Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería

Maestría y Doctorado en Ciencias e Ingeniería

Algoritmo Criptográfico para Sistemas Embebidos de Bajo Poder Computacional

TESIS

que para obtener el grado de

Maestro en Ciencias

Presenta:

Evangelina Lara Camacho

Director de Tesis

Dr. Leocundo Aguilar Noriega

Tijuana B.C. **Diciembre 2015**

Universidad Autónoma de Baja California

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

COORDINACIÓN DE POSGRADO E INVESTIGACIÓN

FOLIO No. 161

Tijuana, B. C., a 20 de noviembre de 2015

C. Evangelina Lara Camacho
Pasante de: Maestro en Ciencias
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS

Es propuesto, por el C. Dr. Leocundo Aguilar Noriega

quien será el responsable de la calidad del trabajo que usted presente, referido al tema: Algoritmo Criptográfico para Sistemas Embebidos de Bajo Poder Computacional.

el cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I.- INTRODUCCIÓN
II.- ANTECEDENTES Y MARCO TEORICO
III.- ALGORITMO SALSA20 MODIFICADO
IV.- ACL: ALGORITMO PROPUESTO
V.- PRUEBAS RESULTADOS
VI.- GENERADOR DE NUMEROS VERDADERAMENTE ALEATORIOS
VII.- CONCLUSIONES Y TRABAJO FUTURO

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Dr. José Luis González Vázquez
Sub-Director Secretario

Dr. Leocundo Aguilar Noriega
Director de Tesis

FACULTAD DE CIENCIAS QUIMICAS E INGENIERIA

Dr. Luis Enrique Palafox Maestre
Director

Resumen

Debido a que los sistemas embebidos están limitados en recursos, las soluciones de seguridad aplicadas en redes tradicionales pueden no ser adecuadas para ellos. En este trabajo de tesis se presentan dos propuestas de algoritmos de cifrado que pueden ser una alternativa para proveer seguridad a este tipo de sistemas. La primera propuesta es una modificación del algoritmo de cifrado de flujo Salsa20, el cual es uno de los finalistas del proyecto eStream. La segunda es un algoritmo de cifrado de flujo completamente nuevo.

En este documento se presenta para ambos algoritmos su diseño, evaluación y requerimientos de cómputo.

Abstract

Due to the fact that embedded systems have limited resources, the security solutions applied to traditional networks might not be adequate for them. In this thesis we present two cipher algorithms that could be an alternative to provide security for this kind of systems. The first proposal is a modification of the Salsa20 algorithm, which was one of the finalists of the eStream project. The second is a completely new cipher algorithm.

This works describes the design, evaluation and computational requirements of the two algorithms mentioned above.

Agradecimientos

Aprovecho este apartado para presentar tres dedicatorias y agradecimientos de esta tesis.

Antes que todo, quiero dedicar este trabajo a mi tutor el Dr. Leocundo Aguilar, y agradecerle sus enseñanzas, consejos y correcciones. También quiero mencionar que muchos de los logros profesionales de sus tutorados y estudiantes han sido gracias a su dedicación a la enseñanza.

Dedico con mucho cariño este trabajo a mi familia, y agradecerles todo su apoyo, educación, enseñanzas y el ánimo que me han brindado para continuar adelante.

Por último, quiero hacer con mucho cariño una dedicatoria a mi novio, y agradecerle todo su apoyo y motivación brindada en este y demás proyectos.

A todos ellos, muchas gracias.

Índice General

Resumen iii

Abstract iv

Agradecimientos v

Lista de Tablas viii

Lista de Figuras x

1. Introducción 1
   - 1.1 Planteamiento del problema 1
   - 1.2 Hipótesis 2
   - 1.3 Propuesta 2
   - 1.4 Estructura de la Tesis 3

2. Antecedentes y Marco Teórico 5
   - 2.1 Sistema Embebido 5
   - 2.2 Cómputo Ubicuo 5
   - 2.3 Criptografía 6
   - 2.4 Aleatoriedad 10
   - 2.5 Números Aleatorios 10
   - 2.6 Números Pseudo Aleatorios 11
   - 2.7 Generador de Números Verdaderamente Aleatorios 11
   - 2.8 Generador de Números Pseudo Aleatorios 12
   - 2.9 Generador de Números Pseudo Aleatorios Criptográficamente seguro 12
   - 2.10 Operación XOR 13
   - 2.11 Proyecto eSTREAM 15
   - 2.12 Algoritmo Salsa20 16
   - 2.13 National Institute of Standards and Technology (NIST) 18
   - 2.14 Suite de Pruebas Estadísticas de NIST 19
   - 2.15 Circuitos Caóticos 21

2.16 Circuito de Chua 22
2.17 Circuitos Caóticos en RNGs 23

3. Algoritmo Salsa20 Modificado 24
3.1 Etapa 1. Combinación y Reducción 25
3.2 Etapa 2. Función Central 27
3.3 Diseño 28
3.4 Seguridad 31

4. ACL: Algoritmo propuesto 33
4.1 Etapa 1. Combinación y Reducción 33
4.2 Etapa 2. Función Central 35
4.3 Diseño 36
4.4 Seguridad 37

5. Pruebas y resultados 39
5.1 Evaluación a Salsa20 Modificado 39
5.2 Evaluación a Salsa20 45
5.3 Evaluación a ACL 46
5.4 Requerimientos de Cómputo 51

6. Generador de Números Verdaderamente Aleatorios 55

7. Conclusiones y trabajo futuro 59

Referencias 60

Lista de Tablas

Tabla 1. Validación de la operación XOR. Probabilidades de ocurrencia de cada símbolo.....14

Tabla 2. Validación de la operación XOR. Probabilidades de ocurrencia en la combinación de los símbolos. 14

Tabla 3. Algoritmos de flujo que forman parte del portafolio de eStream. 15

Tabla 4. *Salsa20 Modificado*. Frecuencia de utilización de cada elemento de R, el cual es resultado de la Etapa de Combinación y Reducción. 30

Tabla 5. Porcentaje de secuencias binarias, obtenidas por medio del *GBPA*, que aprobaron cada prueba estadística. 41

Tabla 7. Porcentaje de secuencias binarias, después del cifrado por medio de *Salsa20 Modificado*, que aprobaron cada prueba estadística. 43

Tabla 6. Porcentaje de secuencias binarias, obtenidas por medio de Salsa20, que aprobaron cada prueba estadística. 45

Tabla 8. Porcentaje de secuencias binarias, obtenidas por medio del $GBPA_2$, que aprobaron cada prueba estadística. 47

Tabla 9. Porcentaje de secuencias binarias, después del cifrado por medio de *ACL*, que aprobaron cada prueba estadística.....49

Tabla 10. Requerimientos de Salsa20 de memoria de programa y de datos y de ciclos de CPU para cifrar un mensaje de 512 bits. 51

Tabla 11. Requerimientos de *Salsa20 Modificado* de memoria de programa y de datos y de ciclos de CPU para cifrar un mensaje de 128 y 512 bits.....52

Tabla 12. Requerimientos de *ACL* de memoria de programa y de datos, y de ciclos de CPU para cifrar un mensaje de 128 y 512 bits.....52

Tabla 13. Porcentaje de secuencias binarias, obtenidas por medio del *TRNG* del circuito de Chua, que aprobaron cada prueba estadística. 57

Lista de Figuras

Fig. 1. Circuito de Chua. 22

Fig. 2. Diagrama del RNG basado en una señal caótica. 23

Fig. 3. Forma típica de la señal del oscilador lento con jitter. 23

Fig. 4. Diagrama de la generación de la secuencia binaria pseudo aleatoria, por medio del uso de la llave secreta como semilla. 24

Fig. 5. Diagrama de la generación del texto cifrado por medio de la operación *xor* entre el mensaje y la secuencia binaria pseudo aleatoria. 25

Fig. 6. Diagrama de las modificaciones que sufren los elementos de z.....30

Fig. 7. Pseudo código para la generación de la secuencia pseudo aleatoria. 40

Fig. 8. Gráfica del porcentaje de secuencias binarias, obtenidas por medio de del *GBPA*, que aprobaron cada prueba estadística. 42

Fig. 10. Pseudo código para la generación de la secuencia pseudo aleatoria después del cifrado con *Salsa20 Modificado*. 43

Fig. 11. Gráfica del porcentaje de secuencias binarias, después del cifrado por medio de *Salsa20 Modificado*, que aprobaron cada prueba estadística.....44

Fig. 9. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del algoritmo Salsa20, que aprobaron cada prueba estadística.....46

Fig. 12. Pseudo código para la generación de la secuencia pseudo aleatoria. 47

Fig. 13. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del $GBPA_2$, que aprobaron cada prueba estadística. 48

Fig. 14. Pseudo código para la generación de la secuencia pseudo aleatoria. 49

Fig. 15. Gráfica del porcentaje de secuencias binarias, después del cifrado por medio de *ACL*, que aprobaron cada prueba estadística.....50

Fig. 16. Gráfica con una comparativa entre la cantidad de memoria de programa y de datos requerida por los tres algoritmos: Salsa20, *Salsa20 Modificado* y *ACL*. 53

Fig. 17. Gráfica con una comparativa entre la cantidad de ciclos de CPU requeridos por los tres algoritmos: Salsa20, *Salsa20 Modificado* y *ACL* para cifrar un mensaje de 512 bits. 53

Fig. 18. Circuito de Chua usado en la propuesta de TRNG.....55

Fig. 19. Grafica de los nodos V1 y V2 del Circuito de Chua. 56

Fig. 20. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del *TRNG* del circuito de Chua, que aprobaron cada prueba estadística. 58

1. Introducción

1.1 Planteamiento del problema

Los protocolos de seguridad que se utilizan en la actualidad se basan en el uso de algoritmos de llave simétrica y llave pública, los cuales llevan a cabo procesos computacionales intensivos. Actualmente la tendencia es que muchas de las aplicaciones de la Tecnología de la Información se realicen en forma de sistemas embebidos, y una gran parte de ellas sean inalámbricas, lo que las hace más vulnerables en cuanto a privacidad y seguridad de los datos [1]. Debido a que estos sistemas inalámbricos están limitados en recursos, las soluciones de seguridad aplicadas en redes tradicionales pueden no ser adecuadas para ellos [2].

