos n a partir del cual la eficiencia del sistema comienza a disminuir. En el programa analizado, del total de las instrucciones se tiene que, el 99.7% es potencialmente paralelizable (líneas 13 a la 42) y el 0.3% se ejecuta de forma secuencial; Por lo tanto, si $T(1) = 639s$ los valores de $S(n)$ y $E(n)$ para $n = 2$ son:

$$T(1) = 639s = 99.7\% + 0.3\% = 637.08s + 1.92s$$

$$S(2) = T(1)/T(2) = 639s/(318.54s + 1.92s) = 639s/320.46s = 1.994$$

$$E(2) = S(2)/2 = T(1)/2 * T(2) = 1.9940/2 = 0.9970$$

(8)

Una primer aproximación al número de procesadores n entre los cuales se puede dividir el programa analizado, se puede establecer a partir de la ecuación 7; esto es, si se tiene que:

$$E(n) = S(n)/n = T(1)/n * T(n)$$

$$E(n) = 50\% = 0.5$$

donde :

$$T(1) = 100 = Ts + Tp(n) = 0.3 + 99.7$$

$$T(n) = Ts + Tp(n) = Ts + (Tp(1)/n) = Ts + (100 - Ts)/n$$

entonces,

$$0.5 = 100/(n(Ts + (100 - Ts)/n))$$

$$0.5 = 100/((n - 1)Ts + 100)$$

$$n - 1 = ((100/0.5) - 100)/Ts$$

$$n = 100/Ts + 1 = 334.33$$

(9)

La figura 14 muestra la gráfica de los valores de eficiencia del sistema $E(n)$ del programa "Encriptado.py", se observa que para $n > 334$ la eficiencia del sistema es menor a 0.5.

El factor de rendimiento, se puede clasificar en tres tipos diferentes, i) con base en que la carga o el tamaño de problema es fijo y lo que se pretende reducir, es el tiempo de ejecución del problema (Hwang y Jotwani, 2011); ii) con el fundamento en que se cuenta con un tiempo determinado para ejecutar el problema o que se pretende aumentar el tamaño de la máquina (Hwang y Jotwani, 2011); y una tercer clasificación, que se aplica a problemas escalables donde la principal limitante en el proceso de ejecución es la memoria del sistema (Hwang y Jotwani, 2011). En el presente trabajo de investigación, se considera un problema en el cual la carga o tamaño de problema es fijo y lo que se pretende, es reducir el tiempo de ejecución del proceso. Por lo anterior, es necesario

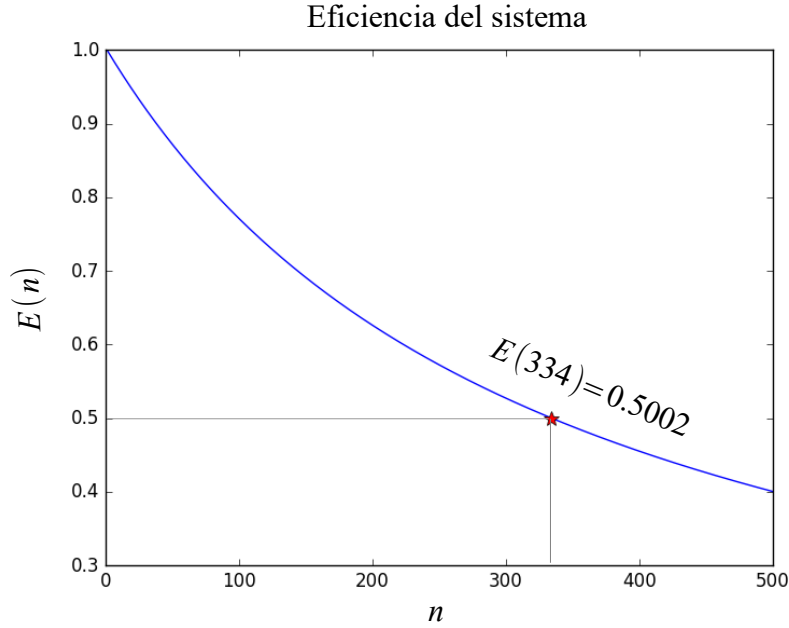


Figura 14: Curva característica de la eficiencia del sistema $E(n)$ del programa "Encriptado.py" con $\alpha = 0.3\%$.

realizar un análisis del factor de rendimiento con base en la ley de Amdahl (Gene, 1967). La ley de Amdahl, considera problemas en donde el objetivo es distribuir la carga fija en mas procesadores para disminuir el tiempo total de ejecución. En este caso, el factor de mejora del rendimiento Amdahl (S_n) descrito por la ecuación 10, esta determinado en función del porcentaje del tiempo secuencial del programa al cual se le identifica como α . Al porcentaje de tiempo paralelizable se le indentifica con $\beta = 1 - \alpha$.

$$S_n = n / (1 + (n - 1)\alpha) \quad (10)$$

En el programa "Encriptado.py", el porcentaje de tiempo secuencial $\alpha = 0.3\%$ implica que S_n se aproxima asintóticamente a $1/\alpha$ a medida que se incrementa el número de procesadores n . La gráfica de la figura 15 muestra la curva caraterística del factor de rendimiento Amdahl considerando una parte secuencial α de 0.3% en comparación con la curva característica cuando la parte paralelizable es nula o $\alpha = 0\%$. Adicionalmente y con propósitos de comparación, se muestran las curvas para diferentes valores de α en

donde se observa, que a medida que se incrementa el porcentaje de tiempo secuencial, el valor sped-up del factor de rendimiento Amdahl comienza a decaer en gran medida.

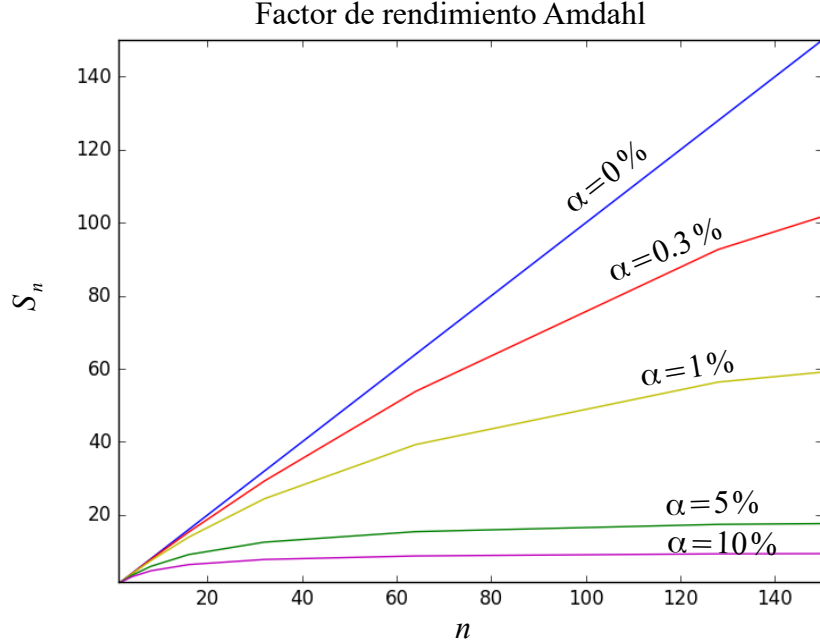


Figura 15: Curvas del factor de rendimiento S_n para diferentes valores α en comparación con la curva de S_n para el programa "Encriptado.py" con $\alpha = 0.3\%$.

En el análisis realizado, se considera que los procesadores, se encuentran bajo condiciones ideales de recurso computacional de software y hardware, es decir, al momento de ejecutar el programa cada vez, considerando diferentes números de procesadores, cada procesador funciona bajo las mismas condiciones y con los mismos recursos computacionales. En la realidad, otro de los factores que se consideran en el estudio del procesamiento en paralelo, es la comunicación entre procesadores. En el programa "Encriptado.py", la carga o tamaño del problema, corresponde a la imagen que se va a encriptar o desencriptar según sea el caso. Considerando que al realizar el proceso de paralelización, la carga se distribuye equitativamente entre el total de los procesadores que ejecutan el programa, entonces, se produce un tiempo de distribución de la imagen

en partes equitativas para ser encriptadas y un tiempo de recolección de las partes encriptadas para obtener el criptograma resultante. Debido a ello, el tiempo total en la ejecución del programa se ve afectado por la distribución y recolección de la carga. En la actualidad, existen herramientas que realizan comunicación colectiva optimizando al máximo los tiempos de distribución y recolección de la carga entre los procesadores, tal es el caso de la herramienta "Message Passing Interface" (MPI) para Python, que proporciona funciones de comunicación colectiva que optimizan los tiempos de distribución y recolección de datos e información entre procesadores (Dalcin *et al.*, 2011, 2005).

IV.1 Método propuesto con procesamiento en paralelo

El sistema para el encriptado y desencriptado de imágenes a color que se propone en el presente trabajo de investigación y que se muestra en la Figura 16, tiene su principal fundamento en el método de cifrado en flujo con clave simétrica (Menezes *et al.*, 1996) y el método propuesto en la sección III, en la generación de caos con gran precisión numérica estudiada en la sección II y en el procesamiento de datos en paralelo con comunicación colectiva (Dalcin *et al.*, 2011, 2005), implementados en un sistema embebido basado en microprocesador SoC Raspberry Pi 3.

En el sistema propuesto, una imagen a color es considerada como un mensaje objeto a encriptar para generar un criptograma y enviarlo a un usuario receptor. Mediante una clave de cifrado y el generador HPG propuesto, se genera una serie cifrante pseudoaleatoria única de gran potencialidad para encriptar el mensaje objeto. Con el propósito de encriptar imágenes de gran resolución en el menor tiempo posible, los elementos de la imagen se distribuyen en sistemas equivalentes para su procesamiento de datos en paralelo. El usuario receptor, utiliza el mismo método para recuperar el mensaje original a partir del criptograma. Cada elemento de $M = \{m_1, m_2, \dots, m_n\}$ es modificado

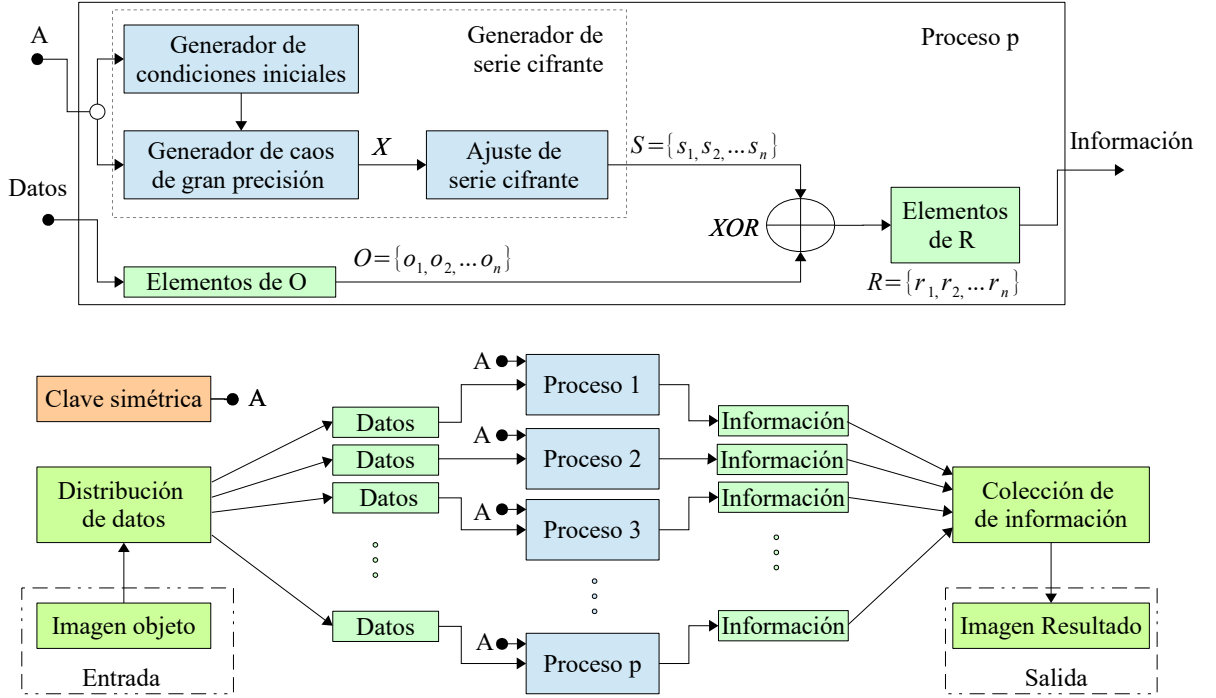


Figura 16: Diagrama de bloques de método de encriptado de imágenes con procesamiento paralelo.

por un correspondiente elemento de una serie cifrante $S = \{s_1, s_2, \dots, s_n\}$ mediante una operación de encriptado para obtener un criptograma resultante $C = \{c_1, c_2, \dots, c_n\}$.