El Cómputo Ubicuo es una idea que describe a los sistemas de información con interfaces capaces de soportar la inteligencia humana, con funcionamiento transparente para el usuario, y que están en todas partes para ser usados [3].

El entorno ubicuo basado en dispositivos con funcionamiento transparente para el usuario crea nuevos problemas de seguridad y privacidad, debido en parte a las limitadas interfaces de usuario existentes que informen al consumidor el nivel de recolección de datos y el uso que se les está dando [4]. Entre los riesgos está la violación de la privacidad de datos personales y de ubicación [5].

La tecnología requerida para el cómputo ubicuo se divide básicamente en tres partes: software para las aplicaciones ubicuas, redes que enlacen estas aplicaciones y dispositivos de bajo poder computacional que provean la información del entorno [6].

Sólo se puede esperar que el cómputo ubicuo tenga una penetración significativa en el mercado si los dispositivos requeridos son de bajo costo [7], lo que lleva de nuevo a la cuestión de que los dispositivos tendrán baja capacidad de cómputo.

Un mecanismo de seguridad en sistemas embebidos son los algoritmos criptográficos, y una opción es la implementación por hardware en dispositivos FPGA debido a que le da agilidad al algoritmo, seguridad física y alto rendimiento [8]. Sin embargo, este tipo de implementación tiene las desventajas de que el costo del sistema se eleva y de que hay un mayor requerimiento de energía por parte del sistema embebido.

1.2 HIPÓTESIS

Es posible proveer seguridad y privacidad a los datos que son transmitidos por un sistema embebido de tiempo real por medio de un algoritmo criptográfico que se base en operaciones computacionales no costosas y que esté optimizado para sistemas con bajo poder de cómputo.

1.3 PROPUESTA

Para este trabajo de tesis se presentan dos propuestas las cuales consisten en algoritmos de cifrado de flujo. La primera propuesta es una modificación del algoritmo Salsa20 y la segunda es un nuevo algoritmo. Los algoritmos se titulan *Salsa20 Modificado* y *ACL Algoritmo de Cifrado Ligero*.

¿Por qué se eligió el algoritmo Salsa20 entre otros algoritmos de flujo? Por tres razones principalmente:

1. Es uno de los finalistas del proyecto eStream. El algoritmo estuvo en evaluación durante 4 años.
2. Se basa en operaciones computacionalmente sencillas, que forman parte del conjunto de instrucciones de la mayoría de los microcontroladores.
3. Gusto personal sobre otros algoritmos. En lo personal lo considero un algoritmo limpio, elegante y simple ya que con pocas instrucciones y muy simples logra que pequeños cambios a nivel de bits generen un dato de salida totalmente nuevo y sin relación aparente.

1.4 Estructura de la Tesis

La organización de este trabajo de tesis se presenta a continuación:

Capítulo 1: En este capítulo se presentó el planteamiento del problema y la hipótesis sobre la cual se desarrolló el trabajo realizado.

Capítulo 2: Consiste en los antecedentes y marco teórico que se tomaron como base para las propuestas de algoritmos de cifrado y de generador de números verdaderamente aleatorios. En este capítulo se presenta una introducción a los conceptos de sistema embebido, cómputo ubicuo, criptografía y números aleatorios y sus generadores.

Capítulo 3: Describe la primera propuesta de algoritmo de cifrado de este trabajo de tesis: *Salsa20 Modificado*.

Capítulo 4: Describe la segunda propuesta de algoritmo de cifrado de este trabajo de tesis: *ACL Algoritmo de Cifrado Ligero*.

Capítulo 5: Presenta las pruebas realizadas a ambas propuestas de algoritmos y la determinación de sus requerimientos de cómputo. La evaluación de la salida de los algoritmos fue por medio de la suite de pruebas estadísticas de NIST.

Capítulo 6: Se describe una propuesta de generador de números verdaderamente aleatorios con base al circuito caótico de Chua. Se presenta además los resultados de la evaluación realizada a este generador.

Capítulo 7: Conclusiones de este trabajo de tesis y trabajo futuro sobre el generador de números verdaderamente aleatorios.

2. Antecedentes y Marco Teórico

2.1 Sistema Embebido

Un sistema embebido es un sistema de procesamiento de información que forma parte de un sistema mayor [9]. A diferencia de las computadoras de propósito general, los sistemas embebidos poseen algunas características que son únicas para cada sistema ya que son específicas a la aplicación.

Algunas características importantes en los sistemas embebidos son [10]:

1. Aplicación y dominio específicos
2. Reactivo y de Tiempo Real
3. Distribuido
4. Tamaño pequeño y bajo peso
5. Energía de alimentación limitada

En esta tesis, un sistema embebido de bajo poder computacional es aquel cuyas características son congruentes con lo siguiente:

- Procesador de 8bits hasta 16 bits
- Memoria de 4kB hasta 8kB
- Frecuencia de 20 MHz hasta 50 MHz

2.2 Cómputo Ubicuo

La idea del cómputo ubicuo consiste en la integración del cómputo en la vida diaria de las personas, siendo las máquinas y sus aplicaciones las que se adaptan al entorno de los individuos y no al contrario. La finalidad es la inmersión de la tecnología en los objetos cotidianos de manera que se rompa la barrera psicológica que actualmente

existe entre la persona y la tecnología, teniendo como consecuencia que se les adopte como algo natural. Este término fue propuesto por Mark Weiser en su artículo *The Computer for the 21st Century* [6].

Para lograr que la integración sea natural se requiere que estas aplicaciones tengan consciencia del contexto que se está desarrollando de forma que puedan adaptar su comportamiento y proveerle al usuario información o acciones acordes a la situación que se desarrolla en el momento. De acuerdo a Weiser, la tecnología requerida para el cómputo ubicuo consiste en computadoras baratas y de baja potencia, software para las aplicaciones ubicuas y redes que unan estas aplicaciones.

Las aplicaciones ubicuas pueden ser vistas como la extensión a la funcionalidad que ya poseen los objetos cotidianos.

2.3 CRIPTOGRAFÍA

Antes de hablar de criptografía es conveniente definir qué es un ataque dentro del marco de la seguridad de datos.

Ataque

Es un asalto a la seguridad del sistema que se deriva de una amenaza inteligente, es decir, un *acto inteligente* que es un intento deliberado de evadir los servicios de seguridad y violar la política de seguridad de un sistema [11].

La criptografía es un mecanismo de seguridad que podría dar una solución a los ataques a la seguridad de un sistema.

Criptografía es el estudio de técnicas matemáticas relacionadas con los aspectos de la seguridad de la información tales como confidencialidad, integridad en los datos, autentificación de entidades y autentificación del origen de los datos [12].

La Criptografía puede ofrecer las siguientes funciones [10]:

- Confidencialidad: Es el aseguramiento de que solo los usuarios autorizados pueden acceder a la información declarada como confidencial.
- Autentificación: Es la identificación de las entidades que se están comunicando en la red y del origen de la información recibida.
- Integridad de los datos: Es la verificación de que el mensaje recibido no ha sido alterado por entidades no autorizadas; tales alteraciones incluyen insertar, eliminar o sustituir contenido.
- No repudio: Es un servicio que previene que una entidad niegue que realizó ciertas acciones o que adquirió ciertos compromisos.

La criptografía trata sobre prevención y detección de engaños y otras actividades maliciosas [12].

Algunas primitivas criptográficas que ayudan a proveer seguridad a la información son los esquemas de cifrado, funciones hash y esquemas de firma digital. El enfoque de este trabajo de tesis son los algoritmos de cifrado.

Los algoritmos de cifrado se clasifican en dos tipos: algoritmos de cifrado de llave pública y algoritmos de cifrado de llave privada, también conocidos como algoritmos asimétricos y simétricos respectivamente. Debido a que los algoritmos de llave pública requieren un alto poder de cómputo y a que este proyecto de tesis está enfocado en sistemas embebidos de bajo poder computacional, la propuesta de algoritmo consiste en un algoritmo simétrico.

Algoritmos simétricos

Esquema de cifrado que consta de los conjuntos de transformaciones de cifrado $\{E_e: e \in K\}$ y de descifrado $\{D_d: d \in K\}$. Donde por cada par de llaves de cifrado y descifrado asociadas (e, d), es trivial determinar d conociendo solo e, y determinar e a partir de d. En la práctica la llave de descifrado suele ser la misma que la de cifrado [12].

Los algoritmos simétricos se clasifican a su vez en dos tipos: algoritmos de cifrado de bloque y algoritmos de cifrado de flujo [13].

Algoritmos de cifrado de bloque

Es un esquema de cifrado en el cual se divide el mensaje a cifrar en bloques de tamaño fijo, y se cifra y descifra un bloque a la vez [13].

Visto esto en forma abstracta se tienen dos conceptos. Primeramente se tiene el concepto de una Función Pseudo Aleatoria, *PRF* por sus siglas en inglés *Pseudo Random Function*. Una PRF es una función que toma como entradas una llave $\{k \in K\}$ y un elemento de $\{x \in X\}$ que es el espacio del bloque del mensaje, y produce como salida un elemento en el espacio Y.

Se tiene además el concepto de una Permutación Pseudo Aleatoria, *PRP* por sus siglas en inglés *Pseudo Random Permutation*, la cual toma un elemento de $\{k \in K\}$ y un elemento de $\{x \in X\}$, y produce como salida un elemento $\{x \in X\}$. En una PRP se tiene el requerimiento que la transformación debe ser invertible. Una PRP es un caso especial de una PRF.

De forma abstracta un algoritmo de bloques es una PRP.

Algoritmos de cifrado de flujo.

Es un esquema de cifrado donde se cifran bits individualmente [13]. Un algoritmo de cifrado de flujo puede ser visto como un algoritmo de bloques donde el tamaño del bloque es uno.

Siendo *K* el espacio de llaves, una secuencia de símbolos $k_0, k_1, \dots, k_i, \dots, k_{n-1} \in K$ se le denomina un keystream. Siendo $m_0, m_1, \dots, m_i, \dots, m_{n-1}$ el mensaje dividido en bloques de tamaño uno. El algoritmo de flujo toma el mensaje y produce el texto cifrado $c_0, c_1, \dots, c_i, \dots, c_{n-1}$, donde *ci* se obtiene de aplicar la trasformación de cifrado bajo el bit *ki* para transformar el bit *mi*.

Las ventajas de este tipo de algoritmos son que los datos se procesan un símbolo a la vez, no hay propagación de errores debido a la característica anterior, y el tamaño del texto cifrado es el mismo que el del mensaje.

Para este trabajo de tesis se eligió que el algoritmo de cifrado a diseñar fuera un algoritmo de flujo debido principalmente a la ventaja de que se procesa un símbolo a la vez lo cual lo hace adecuado para sistemas con poco poder de procesamiento o poca memoria.

Debido a que la propuesta es un algoritmo de flujo, se va a detallar su construcción.

En un algoritmo de flujo la transformación de cifrado consiste en: $c_i = E_{k_i}(m_i) = m_i + k_i$ y la de descifrado en: $m_i = D_{k_i}(c_i) = c_i + k_i$. Donde en este caso el operador + representa la suma módulo dos entre un bit del keystream y un bit del mensaje o un bit del texto cifrado. La suma módulo dos es equivalente a la operación booleana xor. La validación de la operación xor como función que permite transformar el texto plano en texto cifrado se presenta en la sección *2.10 Operación XOR*.

El keystream que se utiliza para realizar el cifrado y descifrado consiste en un número aleatorio o como mínimo un número pseudo aleatorio, el cual es denominado *PRN* por sus siglas en inglés *Pseudo Random Number*. En la práctica se suelen utilizar PRNs que se obtienen por medio de un algoritmo generador de números pseudo aleatorios, *PRNG* por sus siglas en inglés *Pseudo Random Number Generator*. En las secciones 2.4 a la 2.9 se abordan los conceptos de números aleatorios, pseudo aleatorios y sus generadores.

2.4 ALEATORIEDAD

El término aleatoriedad se puede definir como la incapacidad de comprimir o reducir una serie de números a una forma más compacta sin que haya pérdida de información. Es posible decir que una serie de números son aleatorios si el algoritmo más pequeño capaz de especificarlo para una computadora tiene la misma cantidad de bits de información que la serie de números tiene por sí misma [14].

2.5 NÚMEROS ALEATORIOS

Un número verdaderamente aleatorio, *TRN* por sus siglas en inglés *Truly Random Number*, proviene de una fuente natural indeterminista, tal que no es posible realizar predicciones útiles acerca de la salida del generador aún si el diseño de la fuente que lo genera es conocido. Dos ejemplos de fuentes que pueden generarlos son el ruido electrónico y el decaimiento radioactivo [15].

2.6 NÚMEROS PSEUDO ALEATORIOS

Un *TRN* no puede ser producido por medio de métodos aritméticos [16]. Una computadora al ser una máquina determinista y que puede estar solo en una cantidad finita de estados no es capaz de producir números verdaderamente aleatorios. Lo mejor que puede proveer una computadora es de números pseudo aleatorios. Una secuencia pseudo aleatoria es aquella que *luce* aleatoria. El periodo de la secuencia que puede producir el generador debe ser suficientemente grande tal que una secuencia finita de longitud razonable es no periódica. Y estas relativamente pequeñas secuencias no periódicas deben ser indistinguibles de una secuencia verdaderamente aleatoria. Se dijo que un secuencia es pseudo aleatoria si luce aleatoria, esto quiere decir que si la secuencia aprobó las pruebas estadísticas que evalúan la aleatoriedad [17].

2.7 GENERADOR DE NÚMEROS VERDADERAMENTE ALEATORIOS

Un generador de números verdaderamente aleatorios, *TRNG* por sus siglas en inglés *Truly Random Number Generator*, es un sistema cuya salida es una secuencia de números impredecibles, es decir números estadísticamente independientes y sin parcialidad hacia algún valor [18].

Los TRNG se basan en fenómenos indeterministas, sin embargo estas fuentes de aleatoriedad pueden tener defectos estadísticos debido a limitaciones físicas, problemas en la implementación o debido a ataques externos con el fin de manipularlos. Debido a esto, usualmente los TRNG se componen de una etapa de post procesamiento [18].

2.8 GENERADOR DE NÚMEROS PSEUDO ALEATORIOS

Un generador de números pseudo aleatorios es una función que tras ser inicializada con un TRN, denominado como *semilla*, provee una secuencia pseudo aleatoria. Un PRNG es una función determinista, cada que se le inicializa con la misma semilla produce la misma secuencia de números pseudo aleatorios [19].

Un PRNG recibe como entrada una secuencia binaria aleatoria de longitud k y produce una secuencia binaria de longitud l, donde $l \gg k$, la cual *luce* aleatoria [20].

Un concepto relacionado con la generación de números aleatorios es la entropía, la cual mide la cantidad de información que porta una secuencia, es decir, la cantidad de incertidumbre a priori de una secuencia. Debido a que una secuencia que ha sido generada por un PRNG es totalmente dependiente de la semilla usada, su entropía no puede ser mayor a la de esta. Esta es una característica que distingue a los PRNG de los TRNG [19].

Otra diferencia entre un TRNG y un PRNG es que un PRNG expande su entrada, es decir, su semilla. Mientras que un TRNG durante la etapa de post procesamiento realiza lo contrario, reduce su entrada, o dicho de una manera, destila su entropía [18].

2.9 GENERADOR DE NÚMEROS PSEUDO ALEATORIOS CRIPTOGRÁFICAMENTE SEGURO

Un generador de números pseudo aleatorios criptográficamente seguro es un PRNG que tiene el requerimiento de ser impredecible. Esto es, dado una salida de n bits $s_i, s_{i+1}, \dots, s_{i+n-1}$, no existe un algoritmo de tiempo polinomial que pueda predecir el siguiente bit s_{i+n} con una probabilidad mayor al 50%. Igualmente, dada la secuencia anterior, es computacionalmente inviable computar los bits precedentes $s_{i-1}, s_{i-2}, \dots$ [13].

2.10 OPERACIÓN XOR

La validación de la operación *xor* como función que permite transformar el texto plano en texto cifrado está dada por lo siguiente [13]:

Teorema 1:

Sea Y una variable sobre $\{0,1\}^n$.

Sea X una variable independiente y uniforme sobre $\{0,1\}^n$. X representa a una secuencia binaria aleatoria.

Entonces:

$Z = Y \oplus X$ es una variable uniforme sobre $\{0,1\}^n$

Prueba:

El caso más simple es para $n = 1$

Si la salida es uniforme, la probabilidad de que la salida sea cero es del 50%.

$$Pr[Z = 0] = \frac{1}{2}$$

Desarrollo de la prueba para el caso de $Z = 0$:

La variable Y tiene una probabilidad P_0 de ser cero, y una probabilidad P_1 de ser uno. No se conocen los valores de P_0 y P_1 y no necesariamente son 0.5 cada uno.

Al ser X una variable uniforme, cada símbolo tiene la misma probabilidad de aparecer, es decir, su probabilidad es 0.5.

Y	Pr
0	P_0
1	P_1

X	Pr
0	$\frac{1}{2}$
1	$\frac{1}{2}$

Tabla 1. Validación de la operación XOR. Probabilidades de ocurrencia de cada símbolo.

En la Tabla 2 se presenta la combinación entre los posibles valores de *Y* y *X* junto con sus probabilidades. Debido a que los valores que obtienen las variables son eventos independientes las probabilidades se multiplican. Por ejemplo, la probabilidad de que *Y* sea cero y *X* sea cero es $P_0\ x\ \frac{1}{2}$.

Y	X	Pr
0	0	$\frac{P_0}{2}$
0	1	$\frac{P_0}{2}$
1	0	$\frac{P_1}{2}$
1	1	$\frac{P_1}{2}$

Tabla 2. Validación de la operación XOR. Probabilidades de ocurrencia en la combinación de los símbolos.