Para recuperar el mensaje original a partir del criptograma C , se requiere que el método de desencriptado realice una función inversa al método de encriptado; es decir, cada elemento de C es modificado por el correspondiente elemento de la serie S mediante una operación de desencriptado para obtener una serie $\hat{M} = M$. Por lo tanto, es necesario que la serie S sea simétrica en ambos métodos. Considerando imágenes como mensaje objeto, en donde cada elemento del mensaje corresponde a un pixel de la imagen y el operador XOR como operador de cifrado, si se genera una secuencia cifrante simétrica S con n número de elementos del mismo tipo que m_n , entonces $C = XOR(M, S)$ y $\hat{M} = XOR(C, S)$ con $M = \hat{M}$. Al utilizar el operador XOR como operador de cifrado, se puede utilizar el mismo método de cifrado como método de

descifrado tal y como se muestra en la Figura 16, en donde se tiene que, si $O = M$ entonces $R = C$ y si $O = C$, por lo tanto $R = \hat{M}$.

IV.1.1 Implementación de MPI utilizando comunicación colectiva en programa "Encriptado.py"

La sección de código de programa que se muestra a continuación, describe la configuración del sistema de procesadores que intervienen en la ejecución del programa "Encriptado.py", en complemento con una parte del procesamiento secuencial del programa α , esta parte del procesamiento secuencial, es la que se encarga de la lectura desde archivo de la imagen a encriptar o desencriptar según sea el caso, indentificado con el nombre de "entrada.png" y del reacomodo de los datos de forma equitativa para su distribución mediante comunicación colectiva.

```
from mpi4py import MPI
import numpy as np
from PIL import Image
from decimal import *

comm=MPI.COMM_WORLD
size = comm.Get_size()    #número total de procesadores
rank = comm.Get_rank()    #identificador del procesdor
if rank == 0:
    imagen = Image.open('entrada.png')    #lee imagen de entrada
    imagen=np.array(imagen)    #convierte el objeto imagen en arreglo
    ren = imagen.shape[0]
    col = imagen.shape[1]    #dimensines del arreglo
    cap = imagen.shape[2]
    n_ren=np.int(ren/size)    #redimensionamiento del arreglo
    foto=np.reshape(imagen,(size,n_ren,col,cap)) #para su distribución
```

La variable *size* identifica el número total de procesadores que intervienen en la ejecución del programa y la variable *rank*, identifica a cada procesador. La lectura de la imagen, se realiza mediante la función "Image.open()" proporcionada por la herramienta de procesamiento de imágenes para python *PIL*; La función "Image.open()" regresa un objeto de la clase "Image", por lo tanto, se realiza una conversión del objeto a

tipo "array" con el propósito de obtener un arreglo matricial para su procesamiento con la herramienta "numpy". Las instrucciones antes mencionadas son ejecutadas por un primer procesador al cual se le identifica como $rank = 0$ y hace las funciones de procesador principal; dicho de otra forma, el procesador principal es el que se encarga de la lectura de los datos a procesar, de la distribución equitativa de los datos entre los diferentes procesadores y de la recolección de la información para proporcionar el resultado final.

Una vez realizada la lectura y distribución de los datos, cada procesador (incluyendo el procesador principal), realiza las instrucciones del método criptográfico propuesto para el encriptado o desencriptado de la imagen, mostradas en el código de programa a continuación.

```
#_____Sección [clave simétrica]_____
a= Decimal('0.9')
b= Decimal('-0.6013')
c= Decimal('2.0')
d= Decimal('0.5')
x0= Decimal('-0.72')
y0= Decimal('-0.64')
ret=0
npx0=8
np_prec=99

#____Sección [condiciones iniciales pseudoaleatorias] _____
getcontext().prec=npx0
while (ret<rank+1):
    x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
    y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
    x0=x1
    y0=y1
    ret=ret+1

#_____Sección [distribución de datos]_____
foto = comm.scatter(foto,root=0)
ren_local=foto.shape[0]
col_local=foto.shape[1]
cap_local=foto.shape[2]
vector=ren_local*col_local*cap_local
foto=np.reshape(foto,(vector,1))

#_____Sección [Generador HPG mapeo de Tinkerbell]_____
getcontext().prec=np_prec
```

```

s1=0
s2=0
j=0
while(j<vector):
    x1 = Decimal((x0*x0)-(y0*y0)+(a*x0)+(b*y0))
    y1 = Decimal((2*x0*y0)+(c*x0)+(d*y0))
    x0=x1
    y0=y1

    #___Sección [ajuste de serie cifrante]_____
    x1=x1*Decimal(10**np_prec)
    y1=y1*Decimal(10**np_prec)
    x1=Decimal.to_integral(x1)
    y1=Decimal.to_integral(y1)
    if x1<0:
        x1=x1*(-1)
    if y1<0:
        y1=y1*(-1)
    s1=int(x1%255)
    s2=int(y1%255)

    #___Sección [cifrado en flujo con XOR]_____
    foto[j]= foto[j]^s1
    foto[j+1]= foto[j+1]^s2
    j=j+2

#_____Sección [recolección de información]_____
foto=np.reshape(foto,(ren_local,col_local,cap_local))
newData = comm.gather(foto,root=0)

```

En la sección [clave simétrica], se establecen los valores correspondientes a la clave de cifrado y en la sección [condiciones iniciales pseudoaleatorias], se generan las condiciones iniciales para cada procesador de manera independiente, con el propósito de que cada procesador cuente con una clave de cifrado distinta. En la sección [distribución de datos] , se utiliza la función *scatter()* con la finalidad de optimizar el tiempo en la distribución de los datos; adicionalmente, una vez realizada la distribución equitativa, cada procesador cuenta con la parte correspondiente de los datos a encriptar en un arreglo matricial; por lo tanto, para propósitos de optimizar el método criptográfico, se reacomodan los datos en un arreglo unidimensional. Posteriormente, se ejecutan las instrucciones de las secciones [Generador HPG],[ajuste de serie cifrante] y [cifrado en flujo con XOR]. Una vez realizado el proceso criptográfico se ejecutan las instrucciones de la

sección [recolección de información] para enviar al procesador principal el resultado; el envío de información se realiza mediante la función de recolección de datos "gather()". Es importante resaltar, que las funciones "gather()" y "scatter()" funcionan de manera correspondiente, es decir, el formato de datos de envío debe ser igual al fomato de recepción, por lo anterior, es necesario realizar un reacomodo del vector resultante a su formato matricial original.

Finalmente, el procesador principal ($rank = 0$), realiza la recolección de la información y convierte el arreglo matricial resultante en un objeto de la clase "Image" para almacenar el resultado final en un archivo identificado como "salida.png", lo anterior se realiza mediante las intrucciones de la sección del código del programa mostrado a continuación.

```
if rank == 0:
    foto=np.array(newData)
    foto=np.reshape(foto,(ren,col,cap))
    foto=Image.fromarray(foto)
    foto.save('salida.png')
    fin=datetime.now()
    print("hecho!")
    print(fin-inicio)
if __name__ == '__main__':
    encripta()
```

IV.1.2 Máquinas de procesadores Raspberry Pi 3 y Clúster FIAD

Otro de los aspectos que se consideran en el estudio del procesamiento en paralelo, es la máquina de procesadores en la que se ejecuta el programa. La ejecución del programa, implementando procesamiento en paralelo con MPI y la herramienta de perfilado se realiza mediante el comando

```
mpiexec -n < n > kernprof -l -v Encriptado.py
```

donde el parámetro $< n >$ indica el número de procesadores que contiene la máquina virtual que ejecuta el programa. La figura 17 muestra la composición de la máquina

Raspberry Pi 3 en donde se observa que de manera interna cuenta con una estructura QuadCore con bus de comunicación interna y 1GB de memoria RAM.

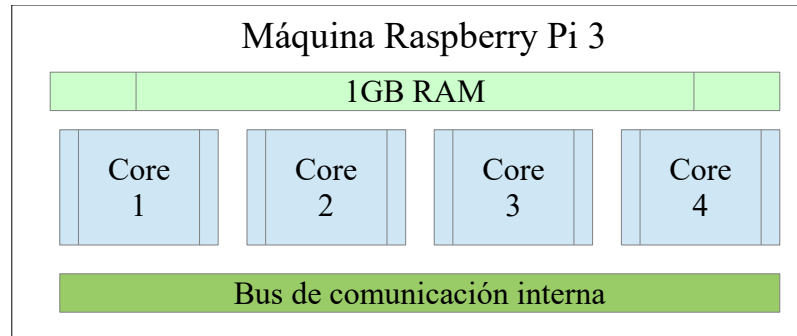


Figura 17: Máquina Raspberry Pi 3.

Al ejecutar el comando, el sistema Raspberry Pi 3, crea una máquina virtual con n procesadores para la ejecución del programa. La figura 18, muestra algunas de las aplicaciones y servicios que están en ejecución en el sistema Raspberry Pi 3 al momento de ejecutar el comando para procesamiento con 1 procesador, se observa que se crea una máquina virtual con un recurso de $\%CPU = 100.0$ y un recurso de memoria de $\%MEM = 2.6$ en un proceso identificado como $PID = 1542$.

File Edit Tabs Help										
top - 07:04:35 up 9:47, 2 users, load average: 3.04, 4.45, 2.21										
Tasks: 141 total, 2 running, 137 sleeping, 2 stopped, 0 zombie										
%Cpu(s): 26.6 us, 0.1 sy, 0.0 ni, 73.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st										
KiB Mem: 850292 total, 339808 used, 510484 free, 26408 buffers										
KiB Swap: 102396 total, 0 used, 102396 free, 188104 cached Mem										
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
1542	pi	20	0	42408	21980	11296	R	100.0	2.6	0:31.10 python
490	root	20	0	153400	48304	23480	S	4.0	5.7	1:15.16 Xorg
1454	pi	20	0	5112	2480	2140	R	0.7	0.3	0:03.71 top
7	root	20	0	0	0	0	S	0.3	0.0	0:04.12 rcu_sched
450	nobody	20	0	2292	1464	1344	S	0.3	0.2	0:02.94 thd
729	pi	20	0	47748	19040	15980	S	0.3	2.2	0:08.37 lxterminal
1543	pi	20	0	5536	2644	2324	S	0.3	0.3	0:00.03 scrot
1	root	20	0	22876	3972	2744	S	0.0	0.5	0:06.83 systemd

Figura 18: Recurso computacional asignado a máquina virtual Raspberry Pi 3 con $n = 1$ procesador.

La figura 19a muestra la ejecución de la máquina virtual considerando $n = 4$ procesadores y la figura 19b, muestra el recurso computacional asignado a cada procesador cuando se ejecuta el método propuesto con $n = 8$ procesadores. Se observa que a cada procesador se le asigna un recurso computacional de CPU distinto al asignado a cada procesador cuando incrementa el número de procesadores de la máquina virtual. Por lo anterior, el factor de rendimiento Amdahl (S_n) calculado y mostrado en la gráfica de la figura 15, se ve afectado en gran medida debido a que las máquinas virtuales no cuentan con el mismo recurso computacional de hardware para su ejecución a medida que se incrementa el número de procesadores. Adicionalmente, el sistema operativo está optimizado para el diseño de aplicaciones científicas, por lo que cuenta con aplicaciones y servicios que se ejecutan en segundo plano, disminuyendo el recurso total de hardware que se asigna a la máquina virtual creada.