Siguiendo la prueba en el caso de $Z = 0$, Z puede obtener este valor solo en dos posibles situaciones; la primera es cuando tanto *X* como *Y* son igual a cero, y la segunda en el caso contrario, cuando *X* y *Y* son igual a uno.

$$Pr[Z = 0] = Pr[(Y, X) = (0,0) \text{ ó } (Y, X) = (1,1)]$$

Al ser estos eventos mutuamente exclusivos, las probabilidades se suman:

$$Pr[(Y,X) = (0,0)] + Pr[(Y,X) = (1,1)] = \frac{P_0}{2} + \frac{P_1}{2} = \frac{P_0 + P_1}{2} = \frac{1}{2}$$

La probabilidad de que la salida sea cero es del 50%, lo que quiere decir que por medio de la operación xor se puede obtener una salida balanceada que es indistinguible de un dato aleatorio siempre y cuando una de sus entradas lo sea también.

2.11 PROYECTO ESTREAM

En 2004 surgió el proyecto eSTREAM cuya finalidad fue la identificación y promoción de algoritmos de cifrado de flujo seguros que se ajustaran a alguno de los siguientes perfiles [21] [22]:

- Perfil 1: Algoritmos de flujo con alto rendimiento para aplicaciones de software
- Perfil 2: Algoritmos de flujo para aplicaciones de hardware con recursos muy limitados

El proyecto aceptó propuestas de algoritmos de la comunidad internacional. Para que una propuesta pasara a formar parte del portafolio de algoritmos de eSTREAM fue necesario que superar tres fases de evaluación. En abril de 2008 se dio por finalizado el proyecto y se presentó un portafolio con ocho algoritmos, sin embargo, en septiembre de ese mismo año se realizó una revisión del mismo y se redujo el portafolio a los 7 algoritmos que se muestran en la siguiente tabla:

Perfil 1	**Perfil 2**
HC-128	Grain v1
Rabbit	MICKEY 2.0
Salsa20/12	Trivium
SOSEMANUK	

Tabla 3. Algoritmos de flujo que forman parte del portafolio de eStream.

El portafolio es continuamente revisado, siendo la última en enero de 2012.

2.12 ALGORITMO SALSA20

Una de las propuestas de este trabajo de tesis es una modificación del algoritmo de cifrado de flujo *Salsa20* de Daniel J. Bernstein, de forma que sea viable para sistemas embebidos de bajo poder computacional. El algoritmo consiste en una función hash que se ejecuta 10 veces en modo de conteo, *counter mode* en inglés, sobre una entrada de 64 bytes y provee una salida del mismo tamaño. La función hash es alimentada con la llave de cifrado, el nonce y el número de bloque. La salida final se utiliza para realizar la operación xor con el texto a cifrar [23] [24] [25] [26].

El algoritmo se compone de tres funciones: *rowround*, *columnround* y *doubleround*. La función doubleround consiste en ejecutar a columnround y posteriormente a rowround. La función doubleround es la función central que es ejecutada 10 veces sobre la entrada de 64 bytes.

Para describir a rowround y columnround se tiene que presentar primero a la función *quarterround*, la cual compone a estas dos funciones.

Siendo $y = (y_0, y_1, y_2, y_3)$, donde cada elemento de y es de tamaño 32 bits, entonces $quarterround(y) = (z_0, z_1, z_2, z_3)$, donde:

$z_1 = y_1 \oplus \left((y_0 + y_3) <<< 7\right)$

$z_2 = y_2 \oplus \left((z_1 + y_0) <<< 9\right)$

$z_3 = y_3 \oplus \left((z_2 + z_1) <<< 13\right)$

$z_0 = y_0 \oplus \left((z_3 + z_2) <<< 18\right)$

Donde el símbolo $a <<< b$ representa la rotación del valor a, b posiciones a la izquierda.

La función rowround se define como $rowround(y) = (z_0, z_1, z_2, z_3, \dots, z_{15})$ siendo $y = (y_0, y_1, y_2, y_3, \dots, y_{15})$, y la cual consiste en:

$(z_0, z_1, z_2, z_3) = quarterround(y_0, y_1, y_2, y_3)$

$(z_5, z_6, z_7, z_4) = quarterround(y_5, y_6, y_7, y_4)$

$(z_{10}, z_{11}, z_8, z_9) = quarterround(y_{10}, y_{11}, y_8, y_9)$

$(z_{15}, z_{12}, z_{13}, z_{14}) = quarterround(y_{15}, y_{12}, y_{13}, y_{14})$

Si y es una secuencia de 16 elementos de 32 bits cada uno, entonces $rowround(y)$ consiste también en una secuencia de 16 elementos de 32 bits cada uno.

Siendo $x = (x_0, x_1, x_2, x_3, \dots, x_{15})$, la función columnround se define como $columnround(x) = (y_0, y_1, y_2, y_3, \dots, y_{15})$ siendo $y = (y_0, y_1, y_2, y_3, \dots, y_{15})$, y la cual consiste en:

$(y_0, y_4, y_8, y_{12}) = quarterround(x_0, x_4, x_8, x_{12})$

$(y_5, y_9, y_{13}, y_1) = quarterround(x_5, x_9, x_{13}, x_1)$

$(y_{10}, y_{14}, y_2, y_6) = quarterround(x_{10}, x_{14}, x_2, x_6)$

$(y_{15}, y_3, y_7, y_{11}) = quarterround(x_{15}, x_3, x_7, x_{11})$

Si x es una secuencia de 16 elementos de 32 bits cada uno, entonces $columnround(y)$ consiste también en una secuencia de 16 elementos de 32 bits cada uno.

El algoritmo Salsa20 se define como $Salsa20(x) = x + doubleround^{10}(x)$, donde cada secuencia de 4 bytes es tratada como un elemento de 32 bits en forma little endian.

Si *k* es una secuencia de 32 o 16 bytes y *n* es una secuencia de 16 bytes, entonces $Salsa20_k(n)$ es una secuencia de 64 bytes.

Cuando k es una secuencia de 32 bytes, Salsa20 corresponde a $Salsa20_{k_0,k_1}(n) = Salsa20(\sigma_0, k_0, \sigma_1, n, \sigma_2, k_1, \sigma_3)$. Donde se tiene que $\sigma_0 = (101,120,112,97)$, $\sigma_1 = (110,100,32,51)$, $\sigma_2 = (50,45,98,121)$ y $\sigma_3 = (116,101,32,107)$, las secuencias están dadas por los valores ASCII de "expand 32-byte k".

Si k es una secuencia de 16 bytes, Salsa20 corresponde a $Salsa20_k(n) = Salsa20(\tau_0, k, \tau_1, n, \tau_2, k, \tau_3)$. Donde se tiene que $\tau_0 = (101,120,112,97)$, $\tau_1 = (110,100,32,49)$, $\tau_2 = (54,45,98,121)$ y $\tau_3 = (116,101,32,107)$, las secuencias están dadas por los valores ASCII de "expand 16-byte k".

La función de cifrado Salsa20 en modo de conteo se denota como $Salsa20_k(v) \oplus m$. Donde:

k corresponde a la llave de cifrado y es una secuencia de 32 o 16 bytes,

m es el texto plano a cifrar, o el texto cifrado a descifrar

v es el nonce, un número de mensaje único. El cual corresponde a una secuencia de 8 bytes si se le concatena una secuencia de 8 bytes que corresponde al número de bloque en el modo de conteo, quedando de la siguiente manera:

$$Salsa20_k(v,0), Salsa20_k(v,1), Salsa20_k(v,2), \dots, Salsa20_k(v, 2^{64}-1)$$

La salida implícitamente se truca al mismo tamaño de m.

2.13 NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST)

El Instituto Nacional de Estándares y Tecnología de Estados Unidos, *NIST* por sus siglas en inglés *National Institute of Standards and Technology*, es una agencia federal cuya misión es promover la innovación y la competitividad de la industria en Estados Unidos mediante el avance en la ciencia de la medición, los estándares, y la tecnología de forma que incremente la seguridad económica y mejore la calidad de vida.

El Laboratorio de Tecnología de la Información de NIST, *ITL* por sus siglas en inglés *Information Technology Laboratory*, tiene entre sus responsabilidades el desarrollo de técnicas, materiales, y manejo de estándares y guías para la seguridad costo-efectiva y privacidad de información sensitiva no clasificada en los sistemas de cómputo federales [27].

ITL pone a disposición del público en general una suite de pruebas estadísticas que permite evaluar secuencias de números aleatorios en base a ciertos criterios. Esta publicación reporta la investigación y esfuerzos en la seguridad informática realizados por el ITL, y su colaboración con la industria, gobierno y organizaciones académicas.

2.14 SUITE DE PRUEBAS ESTADÍSTICAS DE NIST

La suite de pruebas estadísticas utilizada en este trabajo de tesis es "Una Suite de Pruebas Estadísticas para Generadores de Números Aleatorios y Pseudo Aleatorios para Aplicaciones Criptográficas", su título original en inglés es *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*.

Código de la publicación:

National Institute of Standards and Technology Special Publication 800-22 revision 1a Natl. Inst. Stand. Technol. Spec. Publ. 800-22rev1a, 131 pages (April 2010)

Una prueba estadística es formulada para probar una hipótesis nula específica, denominada *H0*, siendo en este caso que la secuencia siendo probada es aleatoria. Asociada con esta hipótesis esta la hipótesis alternativa, denominada *Ha*, que en este caso es que la secuencia no es aleatoria [27].