De la figura 19, el recurso computacional más óptimo para cada procesador, se presenta cuando el número de procesadores que ejecutan el programa es de $n = 4$. Por lo tanto, la máxima ganancia en tiempo de ejecución se obtiene con $n = 4$ procesadores; a partir de $n > 4$ procesadores la ganancia en tiempo de ejecución disminuye cada vez. Otra máquina que se considera en el presente trabajo de investigación conocida como Clúster FIAD, esta compuesta por 6 dispositivos Raspberry Pi 3 y cuya composición de manera general se describe en la figura 20. Se observa que al componerse la máquina completa por una composición de un total de 24 Cores, se espera que la máxima ganancia se obtenga en ejecuciones del programa con $np = 24$ procesadores. Sin embargo, además de que cada sistema Soc Raspberry Pi 3 cuenta con un bus de comunicación interno, cada SoC se comunica de manera externa con los restantes 5 dispositivos mediante un canal de comunicación Ethernet. Por lo anterior, se ve afectado el tiempo de comunicación entre procesos. Adicionalmente, los sistemas operativos instalados en cada SoC, están optimizados para el funcionamiento de programas con técnicas de

File Edit Tabs Help

```
top - 06:59:14 up 9:42, 2 users, load average: 2.76, 0.86, 0.30
Tasks: 146 total, 5 running, 139 sleeping, 2 stopped, 0 zombie
%Cpu(s): 98.9 us, 1.1 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 850292 total, 363788 used, 486504 free, 26172 buffers
KiB Swap: 102396 total, 0 used, 102396 free. 188912 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1459	pi	20	0	106408	20212	11244	R	99.2	2.4	1:13.06	python
1461	pi	20	0	106408	20128	11160	R	99.2	2.4	1:12.52	python
1460	pi	20	0	106408	20040	11072	R	98.2	2.4	1:13.02	python
1458	pi	20	0	107068	21484	11716	R	94.9	2.5	1:12.05	python
490	root	20	0	149364	44216	23416	S	4.6	5.2	1:03.42	Xorg
450	nobody	20	0	2292	1464	1344	S	0.7	0.2	0:02.34	thd

a)

File Edit Tabs Help

```
top - 07:02:17 up 9:45, 2 users, load average: 9.45, 4.88, 1.93
Tasks: 150 total, 9 running, 139 sleeping, 2 stopped, 0 zombie
%Cpu(s): 99.6 us, 0.4 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 850292 total, 407900 used, 442392 free, 26304 buffers
KiB Swap: 102396 total, 0 used, 102396 free. 192564 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1519	pi	20	0	106192	20516	11536	R	50.0	2.4	0:08.71	python
1517	pi	20	0	106192	20536	11556	R	49.7	2.4	0:08.87	python
1520	pi	20	0	106192	20628	11648	R	49.7	2.4	0:08.67	python
1523	pi	20	0	106192	20604	11624	R	49.7	2.4	0:08.79	python
1518	pi	20	0	106192	20680	11700	R	49.3	2.4	0:08.67	python
1522	pi	20	0	106192	20380	11400	R	49.3	2.4	0:08.84	python
1516	pi	20	0	106980	22588	12828	R	47.4	2.7	0:08.98	python
1521	pi	20	0	106192	20440	11460	R	47.4	2.4	0:08.62	python
490	root	20	0	152452	47496	23480	S	5.3	5.6	1:10.42	Xorg

b)

Figura 19: Recurso computacional asignado a máquina virtual Raspberry Pi 3: a) con $n = 4$ procesadores y b) con $n = 20$ procesadores.

procesamiento paralelo, es decir, ejecutan un mínimo de aplicaciones en segundo plano incrementando el recurso total de hardware asignado a la máquina virtual creada.

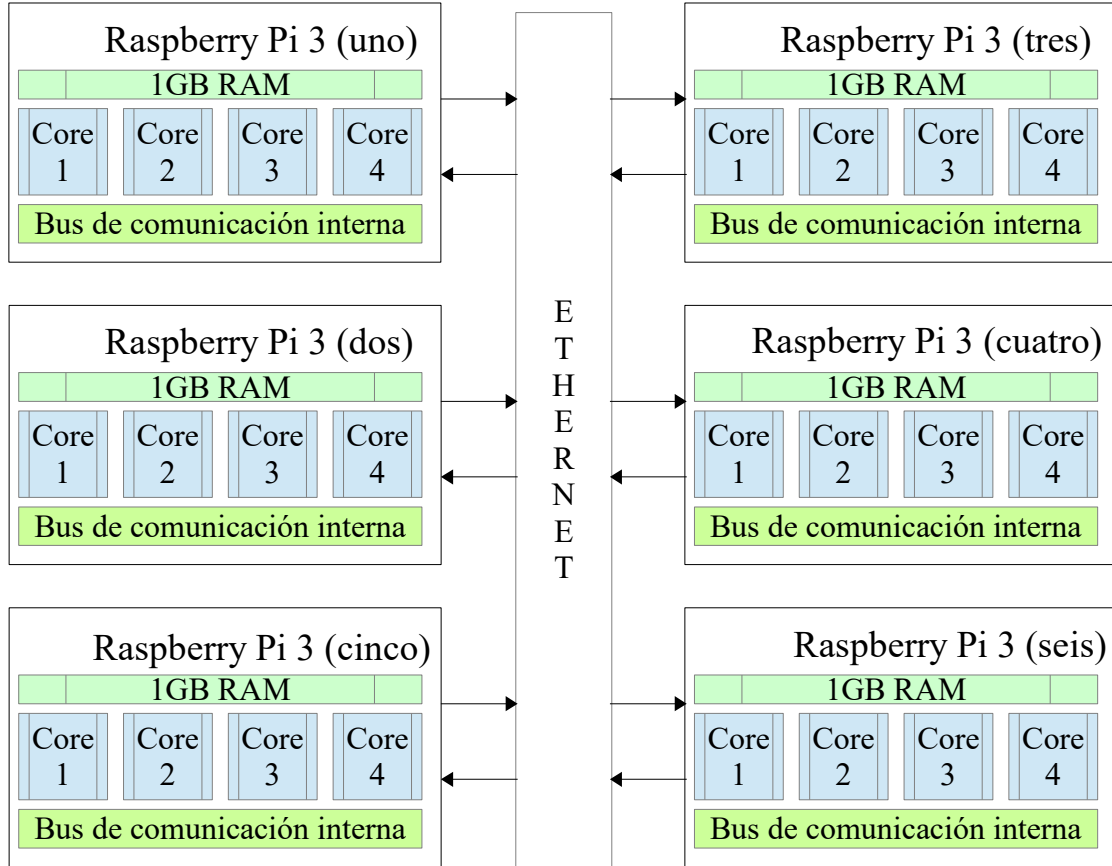


Figura 20: Máquina Clúster FIAD empleando Raspberry Pi 3.

IV.2 Conclusiones

Al implementar el método propuesto en un sistema Raspberry Pi 3, los tiempos de ejecución del programa implementado con procesamiento en paralelo, se ven afectados por la ejecución de aplicaciones y servicios en segundo plano. Por lo que las estimaciones teóricas del factor de rendimiento y de eficiencia del sistema se alejan de los resultados de la ejecución en tiempo real. Adicionalmente, al realizar el encriptado de la imagen

de manera dividida, se realiza un múltiple encriptamiento donde cada proceso de encriptado cuenta con su correspondiente clave de cifrado. Finalmente, se concluye que la ganancia en tiempo de ejecución depende del número de procesadores que intervienen en la ejecución del programa y del recurso en software y hardware asignado a cada procesador, siendo $n = 4$ procesadores el valor óptimo para el caso de un dispositivo Raspberry Pi 3.

Capítulo V

Resultados y validación.

V.1 Resultados de Generador de caos de gran precisión (HPG)

V.1.1 Análisis estadístico con NIST Five-Test 140 – 2

Las series de bits pseudoaleatorias de gran precisión que se obtuvieron a partir del método descrito en la sección II.2.4, se analizaron mediante las pruebas Five-Test, para lo cual se generaron 100 series binarias con una longitud de 100,000 bits cada una como primer caso. En la tabla III se presentan los resultados obtenidos implementando el mapeo caótico de Tinkerbell expresado por el sistema de ecuaciones 1 en el generador de caos, y con una precisión de $np = 99$ decimales significativos. La serie 1, se genera considerando las condiciones iniciales $x_0 = -0.721$ e $y_0 = -0.64$. Las series subsecuentes (2 a 100), se generaron aplicando el método para generar condiciones iniciales con valor pseudoaleatorio descrito en sección II.2.3, considerando un redondeo a $np_{x_0} = 8$. Para un nivel de significancia de $\alpha = 0.05$, los valores límite de las pruebas Five-Test son: $X1 = 3.8415$, $X2 = 5.9915$, $X3 = 14.0671$, $X4 = 9.4877$ y $|X5| = 1.96$, por lo tanto, si los valores obtenidos al realizar las pruebas exceden los valores límite en al menos una prueba, la serie analizada no se considera viable para ser utilizada como serie cifrante en métodos criptográficos (NIST, 2015).

Del análisis de los resultados, al implementar el mapeo de Tinkerbell se obtuvo que: de las 100 series analizadas, 17 de ellas no pasaron al menos una de las pruebas Five-Test obteniendo un 83% de efectividad en las series generadas a partir de los valores de la serie x_{n+1} . En la tabla IV, se presentan los resultados comparativos obtenidos

Tabla III: Resultados de pruebas Five-Test considerando x_{n+1} del mapeo de Tinkerbell.

Mapeo de Tinkerbell	Prueba estadística Five-Test. ($\alpha = 0.05$), $np = 99$					Resultado Five-Test
	X1	X2	X3	X4	X5	
Serie 1	2.401	2.75076152	3.85104851	6.19224308	1.22701281	Viable
Serie 2	0.46656	2.69526162	8.55125551	5.90229427	0.3035908	Viable
Serie 3	0.88804	0.912248	1.97470975	1.15298333	1.20803839	Viable
Serie 4	0.03844	0.04687085	4.20097201	1.28608158	0.0758977	Viable
Serie 5	0.56644	2.28873855	2.90207902	8.13042044	0.27196676	Viable
Serie 6	1.50544	1.70870214	3.1195212	3.33916163	0.49333505	Viable
Serie 7	0.81796	2.57898397	3.56112561	3.07913579	1.02461895	Viable
Serie 8	1.521	3.64628167	7.09204092	4.14049336	2.16308444	No viable
...	...	...	...	...	...	...
Serie 100	4.489	6.58754077	6.78723787	4.41042001	1.10684146	No viable

al aplicar la misma metodología a las series obtenidas a partir de los valores de las series x_{n+1} , y_{n+1} y z_{n+1} correspondientes a los mapeos caóticos de Chen (Chen *et al.*, 2004), Rössler (Itoh *et al.*, 2001), Logístico 2D (Verhulst, 1845) y Hénon (Hénon, 1976) con una precisión de $np = 99$ decimales significativos y un redondeo a $np = 8$ para la generación de nuevas condiciones iniciales. Se observa que el estado x_{n+1} del mapeo de Tinkerbell, presenta un mayor porcentaje de efectividad con un 83%, seguido por el estado z_{n+1} del mapeo de Rössler y el estado y_{n+1} del mapeo de Hénon con 81% de efectividad en ambos casos.

Tabla IV: Tabla comparativa de porcentajes de efectividad Five-Test de NIST considerando los estados de los mapeos caóticos analizados y $np = 99$.

Mapeo caótico	estado con $np = 99$		
	x_{n+1}	y_{n+1}	z_{n+1}
Tinkerbell	82%	76%	No aplica
Chen	70%	77%	No aplica
Rössler	80%	67%	81%
Logístico 2D	65%	64%	No aplica
Hénon	67%	81%	No aplica

Siguiendo la misma metodología, se elige una precisión en el generador de $np = 999$ decimales significativos considerando un redondeo a $np_{x_0} = 8$ para la generación de

nuevas condiciones iniciales. En la tabla V se presentan los resultados de la efectividad que presentan los mapeos caóticos implementados con los diferentes estados caóticos de sus sistemas de ecuaciones; se observa que el estado y_{n+1} del mapeo de Hénon presenta un porcentaje de efectividad del 87% seguido por el estado x_{n+1} del mapeo de Tinkerbell con un 86%.