Por cada prueba aplicada se deriva una decisión, si se rechaza *H0* o si se falla al rechazar *H0*, es decir, si el generador está o no produciendo valores aleatorios basándose en la secuencia que se produjo y evaluó.

La prueba de hipótesis estadísticas es un procedimiento que tiene dos posibles salidas, aceptar H0, la secuencia es aleatoria, o aceptar Ha, la secuencia no es aleatoria. En estos procedimientos se pueden presentar cuatro situaciones:

Siendo la secuencia aleatoria (*H0* es verdadera):

a) La secuencia es aceptada
b) La secuencia es rechazada

No siendo la secuencia aleatoria (*Ha* es verdadera):

c) La secuencia es aceptada
d) La secuencia es rechazada

A la situación *b* se le conoce como error de Tipo I y su probabilidad se denota como α, y es la probabilidad de que la secuencia fue indicada como no aleatoria cuando en realidad lo es. Esto es, la secuencia parece tener propiedades no aleatorias aun cuando un *buen* generador la produjo. Valores comunes para α en criptografía son 0.01, el cual indica que se puede esperar que una secuencia en cien sea rechazada. Esta suite utiliza un α de 0.01 en sus pruebas estadísticas.

Las pruebas estadísticas calculan un *P-value*, el cual resume la fuerza de la evidencia en contra de la *H0*. Un *P-value* ≥ 0.01 puede significar que la secuencia puede ser considerada aleatoria con una confianza del 99%. Un *P-value* < 0.01 puede significar que la secuencia puede ser considerada no aleatoria con una confianza del 99%.

Esta suite consiste en 15 pruebas, las cuales se enfocan en diferentes tipos de no aleatoriedad que pudieran existir en una secuencia. Estas son las pruebas:

1. The Frequency (Monobit) Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Tests for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Linear Complexity Test,
11. The Serial Test,
12. The Approximate Entropy Test,
13. The Cumulative Sums (Cusums) Test,
14. The Random Excursions Test, and
15. The Random Excursions Variant Test.

2.15 CIRCUITOS CAÓTICOS

Un sistema caótico es un sistema determinístico que exhibe comportamiento aleatorio [28]. Los sistemas caóticos son muy sensibles a las condiciones iniciales, donde pequeños cambios en el estado inicial causan un cambio dramático en el comportamiento total del sistema. Además, presentan patrones de oscilación complejos y delimitados los cuales tienen la particularidad de, a pesar de su complejidad, ser totalmente reproducibles cuando se presentan las mismas condiciones iniciales [29]. Un circuito caótico es aquel que presenta este comportamiento.

Para que un circuito autónomo constituido por resistencias, capacitores e inductores presente comportamiento caótico debe tener por lo menos un elemento no linear, por lo menos un resistor localmente activo y por lo menos tres elementos que almacenan energía [30].

2.16 CIRCUITO DE CHUA

En 1983 durante una estancia en Japón el investigador Leon Chua propuso un circuito que presenta comportamiento caótico, al circuito le denominó *Circuito de Chua*. En la Fig. 1 se presenta el circuito propuesto [30].

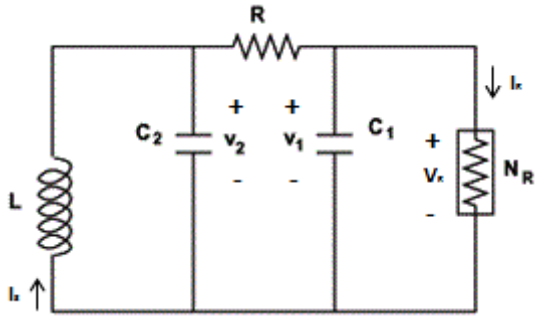

Fig. 1. Circuito de Chua.

En el circuito de Chua los elementos que almacenan energía son los dos capacitores y el inductor, el resistor localmente activo y elemento no lineal es el denominado *diodo de Chua*, que en el diagrama anterior es representado por el componente N_R. Este componente suele implementarse por medio de diodos y amplificadores operacionales.

2.17 CIRCUITOS CAÓTICOS EN RNGS

En el documento [31] se presenta una propuesta para la generación de números aleatorios utilizando una señal caótica para proveer entropía al sistema.

El sistema está constituido por dos señales oscilatorias, una de alta y la otra de baja frecuencia, y por un flip flop tipo D. La señal de baja frecuencia es utilizada como la de fuente de reloj del flip flop, el cual se encarga de muestrear la señal de alta frecuencia cuando la de baja frecuencia presenta su flanco de subida.

Para que el sistema presente entropía, el oscilador de baja frecuencia consiste en la suma de una señal triangular y la señal de salida de un circuito caótico. La aleatoriedad en este sistema se debe principalmente al ruido jitter del oscilador, el cual fue incrementado deliberadamente por medio del circuito caótico.

En la Fig. 2 se presenta el diagrama del RNG propuesto, y en la Fig. 3 se presenta la forma típica de la señal del oscilador lento con jitter.

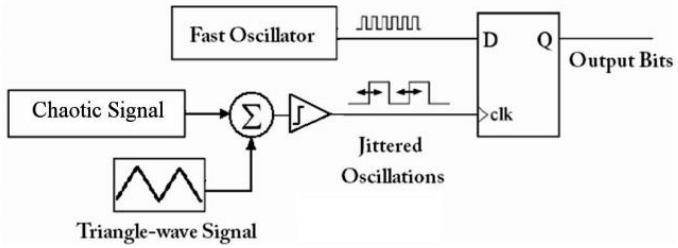

Fig. 2. Diagrama del RNG basado en una señal caótica.

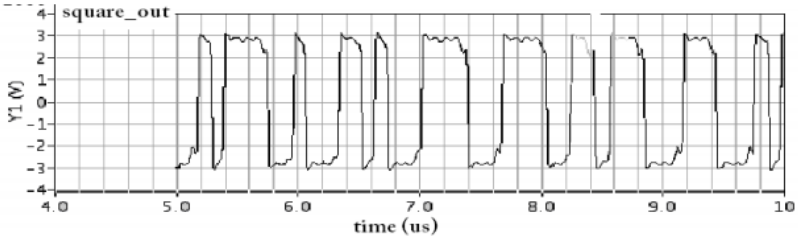

Fig. 3. Forma típica de la señal del oscilador lento con jitter.

3. Algoritmo Salsa20 Modificado

La propuesta 1 denominada *Salsa20 Modificado* consiste en una función que toma 19 bytes de entrada y produce 16 bytes de salida. Donde la salida de esta función es usada para realizar una operación *xor* con 16 bytes del texto plano y por medio de esto transformar el texto plano a texto cifrado.

Esta función es un PRNG, al cual a partir de este momento se le denominará *GBPA* por las siglas *Generador de Bits Pseudo Aleatorios*. Como se definió en la sección *2.8 Generador de Números Pseudo Aleatorios*, el *GBPA* recibe como entrada una secuencia binaria aleatoria de longitud k y produce una secuencia binaria de longitud l, donde $l \gg k$, la cual luce aleatoria.

Sea k una secuencia de 12 bytes, sea v una secuencia de 4 bytes y sea m una secuencia de l bytes $\in \{0,1,\ldots,128\ x\ 2^{24}\}$. La función de cifrado de m con un nonce v bajo la llave k, es una secuencia de l bytes, donde m es el texto plano a cifrar o el texto cifrado a descifrar, v es un valor único para cada mensaje y k es la llave secreta de cifrado, la cual cumple la función de ser la semilla del *GBPA*.

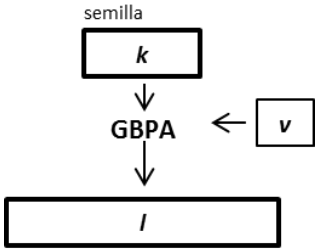

Fig. 4. Diagrama de la generación de la secuencia binaria pseudo aleatoria, por medio del uso de la llave secreta como semilla.

La *GBPA* produce la secuencia binaria pseudo aleatoria en bloques de 16 bytes, pudiendo generar una secuencia de tamaño máximo de 128×2^{24} bits por nonce. Para esto, se ejecuta la *GBPA* en *Modo de Conteo* lo cual consiste en anexar al nonce un

contador que se incrementa por cada bloque generado por la *GBPA*, el contador es de 3 bytes de tamaño.

$$GBPA_k(v,0) \,||\, GBPA_k(v,1)|| \ldots ||\, GBPA_k(v,i)|| \;\ldots ||\, GBPA_k(v,2^{24-1})$$

$$Texto\ cifrado = GBPA_k(v) \oplus m$$

La secuencia binaria se trunca al tamaño del mensaje, el texto cifrado es del mismo tamaño que el mensaje.

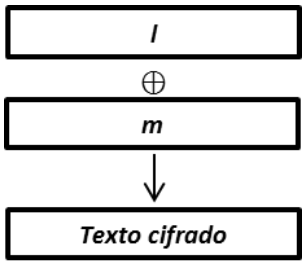

Fig. 5. Diagrama de la generación del texto cifrado por medio de la operación *xor* entre el mensaje y la secuencia binaria pseudo aleatoria.