Tabla V: Tabla comparativa de porcentajes de efectividad Five-Test de NIST considerando los estados de los mapeos caóticos analizados y $np = 999$.

Mapeo caótico	estado con $np = 999$		
	x_{n+1}	y_{n+1}	z_{n+1}
Tinkerbell	86%	81%	No aplica
Chen	83%	78%	No aplica
Rössler	85%	76%	75%
Logístico 2D	72%	72%	No aplica
Hénon	86%	87%	No aplica

En todos los casos se presenta un incremento en la efectividad de las series generadas. Como por ejemplo, para el caso de la serie generada para el estado x_{n+1} del mapeo de Tinkerbell, se incrementa el porcentaje a una efectividad de 86% con $np = 999$ en relación a 82% obtenido con $np = 99$ elementos decimales significativos, estos resultados preliminares nos muestran que mientras mayor sea la precisión numérica para la generación del caos, mayor es el acercamiento del generador caótico a una fuente generadora verdaderamente aleatoria. En la tabla VI, se presentan los resultados obtenidos al aplicar la misma metodología a las series obtenidas con precisión simple, doble precisión, gran precisión con $np = 99$ y gran precisión con $np = 999$ pero con series de menor y mayor longitud y considerando el estado x_{n+1} del mapeo de Tinkerbell. Se observa que al utilizar precisión simple con $np = 9$ y doble precisión con $np = 17$ el porcentaje de efectividad va disminuyendo conforme aumenta la longitud de la serie pseudoaleatoria de bits.

Del análisis estadístico realizado a cada serie de bits generada con los diferentes

Tabla VI: Tabla comparativa de ejemplos de porcentajes de efectividad Five-Test con los diferentes niveles de precisión en series binarias de diferente longitud.

Precisión valor np	% de efectividad Five-Test en x_{n+1} de mapeo de Tinkerbell				
	1, 000 bits	10, 000 bits	50, 000 bits	100, 000 bits	150, 000 bits
$np = 9$	79%	10%	0%	0%	0%
$np = 17$	82%	55%	11%	0%	0%
$np = 99$	78%	81%	81%	82%	76%
$np = 999$	80%	83%	84%	86%	77%

niveles de precisión mediante las pruebas FIPS-NIST (NIST, 2015) se obtuvieron los porcentajes de efectividad que se muestran en la gráfica de la figura 21. Se generaron 100 secuencias binarias con una longitud de 1,000 bits, 100 secuencias binarias con una longitud de 2,000 bits y así sucesivamente, hasta generar 100 secuencias binarias de 150,000 bits de longitud e implementando en cada una de ellas precisión simple, doble precisión, gran precisión con $np = 99$ y gran precisión con $np = 999$; en la generación de condiciones iniciales pseudoaleatorias se consideró un redondeo $np_{x_0} = 5$ para el caso de precisión simple y doble precisión y un redondeo $np_{x_0} = 8$ para el caso de gran precisión. Se observa que los porcentajes de efectividad alcanzados por las series generadas implementando precisión simple y doble precisión son cada vez menores conforme se incrementa el número de elementos de la serie de bits dando un porcentaje de efectividad de 0% en series a partir de 20,000 bits y 77,000 bits respectivamente. Por otra parte, los porcentajes de efectividad alcanzados por las series generadas implementando el método propuesto con gran precisión $np = 99$ y $np = 999$ mantienen un promedio de 77.28% y 78.06% en series de hasta 150,000 bits de longitud y con un máximo de 88% y 86% respectivamente.

V.1.2 Análisis estadístico con NIST 800 – 22 Rev 1a

Con el propósito de evaluar la efectividad que proporciona el generador de secuencia pseudoaleatoria de bits propuesto en comparación con otros generadores de números

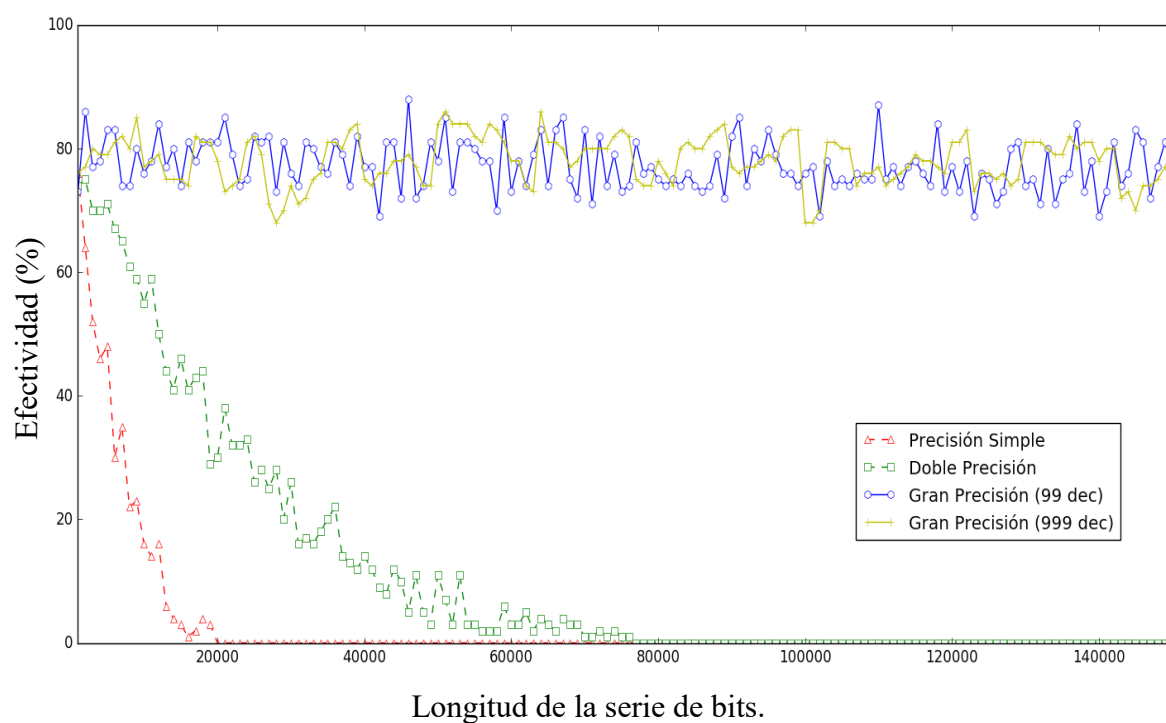


Figura 21: Gráfica de porcentajes de efectividad que proporcionan las series de bits con longitudes desde 1,000 bits y hasta 150,000 bits generadas con los diferentes niveles de precisión numérica.

aleatorios y pseudoaleatorios reportados en la literatura (Rukhin *et al.*, 2010) se realiza un análisis estadístico mediante el conjunto de pruebas NIST 800-22 rev 1a (Rukhin *et al.*, 2010).

La tabla VII, muestra los resultados obtenidos al realizar el conjunto de pruebas al generador propuesto considerando el mapeo de Tinkerbell y un cálculo numérico con gran precisión de $np = 99$ cifras significativas; se observa que alcanza un promedio de porcentajes de éxito de 99.187% en ambos estados caóticos.

Tabla VII: Tabla de porcentajes de éxito del conjunto de pruebas NIST 800–22 realizado al generador propuesto (HPG) considerando el mapeo de Tinkerbell y cálculo numérico con precisión de $np = 99$.

HPG propuesto con mapeo de Tinkerbell con $np = 99$				
NIST 800-22 Rev 1a Prueba	estado x_{n+1}		estado y_{n+1}	
	valor- p	% de éxito	valor- p	% de éxito
Frequency	0.108791	99	0.851383	98
Block Frequency	0.224821	99	0.007694	100
Cumulative sums -Forward	0.304126	99	0.011791	99
Cumulative sums-Reverse	0.383827	99	0.055361	98
Runs	0.102526	97	0.0000051	98
Long Runs of Ones	0.554420	100	0.224821	100
Rank	0.129620	99	0.851383	100
Spectral DFT	0.595549	99	0.924076	99
Non-overlapping Templates	0.971699	100	0.262249	100
Overlapping Templates	0.002971	100	0.699313	100
Universal	0.798139	98	0.779188	98
Approximate Entropy	0.595549	99	0.759756	98
Random Excursions	0.090936	100	0.006196	100
Random Excursions Variant	0.275709	100	0.090936	100
Linear Complexity	0.678686	99	0.514124	99
Serial ($2m \nabla \Psi$)	0.924076	100	0.955835	100

La tabla VIII muestra los resultados comparativos de los porcentajes de éxito obtenidos con los diferentes generadores de números aleatorios y pseudoaleatorios reportados en (Rukhin *et al.*, 2010) donde se observa que el generador HPG propuesto con el

estado x_{n+1} presenta niveles de éxito altamente competitivos con un porcentaje promedio de 99.1875%. A continuación se enlistan los generadores de números aleatorios y pseudoaleatorios comparados en este trabajo de tesis.

Linear Congruential Generator (LCG).

- Quadratic Congruential Generator I (QCG-I).
- Quadratic Congruential Generator II (QCG-II).
- Cubic Congruential Generator II (CCG) .
- Modular Exponentiation Generator (M-EXPG).
- Secure Hash Generator (G-SHA1).
- Micali-Schnorr Generator (MSG).

Tabla VIII: Tabla comparativa de porcentajes de éxito de los diferentes generadores de números aleatorios y pseudoaleatorios y el generador HPG propuesto con el estado x_{n+1} del mapeo de Tinkerbell.

Porcentaje de éxito en el conjunto de pruebas NIST 800-22 Rev 1a(%)								
Prueba	LCG	QC-1	QC-II	CCG	M-EXP	SHA1	MSG	HPG
Frequency	98	95	99	96	96	100	97	99
Block Frequency	100	98	97	99	100	99	99	99
Cum. sums-Forward	97	95	99	98	97	100	97	99
Cum. sums-Reverse	98	95	99	96	97	99	99	99
Runs	100	99	99	64	100	99	100	97
Long Runs of Ones	97	98	98	98	98	99	100	100
Rank	99	99	99	99	98	99	99	99
Spectral DFT	100	98	99	97	100	99	97	99
Non-overlapping T.	100	100	100	100	100	100	100	100
Overlapping T.	99	100	100	99	98	100	96	100
Universal	99	99	98	99	98	99	98	98
Approximate Entropy	99	98	97	98	99	99	99	99
Random Exc.	100	100	100	100	100	100	100	100
Random Exc. Variant	100	100	100	100	100	100	100	100
Linear Complexity	99	98	99	97	99	99	99	99
Serial ($2m\nabla\Psi$)	100	100	100	100	100	100	100	100
Promedio	99.06	98.28	98.93	96.25	98.75	99.43	98.75	99.18

Finalmente la tabla IX muestra los resultados comparativos de realizar el conjunto de pruebas NIST 800-22 Rev 1a a los diferentes estados de los mapeos caóticos analizados

en este trabajo de investigación, en donde se observa que el estado x_{n+1} del mapeo de Tinkerbell presenta los porcentajes de éxito más altos, seguido por el estado y_{n+1} del mapeo de Hénon.

Tabla IX: Tabla comparativa de porcentajes de éxito de los diferentes mapeos caóticos implementados en el generador HPG propuesto con $np = 99$.