3.1 Etapa 1. Combinación y Reducción

La *GBPA* utiliza dos valores constantes T_0 y T_1 de 2 bytes cada uno. Estos proveen un soporte al algoritmo ante el uso de llaves débiles. Sus valores son $T_0 = (84{,}48)$ y $T_1 = (116{,}49)$, los cuales están dados por "T0" y "t1" respectivamente.

La *GBPA* consiste en $GBPA_k(v) = GBPA(k, v, T_0, T_1)$ lo cual equivale a 23 bytes, y produce una salida de 16 bytes.

La primera etapa del algoritmo es una reducción de la entrada al tamaño del bloque de salida. No se descarta ningún bit, para llevar a cabo la reducción se realiza una combinación entre algunos de los parámetros de entrada.

Dado $T_0 = (T_{0_1}, T_{0_0})$, $T_1 = (T_{1_1}, T_{1_0})$, $k = k_{11}, k_{10}, \dots, k_0$, $v = v_3, v_2, \dots, v_0$ y el contador $c = c_2, c_1, c_0$, esta etapa se define de la siguiente manera:

$R_3 = v_3 + c_0 + T_{1_1}$

$R_2 = v_2 + c_1 + T_{1_0}$

$R_1 = v_1 + c_2 + T_{0_1}$

$R_0 = v_0 + k_{11} + T_{0_0}$

Esta forma de combinación permite que los bytes que son más propensos a tener valores de cero, tales como la parte más significativa del contador y del nonce, adquieran nuevos valores no nulos, y por consecuente, se alimente a las rondas del algoritmo con valores con mayor entropía.

El byte con mayor entropía es R_0 ya que se obtuvo a partir de la llave secreta y de la parte menos significativa del nonce, la cual es la más cambiante. Debido a esto, es uno de los bytes que tiene mayor frecuencia de uso en el algoritmo.

Posteriormente se realiza una segunda combinación entre los elementos de R con la idea de prevenir que sus valores sean muy cercanos a los iniciales, es decir, a los valores de las constantes. Esto es importante cuando en el nonce y contador se encuentran los bytes más significativos en cero.

Dado el símbolo $a >>> b$ que representa la rotación del valor a, b posiciones a la derecha. La segunda combinación se define como $R = R + (R >>> 8)$, quedando de la siguiente manera:

$R >>> 8 = (R_0, R_3, R_2, R_1)$

$R_0 = R_0 + R_1$

$R_1 = R_1 + R_2$

$R_2 = R_2 + R_3$

$R_3 = R_3 + R_{0_{prev}}$

$R = (R_3, R_2, R_1, R_0)$

$R_{0_{prev}}$ se refiere al valor del elemento R_0 antes de realizar la adición de R_1.

3.2 Etapa 2. Función Central

La función central que utiliza este algoritmo para la generación de la secuencia binaria pseudo aleatoria es la denominada función *quarterround* del algoritmo *Salsa20*. Considerando que $y = (y_0, y_1, y_2, y_3)$, la función se define de la siguiente manera:

$z_1 = y_1 \oplus \left((y_0 + y_3) <<< 7\right)$

$z_2 = y_2 \oplus \left((z_1 + y_0) <<< 9\right)$

$z_3 = y_3 \oplus \left((z_2 + z_1) <<< 13\right)$

$z_0 = y_0 \oplus \left((z_3 + z_2) <<< 18\right)$

Donde el símbolo $a <<< b$ representa la rotación del valor a, b posiciones a la izquierda. La *GBPA* utiliza esta función con diferentes parámetros de entrada y con diferentes tamaños que en el caso de Salsa20.

En el caso de la *GBPA* se define $y_n = \{0,1\}^{32}$, $z = (z_3, z_2, z_1, z_0)$, con $z_n = \{0,1\}^{32}$, donde:

$y_0 = (k_2, k_1, k_0, R_0)$

$y_1 = (R_2, k_5, k_4, k_3)$

$y_2 = (k_8, k_7, k_6, R_1)$

$y_3 = (R_3, k_{11}, k_{10}, k_9)$

$z = quarterround(y)$.

Se define además a $x = (x_3,\ x_2,\ x_1,\ x_0)$, donde $x = y$. A x se le denomina el *estado inicial*.

Se ejecutan 10 rondas de la función *quarterround*, en las primeras 9 rondas se actualiza a y convirtiéndose en $y_3 = z_3,\ y_2 = z_2,\ y_1 = z_1,\ y_0 = z_0$.

Finalmente se realiza $z = x + (z <<< 8)$, quedando de la siguiente manera:

$z <<< 8 = (z_2, z_1, z_0, z_3)$

$z_3 = x_3 + z_2$

$z_2 = x_2 + z_1$

$z_1 = x_1 + z_0$

$z_0 = x_0 + z_3$

$z = (z_3, z_2, z_1, z_0)$

La salida del *GBPA* es z.

3.3 DISEÑO

Las operaciones de la etapa de *Combinación y Reducción* consisten en sumas y rotaciones.

Se eligió a la suma ya que conforme se va realizando la adición de los operandos, bit a bit se puede o no generar un acarreo el cual se agregaría al bit inmediato más significativo, esto quiere decir que, a excepción del bit menos significativo, cada bit del

resultado de esta etapa depende no solo de los bits inmediatos de los operandos sino también de los valores de los bits anteriores. Los valores de T_0 y T_1 son contantes, pero el nonce, contador y la llave no, por lo que cambios, que pueden ser el incremento en uno en el nonce entre un cifrado y el anterior, causan que la salida sea diferente ya que estos cambios, aún pequeños, se fueron propagando entre los bits del resultado de esta etapa y posteriormente en la *Función Central*. En la etapa de evaluación de esta propuesta de algoritmo se estuvo trabajando con secuencias que se obtuvieron al alimentar al algoritmo con entradas que estaban estrechamente relacionadas, y los resultados obtenidos fueron satisfactorios. Esto se describe con mayor amplitud en la sección *5.1 Evaluación a Salsa20 Modificado*.

Respecto a la operación rotación, esta nos permite propagar cambios, que en este caso son entre la parte menos significativa del dato a la más significativa, y sin que se pierda ningún bit del dato, lo que es contrario a la operación corrimiento.

Tanto la suma como la rotación son operaciones computacionalmente sencillas para un procesador.

Respecto a las rotaciones en la *Función Central*, en este caso se busca difundir los cambios de los bits más significativos a los menos significativos ya que los más significativos tienen una mayor entropía debido a los acarreos que se propagan posición por posición. Las distancias de las rotaciones son constantes, y al ser ninguna múltiplo de 8, ningún byte queda intacto lo que ocasiona que se vayan a producir mayores cambios en los valores tras aplicar la operación *xor*.

En cuanto a la secuencia *R* que se obtiene en la etapa de *Combinación y Reducción,* después de ejecutarse las 10 rondas y la suma final, la frecuencia de utilización de cada elemento de *R* en la *GBPA* es la siguiente:

Elemento	Frecuencia
R_3	71
R_2	61
R_1	41
R_0	81

Tabla 4. *Salsa20 Modificado*. Frecuencia de utilización de cada elemento de R, el cual es resultado de la Etapa de Combinación y Reducción.

El byte R_0 es el que tiene mayor entropía, y por lo tanto se buscó que de ellos fuera el tuviera una mayor frecuencia de uso.

El diseño del algoritmo sigue el patrón de Salsa20 en cuanto a las modificaciones a los elementos de z, que vistas en alto nivel, siguen un modelo matricial similar al del autor.

Por cada ronda, las modificaciones siguen este patrón:

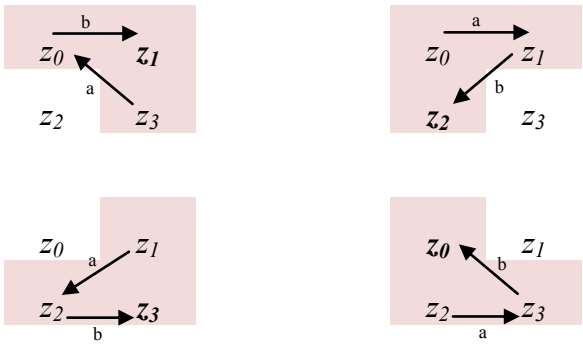

Fig. 6. Diagrama de las modificaciones que sufren los elementos de z.

En el diagrama, las etiquetas *a* y *b* denotan el orden en que son utilizados los elementos en la sentencia, siendo los elementos de la etiqueta *a* los que se utilizan primero, y *b* los que se utilizan después.

Los símbolos en negritas son quienes al final de la sentencia reciben el nuevo valor, el cual es dependiente de su propio valor y el del resto de los símbolos involucrados.

Estas modificaciones ocurren para cada ronda y siguiendo ese mismo patrón.

Esta propuesta de algoritmo puede ser vista como una simplificación del algoritmo original Salsa20.

3.4 SEGURIDAD

La cantidad de operaciones que realiza el algoritmo así como la cantidad de movimientos de datos, por ejemplo las distancias de las rotaciones, son constantes y no dependen de los valores de los parámetros de entrada, debido a esto los ataques contra la seguridad del algoritmo tales como los *Ataques en Base a Tiempo*, en inglés *Timing attack*, no son viables ya que estos se basan en medir las variaciones de los tiempos de ejecución; cuando el tiempo de ejecución de un algoritmo no es constante, este puede filtrar información acerca de los parámetros secretos involucrados [32].