Porcentaje de éxito (%) implementando HPG con $np = 99$									
Prueba	Tinkerbell		Chen		Hénon		Rossler		
	x_{n+1}	y_{n+1}	x_{n+1}	y_{n+1}	x_{n+1}	y_{n+1}	x_{n+1}	y_{n+1}	z_{n+1}
Frequency	99	98	98	96	96	100	98	100	96
Block Frequency	99	100	100	99	100	99	99	100	100
Cum. sums-Forward	99	99	98	99	97	100	99	99	96
Cum. sums-Reverse	99	98	99	100	97	100	99	100	96
Runs	97	98	98	99	100	97	97	86	100
Long Runs of Ones	100	100	89	98	98	99	100	100	99
Rank	99	100	96	99	98	98	99	99	99
Spectral DFT	99	99	97	99	100	97	97	99	100
Non-overlapping T.	100	100	99	100	100	100	100	100	100
Overlapping T.	100	100	100	100	100	98	96	96	97
Universal	98	98	99	98	98	100	98	98	98
Approximate Entropy	99	98	99	98	99	99	99	99	99
Random Exc.	100	100	100	100	100	100	100	100	100
Random Exc. Variant	100	100	100	100	100	100	100	100	100
Linear Complexity	99	99	100	100	99	100	100	98	97
Serial ($2m\nabla\Psi$)	100	100	100	99	100	98	100	100	100
Promedio	99.18	99.18	98.28	99.00	98.87	99.06	98.87	98.37	98.56

V.2 Resultados de encriptado y desencriptado de imágenes de alta resolución

V.2.1 Espacio de claves.

La tabla X muestra los espacios de clave en términos de bits proporcionados por el generador de caos de gran precisión propuesto para diferentes mapeos caóticos, en relación a la precisión predefinida de decimales significativos utilizados en el cálculo numérico determinado para: $np = 99$ y $np = 999$. Se observar un incremento exponencial en

el espacio de claves empleando gran precisión con $np = 99$ y $np = 999$, en relación a los estándares que definen la precisión simple y doble precisión. Los espacios de claves obtenidos se ajustan bien a uno de los principios de Kerckhoffs, en donde hace referencia a que la seguridad del sistema criptográfico debe descansar en la seguridad de la clave, debiéndose suponer el resto de los parámetros del sistema criptográfico e incluso su algoritmo deben ser conocidos (Inzunza-González, 2013).

Tabla X: Tabla comparativa de espacios de clave del método propuesto con diferentes niveles de precisión.

Espacio de clave en términos de bit				
Mapeo caótico	Precisión simple	Doble precisión	Gran precisión	
	$np = 6 - 9$	$np = 15 - 17$	$np = 99$	$np = 999$
Tinkerbell	2^{192}	2^{384}	$2^{2,041}$	$2^{19,979}$
Chen	2^{128}	2^{256}	$2^{1,376}$	$2^{13,335}$
Rössler	2^{320}	2^{640}	$2^{3,319}$	$2^{33,268}$
Logístico 2D	2^{192}	2^{384}	$2^{2,041}$	$2^{19,979}$
Hénon	2^{128}	2^{256}	$2^{1,376}$	$2^{13,335}$

V.2.2 Encriptado de imágenes RGB de alta resolución.

Al implementar la metodología propuesta para generar series pseudoaleatorias potencialmente viables para ser utilizadas como series cifrantes, en el método criptográfico propuesto, se presentan los resultados mostrados en las figuras 22-24 implementando el método propuesto con el mapeo de Tinkerbell. Las imágenes utilizadas son, "Mandrill.png" con una resolución de 700×700 pixeles, "Paisaje.png" de 1950×1200 pixeles y "Loto.png" de 2560×1600 pixeles. En las figuras 22b ,23b y 24b que se generaron utilizando precisión simple, se observa que las imágenes resultantes muestran ciertos vestigios de las imágenes originales, mientras que en los criptogramas de las figuras 22c ,23c y 24c que se obtuvieron implementando $np = 99$ decimales significativos se observan imágenes completamente ininteligibles.

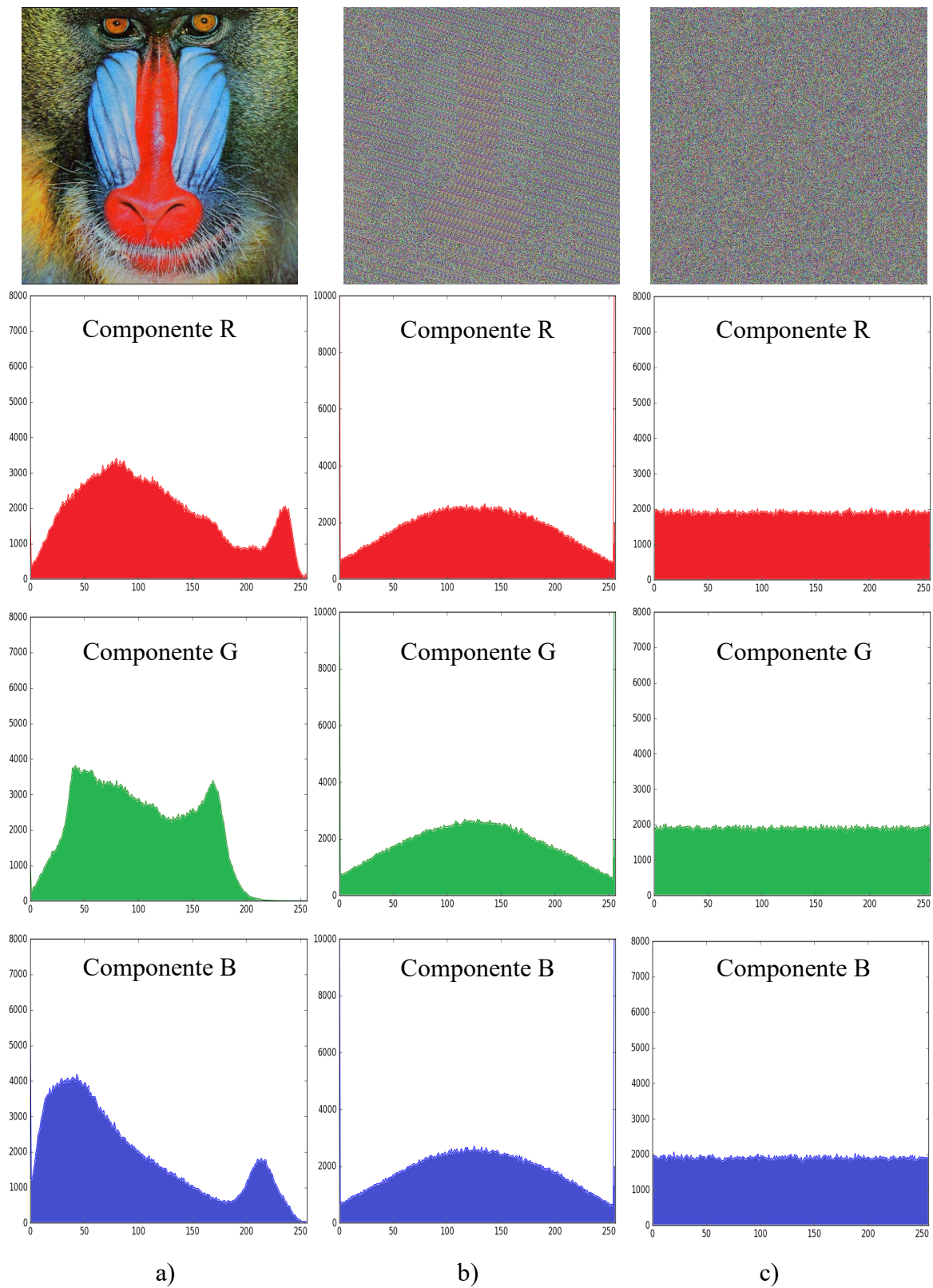


Figura 22: a) Imagen "Mandrill.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbell con $np = 99$.

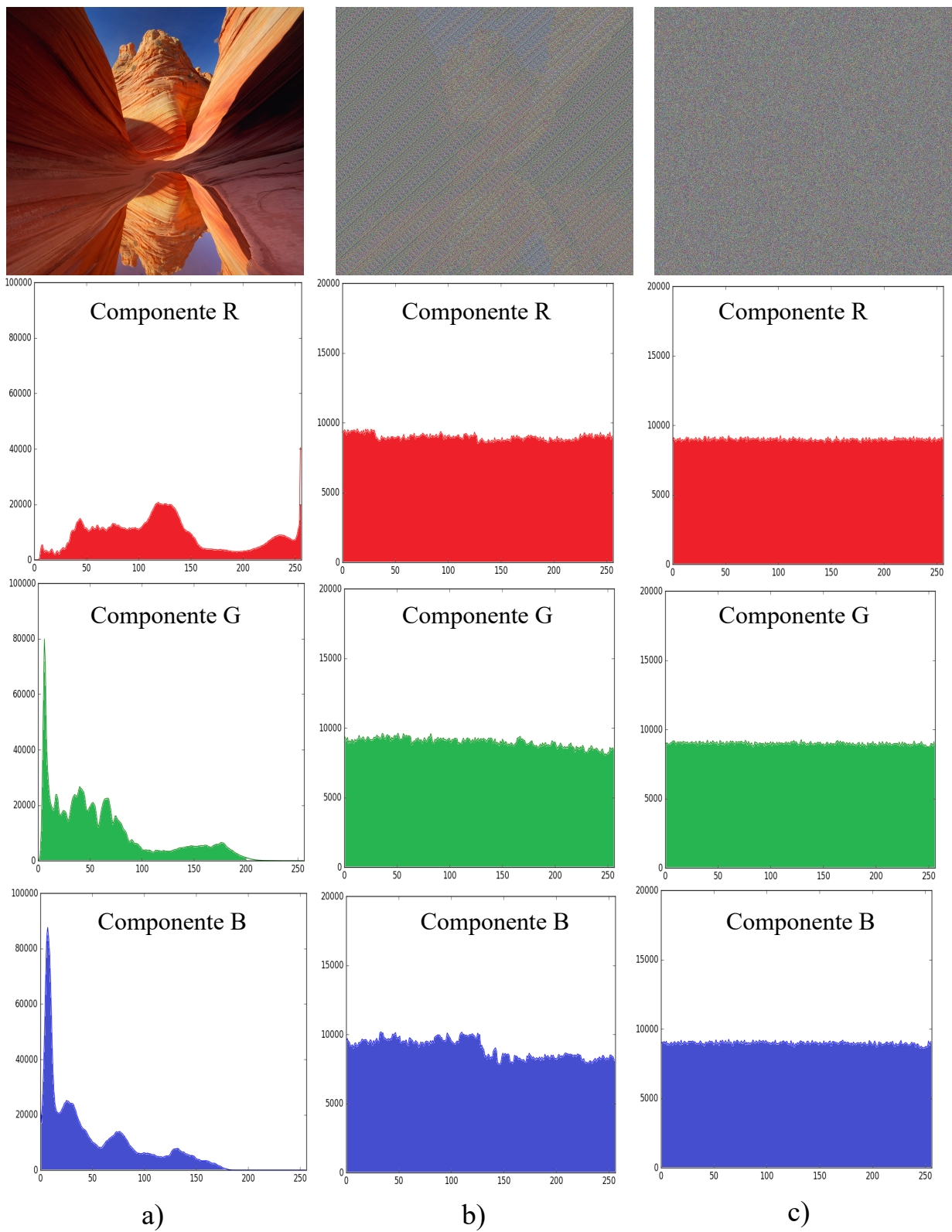


Figura 23: a) Imagen "Paisaje.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbell con $np = 99$.

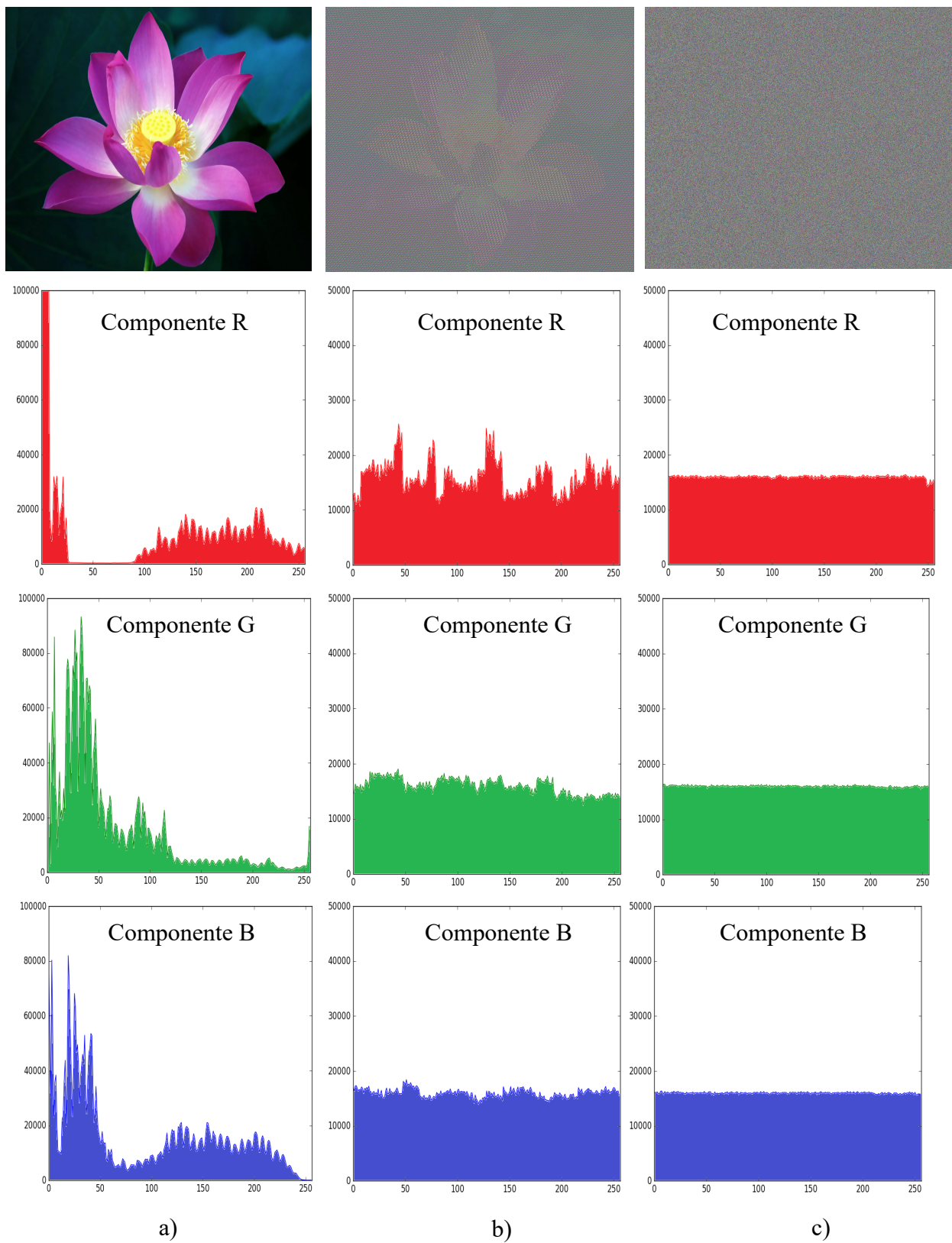


Figura 24: a) Imagen "Loto.png", b) Criptograma obtenido implementando precisión simple, c) Criptograma obtenido implementando HPG mapeo de Tinkerbell con $np = 99$.

V.2.3 Prueba XOR en criptogramas con clave semejante.

Al utilizar el método propuesto y en específico, la implementación de la precisión arbitraria, la sensibilidad de la clave de cifrado depende de la precisión numérica que se determine utilizar. Esto quiere decir, que si se establece una precisión de $np = 99$ decimales significativos, el cambio mínimo aceptable en al menos uno de los valores numéricos en los parámetros de control o en las condiciones iniciales que componen la clave es de $\Delta = 1 \times 10^{-99}$. La figura 25 muestra los criptogramas C_1 que corresponde a la imagen "Lena.png" de 256×256 pixeles y C_2 que corresponde a la imagen "Loto.png" ajustada a un tamaño de 256×256 pixeles, obtenidos aplicando el método propuesto con el mapeo de Tinkerbell y en donde las claves de encriptado CL_1 y CL_2 , que producen S_1 y S_2 respectivamente; tienen un valor numérico con una diferencia de $\Delta x = 1 \times 10^{-99}$ en la condición inicial x_0 .

$M_1 = \text{Lena.png}$, 256×256 pixeles.

$M_2 = \text{Loto.png}$, 256×256 pixeles.

$CL_1 = \{a, b, c, d, x_0, y_0, np_{x0}, ret, np\}$ que produce S_1 ,

$CL_2 = \{a, b, c, d, x_0 - \Delta x, y_0, np_{x0}, ret, np\}$ que produce S_2 ,

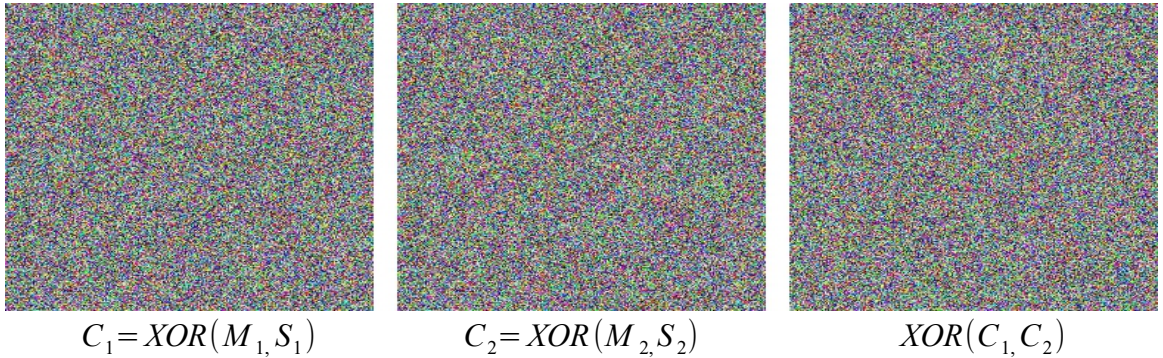


Figura 25: Prueba XOR en dos criptogramas diferentes obtenidos con clave simétrica semejante.

V.2.4 Descriptado de imagen RGB

La figura 26 muestra los resultados del proceso de descriptado utilizando la metodología propuesta, se puede observar que la figura 26a se obtiene de descriptar el criptograma de la figura 22c, la figura 26b se obtiene de descriptar el criptograma de la figura 23c y la figura 26c se obtiene de descriptar el criptograma de la figura 24c, siendo estas imágenes idénticas a las respectivas imágenes originales, es decir, se recupera el 100% de la información de la imagen original.

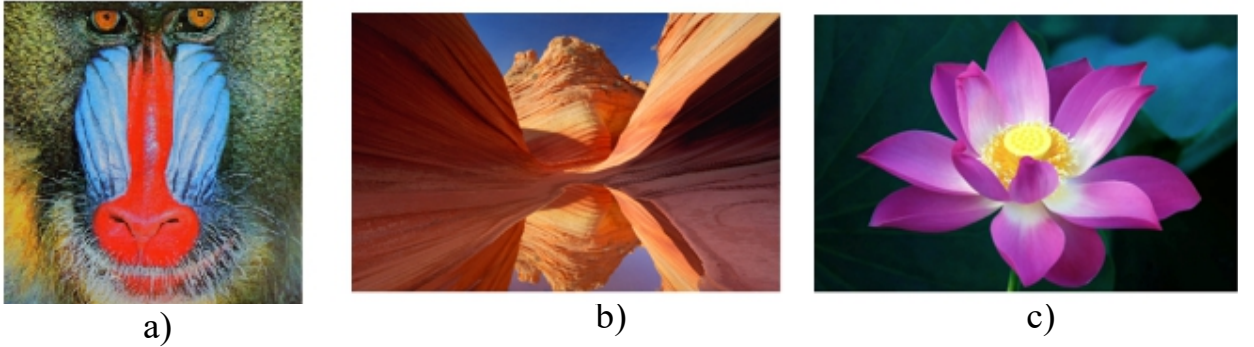


Figura 26: Imágenes recuperadas considerando la metodología propuesta; a) Recuperación del criptograma de "Mandrill.png", b) Recuperación del criptograma de "Paisaje.png" y c) Recuperación del criptograma de "Loto.png".

V.2.5 Pruebas contra ataques diferenciales NPCR y UACI

Considerando los criptogramas C_1 y C_2 que se obtuvieron de la imagen "Paisaje.png", ajustada a un tamaño de 1024×1024 y cuyo formato de imagen a color es RGB de 8 bits. Según (Wu *et al.*, 2011), los valores críticos para la prueba NPCR en el encriptamiento de imágenes con los niveles de significancia N_α son: $N_{0.05} = 99.5994\%$, $N_{0.01} = 99.5952\%$ y $N_{0.001} = 99.5906\%$. La tabla XI muestra los resultados de implementar la prueba NPCR a los criptogramas obtenidos mediante la metodología propuesta e implementando los mapeos caóticos utilizados en el presente trabajo de investigación. Si los valores obtenidos son menores a N_α , entonces, se considera que C_1

y C_2 fallan la prueba. En la tabla XII, se muestran los resultados de la prueba UACI con niveles de significancia U_α y considerando los valores críticos correspondientes (Wu *et al.*, 2011). Si los valores obtenidos están fuera del intervalo (U_α^-, U_α^+) , entonces, se considera que C_1 y C_2 fallan la prueba.

Tabla XI: Tabla de resultados de la prueba NPCR a criptograma resultante implementando HPG mapeo de Tinkerbell x_{n+1} con $np = 99$ y la imagen "Paisaje.png".

Tamaño de imagen criptograma 1024×1024		Valores críticos NPCR		
Mapeo caótico	Valor obtenido	$N_{0.05} = 99.5994$	$N_{0.01} = 99.5952$	$N_{0.001} = 99.5906$
Tinkerbell	99.6086%	Pasó	Pasó	Pasó
Chen	99.3496%	Falló	Falló	Falló
Rössler	99.6070%	Pasó	Pasó	Pasó
Logístico 2D	99.6071%	Pasó	Pasó	Pasó
Hénon	99.2842%	Falló	Falló	Falló

Tabla XII: Tabla de resultados de la prueba UACI a criptograma resultante implementando HPG con mapeo de Tinkerbell x_{n+1} con $np = 99$ y la imagen Paisaje.png.

Tamaño de imagen criptograma 1024×1024		Valores críticos UACI		
Mapeo caótico	Valor obtenido	$U_{0.05}^- = 33.4183\%$	$U_{0.01}^- = 33.4040\%$	$U_{0.001}^- = 33.3875\%$
		$U_{0.05}^+ = 33.5088\%$	$U_{0.01}^+ = 33.5231\%$	$U_{0.001}^+ = 33.5396\%$
Tinkerbell	33.4397%	Pasó	Pasó	Pasó
Chen	33.4271%	Pasó	Pasó	Pasó
Rössler	33.4526%	Pasó	Pasó	Pasó
Logístico 2D	33.4367%	Pasó	Pasó	Pasó
Hénon	33.4518%	Pasó	Pasó	Pasó

V.2.6 Pruebas de criptogramas con entropía de información

Considerando el criptograma C_1 , obtenido de la imagen Paisaje.png, ajustada a un tamaño de 1024×1024 pixeles y cuyo formato de imagen a color es RGB de 8 bits. Según (Inzunza-González, 2013), la entropía de la información permite medir la aleatoriedad que presentan los criptogramas como resultado de modificar la imagen original mediante un método criptográfico. Debido a que en la presente investigación se utilizaron sistemas determinísticos; los valores obtenidos mostrados en las tablas XIII y

XIV, se aproximan al valor de entropía de total aleatoriedad. La información que corresponde a los elementos de $C = \{c_1, \dots, c_i\}$, son pixeles a color de 8 bits. Por lo tanto, el valor numérico de entropía de la información de la total aleatoriedad de los colores de la imagen del criptograma, para este caso es de 8 bits/símbolo. La tabla XIII muestra los resultados de entropía obtenidos aplicando el método criptográfico con precisión simple, doble precisión y gran precisión a la imagen "Paisaje.png" de 1024×1024 pixeles e implementando el mapeo de Tinkerbell; se observa que al implementar gran precisión, se obtiene mejor pseudoaleatoriedad en el criptograma. La tabla XIV, muestra los resultados de entropía obtenidos al aplicar gran precisión en el método criptográfico con los mapeos caóticos implementados en el presente trabajo de investigación, con el propósito de comparar la pseudoaleatoriedad producida en el criptograma resultante.