Respecto a ataques por llaves débiles, el algoritmo utiliza las constantes T_0 y T_1 para sobrellevar estas situaciones, en la etapa de evaluación se alimentó al algoritmo con llaves débiles y los resultados obtenidos fueron satisfactorios. En la sección *5.1 Evaluación a Salsa20 Modificado* se aborda esto con mayor amplitud.

En 1949 Claude Shannon identificó dos propiedades de los algoritmos de cifrado para que sean seguros, a estas propiedades las denominó *Confusión* y *Difusión* [33]. El término confusión se refiere a que debe existir una relación compleja y envolvente entre la llave de cifrado y la salida del algoritmo. Difusión trata sobre disipar o difundir las propiedades estadísticas del mensaje de entrada a cifrar sobre todo el texto de salida.

En esta propuesta de algoritmo, la confusión se logra por medio de la utilización de la llave de cifrado como parámetro de entrada en cada una de las etapas del algoritmo.

En la etapa de Combinación y Reducción se utiliza un byte de la llave con el fin de añadir entropía al símbolo R_0, el cual es el más utilizado de los cuatro que forman la salida de esta etapa. En las diez rondas del algoritmo la llave se utiliza completa y ampliamente ya que 12 de los 16 bytes de entrada corresponden a la llave en su estado inicial, $llave_{t=0}$, o en un estado transformado, $llave_{t=1..10}$; la salida de cada ronda consiste mayoritariamente en la permutación de la llave secreta. Y por último, en la etapa final, a la salida de la última ronda, con $llave_{t=10}$, se le suma el estado inicial, con $llave_{t=0}$, causando una mayor dependencia con la llave de cifrado.

La difusión, que consiste en difundir las características estadísticas del mensaje, se logra por medio de la utilización de la operación lógica xor, cuya comprobación se presentó anteriormente.

4. ACL: Algoritmo propuesto

La propuesta 2, denominada *ACL Algoritmo de Cifrado Ligero*, consiste en la función $GBPA_2$ que toma 23 bytes de entrada y produce 16 bytes de salida. La salida de esta función es usada para realizar una operación xor con 16 bytes del texto plano y por medio de esto transformar el texto plano a texto cifrado.

A diferencia de la propuesta anterior, este algoritmo utiliza una llave de 16 bytes. Sea k una secuencia de 16 bytes, sea v una secuencia de 4 bytes y sea m una secuencia de l bytes $\in \{0,1,\dots,2^{24}\}$. La función de cifrado de m con un nonce v bajo la llave k, es una secuencia de l bytes, donde m es el texto plano a cifrar o el texto cifrado a descifrar, v es un valor único para cada mensaje y k es la llave secreta de cifrado, la cual cumple la función de ser la semilla del $GBPA_2$.

El $GBPA_2$ produce la secuencia binaria pseudo aleatoria en bloques de 16 bytes. Pudiendo generar una secuencia de tamaño máximo de 128×2^{24} bits por nonce. Para esto, se ejecuta el $GBPA_2$ en *Modo de Conteo*, utilizando el parámetro de entrada contador, el cual es de 3 bytes de tamaño.

4.1 Etapa 1. Combinación y Reducción

En esta propuesta se reutilizó el diseño de la Etapa de Combinación y Reducción que forma parte del algoritmo anterior, con modificaciones.

El $GBPA_2$ utiliza los dos valores constantes $T_0 = (84,48)$ y $T_1 = (116,49)$. Y se describe de la siguiente manera:

$$GBPA_{2_k}(v) = GBPA_2(k, v, T_0, T_1),$$

lo cual equivale a 27 bytes, y produce una salida de 16 bytes.

Dado $T_0 = \left(T_{0_1}, T_{0_0}\right)$, $T_1 = \left(T_{1_1}, T_{1_0}\right)$, $k = k_{15}, k_{14}, \dots, k_0$, $v = v_3, v_2, \dots, v_0$ y el contador $c = c_2, c_1, c_0$, esta etapa se define de la siguiente manera:

$R_3 = v_3 + c_0 + T_{1_1} + k_{15}$

$R_2 = v_2 + c_1 + T_{1_0} + k_{14}$

$R_1 = v_1 + c_2 + T_{0_1} + k_{13}$

$R_0 = v_0 + k_5 + T_{0_0} + k_{12}$

En este diseño los cuatro elementos de R tienen un alto nivel de entropía al haberse generado todos a partir de la llave secreta.

La segunda combinación se define como $R = R + (R >>> 8)$, quedando de la siguiente manera:

$R >>> 8 = (R_0, R_3, R_2, R_1)$

$R_0 = R_0 + R_1$

$R_1 = R_1 + R_2$

$R_2 = R_2 + R_3$

$R_3 = R_3 + R_{0_{prev}}$

$R = (R_3, R_2, R_1, R_0)$

$R_{0_{prev}}$ se refiere al valor del elemento R_0 antes de realizar la adición de R_1.

4.2 ETAPA 2. FUNCIÓN CENTRAL

La equivalencia entre la Función Central de esta propuesta contra la anterior es solo en el tipo de operaciones en las que se basan: suma, rotación y xor, ya que respecto a la serie de transformaciones que realizan no mantienen relación.

Dado el símbolo $a >> b$ que representa el corrimiento del valor a, b posiciones a la derecha; y considerando que $y = (y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$, la Función Central se define de la siguiente manera:

$y_0 = y_0 \oplus \big((y_7 <<< 3) + (y_1 >> 4)\big)$

$y_1 = y_1 \oplus \big((y_2 + y_0) <<< 7\big)$

$y_2 = y_2 \oplus \big((y_3 + y_1) <<< 11\big)$

$y_3 = y_3 \oplus \big((y_4 + y_2) <<< 3\big)$

$y_4 = y_4 \oplus \big((y_5 + y_3) <<< 7\big)$

$y_5 = y_5 \oplus \big((y_6 + y_4) <<< 9\big)$

$y_6 = y_6 \oplus \big((y_7 + y_5) <<< 3\big)$

$y_7 = y_7 \oplus \big((y_0 + y_6) <<< 7\big)$

En el caso del $GBPA_2$ se define $y_n = \{0,1\}^{16}$.

Considerando al símbolo $a << b$ que representa el corrimiento del valor a, b posiciones a la izquierda, y al símbolo $a \mid b$ que representa la operación lógica *or* entre el valor a y b. Se tiene que y equivale a:

$y_0 = (k_0 << 8) \mid R_0$

$y_1 = (k_2 << 8) \mid k_1$

$y_2 = (k_3 << 8) \mid R_1$

$y_3 = (k_5 << 8) \mid k_4$

$y_4 = (k_6 << 8) \mid R_2$

$y_5 = (k_8 << 8) \mid k_7$

$y_6 = (k_9 << 8) \mid R_3$

$y_7 = (k_{11} << 8) \mid k_{10}$

Se define además a $x = (x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$, donde $x = y$. A x se le denomina el *estado inicial*.

Se ejecutan 32 rondas de la Función Central, en cada ronda se actualizan todos los elementos de y.

Finalmente se realiza $y = x + y$, quedando de la siguiente manera:

$y_0 = y_0 + x_0$

$y_1 = y_1 + x_1$

$\vdots$

$y_7 = y_7 + x_7$

La salida del $GBPA_2$ es y.

4.3 DISEÑO

El algoritmo *ACL* sigue un esquema similar a la propuesta *Salsa20 Modificado*, es decir, consiste principalmente en una rotación de la suma de dos elementos de y, y posteriormente en una xor de este valor con el previo que tenía el elemento. La idea en *ACL* es la posibilidad de obtener un nivel de seguridad afín entre el algoritmo Salsa20 y

Salsa20 Modificado pero con menores requerimientos de cómputo. El algoritmo *ACL* consistió en replicar estos algoritmos alterando los procesos que realizan amplias modificaciones sobre los datos, a muchos pequeños procesos con pequeños cambios y pequeño impacto en el procesador de forma que al final se obtenga el mismo resultado pero con menor costo computacional. También se buscó que operen sobre parámetros de menor tamaño de forma que le sea posible al procesador mapear en sus registros internos los datos que se están procesando.

Las distancias de las rotaciones son diferentes a las de los algoritmos anteriores, aunque igualmente se buscó que causaran modificaciones entre *medio-bytes*, es decir, que el resultado no fuera simplemente la sustitución de un elemento de y por otro. Las distancias se eligieron de forma que los bits más significativos quedaran en posiciones menos significativas, ya que los MSB son los más cambiantes, es decir, que los MSB se fueran posicionando en los tres nibbles menos significativos.

La estructura general de la Función Central del algoritmo es una especie de doble enlace con retroalimentación. Cada cambio en un elemento de y depende de los dos elementos inmediatos, donde uno de los cuales utilizó a su vez el valor de ese elemento, es por eso que se dice que el enlace es retroalimentado.

4.4 SEGURIDAD

Al igual que en *Salsa20 Modificado*, la cantidad de operaciones que realiza el algoritmo y la cantidad de movimientos de datos son constantes y no dependen de los valores de los parámetros de entrada, debido a esto los ataques contra la seguridad del algoritmo tales como los *Ataques en Base a Tiempo* no son viables ya que estos se basan en medir las variaciones de los tiempos de ejecución.

Respecto a ataques por llaves débiles, el algoritmo utiliza también las constantes T_0 y T_1 para sobrellevar estas situaciones, en la etapa de evaluación se alimentó al algoritmo con llaves débiles y los resultados obtenidos fueron satisfactorios. En la sección *5.3 Evaluación a ACL* se aborda esto con mayor amplitud.