Tabla XIII: Valores de entropía de la información del criptograma resultante usando precisión simple, doble precisión y gran precisión con $np = 99$ en el método criptográfico propuesto considerando la imagen "Paisaje.png".

Nivel de precisión	Entropía por componente (bit/símbolo)		
	R	G	B
Precisión simple	7.9891	7.9913	7.9963
Doble precisión	7.99978	7.99975	7.9998307
Gran precisión	7.99979	7.99979	7.9998352

Tabla XIV: Valores de entropía de la información del criptograma resultante considerando $np = 99$ y distintos mapeos caóticos en el generador HPG propuesto con la imagen "Paisaje.png".

Criptograma (1024×1024 pixeles)		Entropía por componente (bit/símbolo)		
Mapeo caótico	entropia promedio	R	G	B
Tinkerbell	7.99980	7.99979	7.99979	7.99983
Chen	7.99936	7.99902	7.99936	7.99970
Rössler	7.99980	7.99978	7.99978	7.99982
Logístico 2D	7.99980	7.99976	7.99983	7.99981
Hénon	7.99929	7.99884	7.99933	7.99971

V.3 Resultados del sistema criptográfico de imágenes a color con procesamiento en paralelo

Con propósitos experimentales, se consideró la imagen a color de 32 bits "Lena.png" de 320×320 pixeles, lo anterior para garantizar que la imagen tenga una distribución equitativa entre los procesadores que intervienen en la ejecución del programa. El programa se ejecutó con 2, 4, 8, 10, 16 20, 32, y 40 procesadores cada vez.

V.3.1 Resultados de implementación en máquina Raspberry Pi 3.

La tabla XV muestra los resultados obtenidos de los tiempos de ejecución de la parte secuencial α del programa, así como de los tiempos de ejecución de las partes con procesamiento paralelo β_p y de comunicación al ser implementados en un solo dispositivo Raspberry Pi 3 como máquina de procesamiento. La figura 27, muestra la gráfica de los resultados obtenidos de los tiempos de ejecución del programa.

Tabla XV: Tabla de resultados de los tiempos de ejecución del método criptográfico propuesto en máquina Raspberry Pi 3.

Número de Procesadores n	Tiempos de ejecución en segundos del método propuesto.			
	Total	secuencial α	paralelo β_p	comunicación
1	708.5599	708.5599	No aplica	No aplica
2	364.228	0.8624	363.355	0.0107
4	194.0539	0.8412	193.193	0.0192
8	197.3918	0.9151	188.1383	8.3384
10	198.8052	0.9837	187.5458	10.2757
16	198.8154	1.1086	187.1824	10.5244
20	198.7596	1.1349	188.5569	9.0678
32	201.0563	1.4636	199.1393	0.4535
40	201.3307	1.6168	189.2413	10.4724

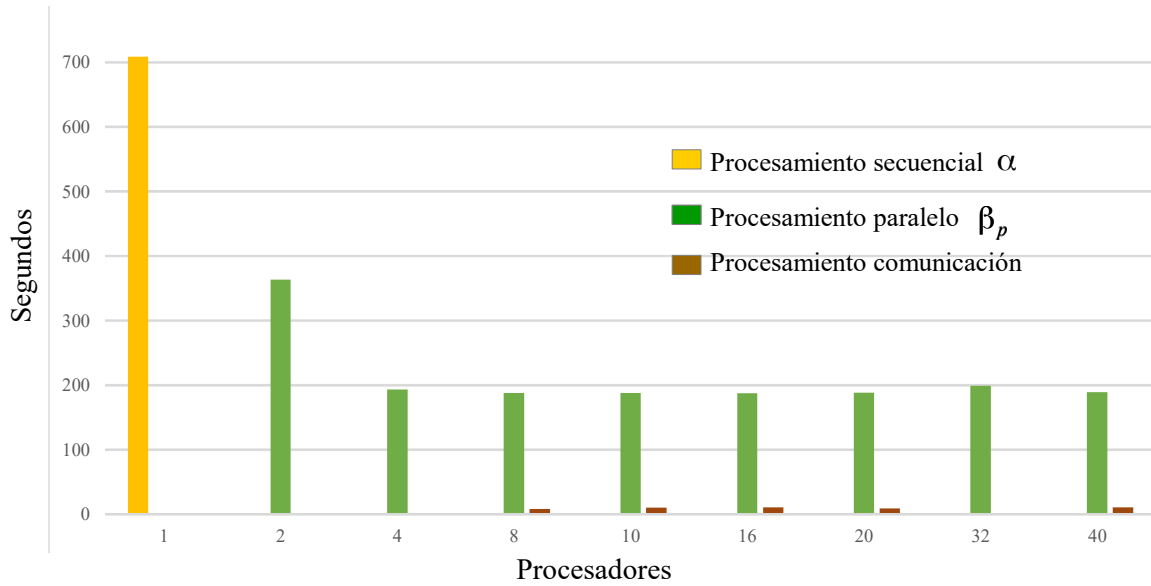


Figura 27: Gráfica de rendimiento del método criptográfico propuesto con procesamiento en paralelo en máquina Raspberry Pi 3.

V.3.2 Resultados de implementación en máquina Clúster FIAD

La tabla XVI muestra los resultados obtenidos de los tiempos de ejecución de la parte secuencial α del programa, así como de los tiempos de ejecución de las partes con procesamiento paralelo β_p y de comunicación. La figura 28, muestra la gráfica de los resultados obtenidos de los tiempos de ejecución del programa, se observa que a medida que se incrementa el número de procesadores disminuye el tiempo de procesamiento paralelo α y se incrementa el tiempo de distribución de datos y recolección de información entre procesos.

En la figura 29 se muestra el resultado del proceso de encriptado de la imagen de entrada "Lena.png", se observa que la imagen recuperada 29c es idéntica a la imagen original 29a y que el criptograma 29b es totalmente ininteligible.

De la misma manera en la figura 30 se muestra el resultado del proceso de desencriptado con un error en el parámetro correspondiente al número de procesadores que

Tabla XVI: Tabla de resultados de los tiempos de ejecución del método criptográfico propuesto en máquina Clúster FIAD.

Número de Procesadores n	Tiempos de ejecución en segundos del método propuesto.			
	Total	Secuencial α	Paralelo β_p	Comunicación
1	87.6573	87.6563	No aplica	No aplica
2	46.7736	0.5897	45.8394	0.68
4	44.2705	1.4891	41.79	1.9574
8	24.7102	1.1856	22.0905	2.2684
10	19.2366	0.9068	17.6417	0.9901
16	12.8329	1.1119	10.9127	1.5612
20	11.7413	1.2129	8.798	2.6352
32	11.6232	1.5254	8.1963	1.0906
40	11.5619	1.7773	7.8896	3.2476
64	12.7167	2.0571	7.3751	4.2
82	13.0677	3.0619	7.8106	3.1401

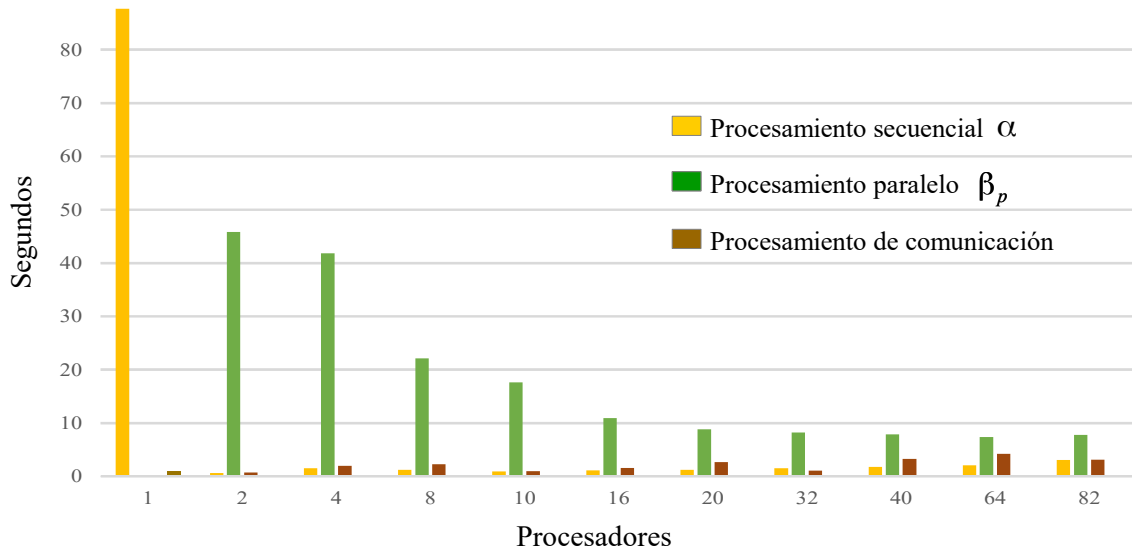


Figura 28: Gráfica de rendimiento del método criptográfico propuesto con procesamiento en paralelo.

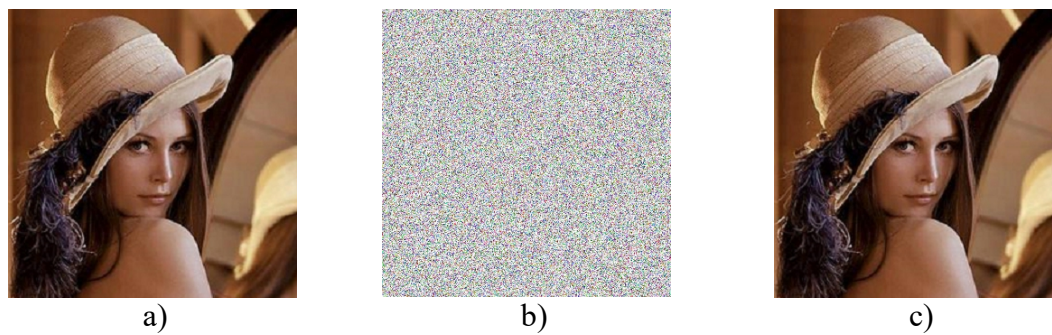


Figura 29: Resultado de encriptado con procesamiento en paralelo: a) Imagen de entrada, b) Criptograma y c) Imagen recuperada.

intervienen en el proceso de encriptado.

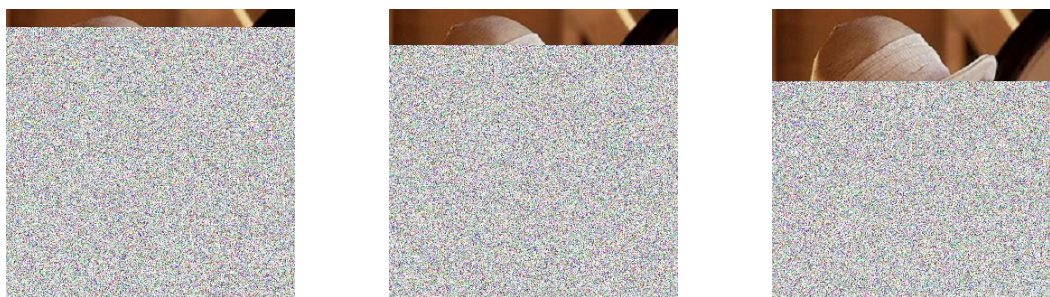


Figura 30: Ejemplos de recuperación de la imagen en el proceso de desencriptado con un error en el parámetro correspondiente al número de procesadores.

Capítulo VI

Conclusiones generales.

En este trabajo de tesis doctoral, se presentó un método para incrementar la precisión numérica en un generador de caos digital, basado en la implementación utilizando aritmética de precisión arbitraria en sistemas embebidos SoC tales como Raspberry Pi 3. Con la implementación del incremento en la precisión numérica bajo este contexto, se disminuye significativamente el efecto de degradación digital del caos. En la metodología propuesta, es práctico realizar la experimentación con diferentes modelos caóticos reportados en la literatura, debido a que el modelo caótico se implementa por software, y en caso de ser necesario de cambiar de sistema caótico, basta solamente con cambiar el sistema de ecuaciones y sus respectivos parámetros de control en el código correspondiente. Adicionalmente, con el método propuesto para generar condiciones iniciales pseudoaleatorias, se garantiza que los valores utilizados como condiciones iniciales para un mapeo caótico, son válidos y aseguran la generación de trayectorias caóticas. Se observó que la viabilidad de la serie generada depende en gran medida del establecimiento de las condiciones iniciales y de la elección de los parámetros de control. De los resultados obtenidos en el análisis estadístico mediante las pruebas FIPS PUB 140-2 y el conjunto de pruebas NIST SP 800-22 Rev. 1a, a las secuencias cifrantes obtenidas con el método propuesto, se demostró que los mapeos caóticos implementados en el generador de caos propuesto, pueden generar secuencias pseudoaleatorias de bits con una mayor aproximación a una fuente generadora de bits verdaderamente aleatoria. Por lo tanto, se demostró que el generador de caos propuesto es viable y altamente competitivo comparado con otros generadores de números aleatorios y pseudoaleatorios reportados

en la literatura. Por otra parte, las series generadas presentan un alto nivel de aleatoriedad proporcionando una mayor seguridad durante un mayor número de iteraciones del mapeo caótico implementado, en comparación con las secuencias generadas al utilizar el estandar IEEE 754 para aritmética de punto flotante con precisión simple o doble precisión. Con respecto al método propuesto para el encriptado y desencriptado de imágenes a color de muy alta resolución, se presenta un esquema de fácil integración en diversos sistemas SoC gestionados por un sistema operativo de uso común y de código abierto, basado en GNU/Linux. Debido a ello, es posible la implementación de técnicas de procesamiento en paralelo, con el propósito de optimizar el tiempo de encriptado y desencriptado de la información confidencial, proporcionando un alto nivel de seguridad criptográfica debido al incremento del espacio de clave simétrica. Este espacio de claves obtenido con el sistema criptográfico propuesto, se incrementó exponencialmente (del orden de $2^{33,268}$ o superior) proporcionando una mayor seguridad criptográfica con base en el concepto de mantener un método simple de cifrado con gran espacio de claves. Del análisis de seguridad y entropía de información, se confirma que el sistema criptográfico propuesto presenta altos niveles de seguridad y es robusto contra ataques diferenciales y estadísticos. Por lo anterior, se cumple con uno de los preceptos mas importantes de la seguridad criptográfica que establece, que la seguridad en un método criptográfico depende del espacio de claves que proporciona y no de la complejidad del método criptográfico o del algoritmo utilizado.

El método propuesto es potencialmente factible de ser implementado en sistemas embebidos de alto rendimiento computacional y de última generación que tiene hardware de múltiples procesadores.

VI.1 Trabajos a futuro

El SoC Raspberry Pi 3, es gestionado por una adaptación de un sistema operativo de uso común basado en GNU/Linux, debido a ello, existen diferentes adaptaciones de diversos sistemas operativos de código abierto, diseñados para implementarse en el dispositivo SoC. Por lo anterior, se propone realizar pruebas experimentales en diferentes sistemas operativos adaptados, con el propósito de determinar una configuración en software óptima para la ejecución del sistema criptográfico propuesto.

Por otra parte, con la viabilidad del método propuesto con relación a la capacidad de encriptar grandes cantidades de información, se propone implementar el encriptado de grandes cantidades de información, por ejemplo, secuencias de imágenes, video en streaming y audio. Adicionalmente se recomienda realizar como trabajo futuro:

- Desarrollar un nuevo algoritmo de encriptado caótico empleando técnicas de cómputo paralelo con otros dispositivos embebidos de alto rendimiento y múltiples núcleos.
- Implementar el algoritmo propuesto en otras plataformas de dispositivos embebidos de multiprocesamiento avanzado empleando GPU's (unidades de procesamiento gráfico).
- Desarrollar una aplicación para un teléfono celular inteligente de última generación que tenga múltiples núcleos de procesamiento y utilice el método propuesto para el encriptado de información confidencial y la generación dinámica de claves de cifrado.

Bibliografía

- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., y Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communication Surveys and Tutorials.*, **17**(4): 2347–2376.
- Azzaz, M. S., Camel Tanougast, S. S., y Bouridane., A. (2013a). Synchronized hybrid chaotic generators: Application to real-time wireless speech encryption. *Communications in Nonlinear Science and Numerical Simulation*, **18**(8): 2035–2047.
- Azzaz, M. S., Tanougast, C., Sadoudi, S., Fellah, R., y Dandache, A. (2013b). A new auto-switched chaotic system and its FPGA implementation. *Communications in Nonlinear Science and Numerical Simulation*, **18**(7): 1792–1804.
- Cerqueira Ferreira, Hiro Gabriel; De Sousa Junior, R. T. (2017). Security analysis of a proposed internet of things middleware. *Cluster Comput*, **20**: 651–660.
- Chen, G., Mao, Y., y Chui, C. K. (2004). A symmetric image encryption based on 3D chaotic cat maps. *Chaos, Solitons and Fractals*, **21**(3): 749–761.
- Cruz-Hernández, C., Inzunza-González, E., López-Gutiérrez, R. M., Serrano-Guerrero, H., y García-Guerrero, E. E. (2010). Encrypted audio communication based on synchronized unified chaotic systems. *World Academy of Science, Engineering and Technology*, **66**: 869–874.
- Dalcin, L., Paz, R., y Storti, M. (2005). MPI for Python. *Journal of Parallel and Distributed Computing*, **65**(9): 1108–1115.
- Dalcin, L., Kler, P., Paz, R., y Cosimo, A. (2011). Parallel distributed computing using Python. *Advances in Water Resources*, **34**(9): 1124–1139.
- Diffie, W. y Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions of Information Theory*, **IT-22**: 644–654.

- Fuster, A., Martinez, D., Hernández, L., Montoya, F., y Muñoz, J. (2001). *Técnicas Criptográficas de Protección de Datos*. Alfaomega Ra-Ma.
- Gene, A. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. *FIPS Conference Proceedings*, (30): 483–485.
- Goldszsztejn, A., Hayes, W., y Collins, P. (2011). Tinkerbell is chaotic. *Siam Journal on Applied Dynamical Systems*, **10** (4): 1480–1501.
- Hanping, H., Yashuang, D., y Lingfeng, L. (2014). Counteracting the dynamical degradation of digital chaos via hybrid control. *Communications in Nonlinear Science and Numerical Simulation*, **19**(6): 1970–1984.
- Heidari-Bateni, G. y McGillem, C. D. (1994). A chaotic direct-sequence spread-spectrum communication system. *IEEE Transactions on Communications*, **42**(2/3/4): 1524–1527.
- Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, **50**(1): 69–77.
- Hwang, K. y Jotwani, N. (2011). *Advanced Computer Architecture: Parallelism, Scalability, Programmability..* Mc Graw Hill, tercera edición.
- IEEE, C. S. (2008). Standard for floating - point arithmetic. *IEEE Std 754 - 2008*, páginas 1–58.
- IEEE Computer, S. (2008). IEEE standard VHDL language reference manual. *IEEE Std 1076-2008*, páginas 1–640.
- Inzunza-González, E.; Cruz-Hernández, C. (2013). Double hyperchaotic encryption for security in biometric systems. *Nonlinear Dynamics and Systems Theory*, **13**(1): 55–68.
- Itoh, M., Yang, T., y Chua (2001). L.O. conditions for impulsive synchronization of chaotic and hyperchaotic systems. *International Journal of Bifurcation and Chaos*, **11**(2): 551–560.

- Kocamaz, U. E., Cicek, S., y Uyaroglu, Y. (2018). Secure communication with chaos and electronic circuit design using passivity-based synchronization. *Journal of Circuits, Systems and Computers*, **27**(04): 1850057[20pages].
- Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., y Christensen., R. (2017). The atomic simulation environment - a Python library for working with atoms. *Journal of Physics: Condensed Matter*, **29**: 273002[30 pages].
- Li, S., Chen, G., y Mou, X. (2005). On the dynamical degradation of digital piecewise linear chaotic maps. *International Journal of Bifurcation and Chaos*, **15**(10): 3119–3151.
- Mazloom, A. y Eftekhari-Moghadam, A. M. (2009). Color image encryption based on coupled nonlinear chaotic map. *Chaos, Solitons and Fractals*, **42**(2009): 1745–1754.
- Menezes, A. J., Oorschot, P. C. V., y Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC Press.
- Moon, F. C. (1992). *Chaotic And Fractal Dynamics. An Introduction for applied Scientists and Engineers*. WILEY-VCH Verlag Gmbh and Co. KGaA.
- Morabito, R., Petrolo, R., Loscrí, V., y Nathalie, M. (2018). LEGIoT: A lightweight edge gateway for the internet of things. *Future Generation Computer Systems*, **81**: 1–15.
- Murillo-Escobar, M., Cruz-Hernández, C., Abundiz-Pérez, F., López-Gutiérrez, R., y Acosta del Campo, O. (2015). A RGB image encryption algorithm based on total plain image. *Signal Processing*, **109**: 119–131.
- Murillo-Escobar, M. A., Cruz-Hernández, C., Abundiz-Pérez, F., y López-Gutiérrez, R. (2016). Implementation of an improved chaotic encryption algorithm for real-time embedded systems by using a 32-bit microcontroller. *Microprocessors and Microsystems*, **45**: 297–309.

- Ng, Irene C.L.; Wakenshaw, S. Y. (2017). The internet-of-things: Review and research directions. *International Journal of Research in Marketing*, **34**: 3–21.
- NIST (2015). Security requirements for cryptographic modules. *Federal Information Processing Standard (FIPS PUB 140-2)*, páginas 1–56.
- Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., y Vo, S. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications. *National Institute of Standards and Technology Special Publication 800-22 Rev.1a*, páginas 1–131.
- Sadoudi, S., Tanougast, C., Azzaz, M. S., y Dandache., A. (2013). Design and FPGA implementation of a wireless hyperchaotic communication system for real-time image transmission. *EURASIP Journal on Image and Video Processing*, **43**: 1–18.
- Shaoliang, Y. y Jiang, T. (2011). Bifurcation and chaos in the Tinkerbell map. *International Journal of Bifurcation and Chaos*, **11**: 3137–3156.
- Shujun, L., Xuanqin, M., Yuanlong, C., Zhen, J., y Jihong, Z. (2003). On the security of a chaotic encryption scheme: Problems with computerized chaos in finite computing precision. *Computer Physics Communications*, **153**: 52–58.
- Smith, D. M. (2003). Using multiple-precision arithmetic. *Computing in Science & Engineering*, **5**(4): 88–93.
- Tlelo-Cuautle, E., Carbajal-Gomez, V. H., Obeso-Rodelo, P. J., Rangel-Magdaleno, J. J., y Núñez Pérez, J. C. (2015a). FPGA realization of a chaotic communication system applied to image processing. *Nonlinear Dynamics*, **82**(4): 1879–1892.
- Tlelo-Cuautle, E., Rangel-Magdaleno, J. J., Pano-Azucena, A., Obeso-Rodelo, P. J., y Núñez Pérez, J. C. (2015b). FPGA realization of multi-scroll chaotic oscillators. *Communications in Nonlinear Science and Numerical Simulation*, **27**(1-3): 66–80.

- Verhulst, P. F. (1845). Recherches mathematiques sur la loi d' accroissement de la population. *Nouv. mém. de l' Academie Royale des Sci. et Belles-Lettres de Bruxelles*, **18**: 1–41.
- Wei, H., Hong, G., Hong, G., Kai, Z., Jun, L., y Liu, X. (2017). A novel design of software system on chip for embedded system. *Journal of Signal Proccesing Systems*, **86**: 135–147.
- Wu, Y., Noonan, J. P., y Agaian, S. (2011). NPCR and UACI randomness tests for image encryption. *Multidisciplinary Journals in Science and Technology*, páginas 31–38.
- Yashuang, D., Hanping, H., Naixue, X., Wei, X., y Lingfeng, L. (2015). A general hybrid model for chaos robust synchronization and degradation reduction. *Information Sciences*, páginas 146–164.
- Yu-Ming, X., Bo-Cheng, B., y Qiang, X. (2010). Grid-scroll hyperchaotic system based on microcontroller digital hardware implementation. *Acta Physica Sinica*, **59**(9): 5959–5966.