Respecto a las propiedades Confusión y Difusión identificadas por Shannon, la confusión se logra mediante la utilización de la llave secreta en cada uno de los procesamientos que realiza el algoritmo y haciendo que cada uno de los elementos de y este conformado por una parte de la llave de cifrado. En cuanto a la difusión, esta se logra por medio de la utilización de la operación lógica xor, cuya comprobación se presentó en la sección 2.10 *Operación XOR*.

5. Pruebas y resultados

En este capítulo se presentan los resultados de las evaluaciones realizadas a *Salsa20 Modificado* y *ACL*, las cuales fueron por medio de la suite de pruebas estadísticas de la NIST. Se realizó también una evaluación a Salsa20 para tener una referencia. Se presenta además los requerimientos de cómputo de los algoritmos incluyendo una comparativa con respecto a los de Salsa20.

5.1 EVALUACIÓN A SALSA20 MODIFICADO

Para la evaluación del algoritmo *Salsa20 Modificado* se tomó como premisa el *Teorema 1* que se presentó en la sección 2.10 *Operación XOR*, el cual dicta que la operación xor produce una salida uniforme sobre $\{0,1\}^n$ si una de sus entradas también lo es. Por lo que se procedió a realizar la evaluación de la secuencia de salida que produce el algoritmo para determinar si es una salida uniforme, es decir, una secuencia que presenta características similares a las secuencias aleatorias de forma que es indistinguible de una de ellas. Si el algoritmo supera la evaluación, debido al *Teorema 1* se puede determinar que la salida que se obtiene tras la operación xor también será uniforme sobre $\{0,1\}^n$.

Para la evaluación se requieren por lo menos 1,000 secuencias binarias. Para obtenerlas se realizó una implementación del algoritmo en lenguaje C, y se realizaron múltiples ejecuciones con diferentes parámetros de entrada.

La secuencia binaria pseudo aleatoria a evaluar se obtuvo alimentando al algoritmo con *parámetros débiles*, esto es, llaves con muy estrecha relación entre ellas y llaves débiles con la mayoría de sus bytes en cero. Y lo mismo para el caso del nonce y contador que se manejaron con valores relacionados entre cada ejecución del algoritmo. La finalidad de esto es el aseguramiento de que el algoritmo es capaz de generar

secuencias pseudo aleatorias aun cuando sus entradas no tendrían, de cierta manera, estas características.

Se realizaron 8,000,000 de ejecuciones del algoritmo, cada ejecución produce una salida de tamaño 128 bits pseudo aleatorios, se concatenó la salida de cada ejecución en una sola secuencia de 1,024,000,000 bits.

Se presenta el pseudo código para la obtención de la secuencia pseudo aleatoria:

```
llave = 0
nonce = 0
contador = 0
desde i = 1 hasta 200
      desde j = 1 hasta 200
            desde k = 1 hasta 200
                  GBPA(llave, nonce, contador)
                  salida z
                  nonce = nonce + 1
            fin
            contador = contador + 1
            llave = llave + 1
      fin
fin
```

Fig. 7. Pseudo código para la generación de la secuencia pseudo aleatoria.

Como puede verse en la Fig. 7, los valores que toman los tres parámetros de entrada, llave, nonce y contador, están estrechamente relacionados en cada ejecución respecto a sus valores previos.

En la evaluación de la secuencia binaria pseudo aleatoria se utilizó la suite de pruebas estadísticas de NIST, que se describió en la sección *2.14 Suite de Pruebas Estadísticas de NIST*.

Cada prueba requiere una cantidad mínima de bits para poder realizar la evaluación, las pruebas *Overlapping Template Matching Test*, *Linear Complexity Test*, *Random Excursions Test* y *Random Excursions Variant Test* requieren que como mínimo

la secuencia sea de tamaño 1,000,000 de bits. Debido a esto y para tener por lo menos las 1,000 secuencias binarias, la secuencia de salida obtenida se dividió en 1,024 secuencias a evaluar, de 1,000,000 de bits cada una.

En la siguiente tabla se presenta el porcentaje de secuencias binarias que aprobaron cada prueba estadística.

Prueba	Porcentaje
Frequency	99.1211
Frequency within a Block	99.0234
The Runs	99.1211
Longest-Run-of-Ones in a Block,	99.2188
Binary Matrix Rank	98.9258
Discrete Fourier Transform	99.2188
Non-overlapping Template Matching	98.9522
Overlapping Template Matching	98.5352
Maurer's "Universal Statistical" Test	98.1445
Linear Complexity	99.2188
Serial Test	98.5352
Approximate Entropy	98.9258
Cusums	99.17
Random Excursions Test	98.6736
Random Excursions Variant	98.9193

Tabla 5. Porcentaje de secuencias binarias, obtenidas por medio del *GBPA*, que aprobaron cada prueba estadística.

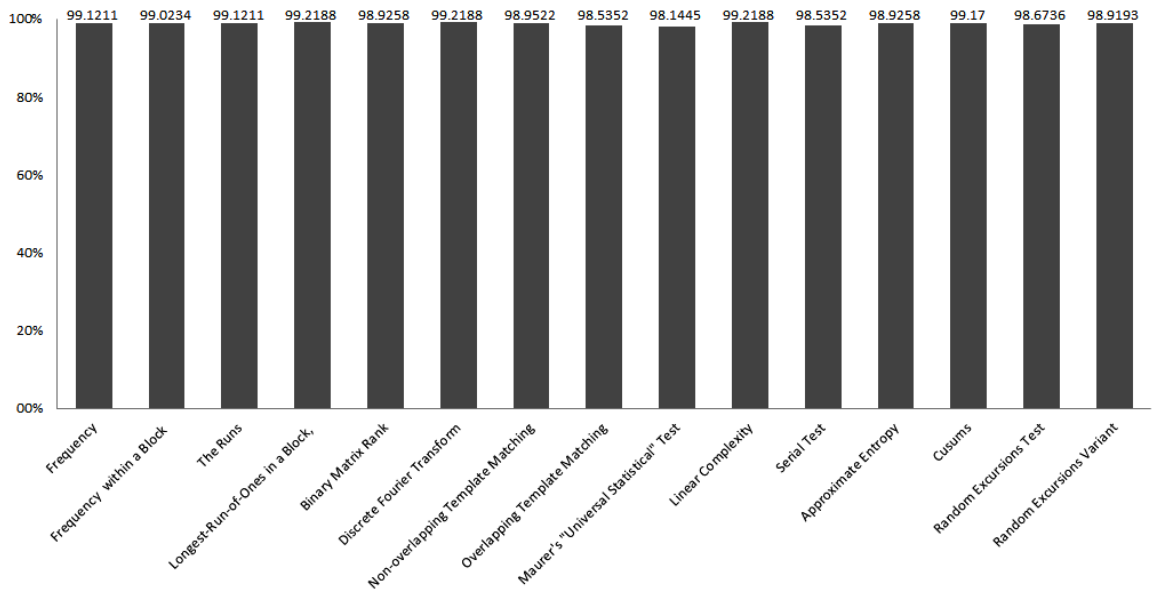

Fig. 8. Gráfica del porcentaje de secuencias binarias, obtenidas por medio de del *GBPA*, que aprobaron cada prueba estadística.

En base a estos resultados, se tiene un porcentaje global de aprobación del 98.9136.

A pesar de que el Teorema 1 dicta que mientras una de las entradas sea una variable uniforme la salida también lo será, y de que se comprobó que la salida del algoritmo provee esa variable uniforme, se procedió aun así a realizar una evaluación de la salida que se obtiene al realizar la operación xor, y los resultados fueron aprobatorios tal como se esperaban debido a lo que indica el Teorema 1.

El mensaje que se utilizó para realizar el cifrado se define como $m = (m_0, m_1, m_2, m_3)$ donde $m = \{0,1\}^{128}$, y los valores particulares de inicialización para cada elemento son $m_n = (8000000)$. Después de cada cifrado, se decremento en uno el valor de cada elemento de m. La única razón por la cual se eligió este valor de inicialización fue para que al finalizar la ronda de ejecuciones del algoritmo, el mensaje llegara a 0.

```
llave = 0
nonce = 0
contador = 0
desde i = 1 hasta 200
      desde j = 1 hasta 200
            desde k = 1 hasta 200
                  Salsa20_modificado(llave, nonce, contador)
                  nonce = nonce + 1
                  m = m - 1
            fin
            contador = contador + 1
            llave = llave + 1
      fin
fin
```

Fig. 10. Pseudo código para la generación de la secuencia pseudo aleatoria después del cifrado con *Salsa20 Modificado*.

Prueba	Porcentaje
Frequency	98.9258
Frequency within a Block	99.0234
The Runs	99.5117
Longest-Run-of-Ones in a Block,	98.5352
Binary Matrix Rank	99.0234
Discrete Fourier Transform	99.3164
Non-overlapping Template Matching	99.0525
Overlapping Template Matching	98.7305
Maurer's "Universal Statistical" Test	98.7305
Linear Complexity	99.6094
Serial Test	98.9258
Approximate Entropy	98.6328
Cusums	98.9746
Random Excursions Test	98.9114
Random Excursions Variant	99.3088

Tabla 7. Porcentaje de secuencias binarias, después del cifrado por medio de *Salsa20 Modificado*, que aprobaron cada prueba estadística.

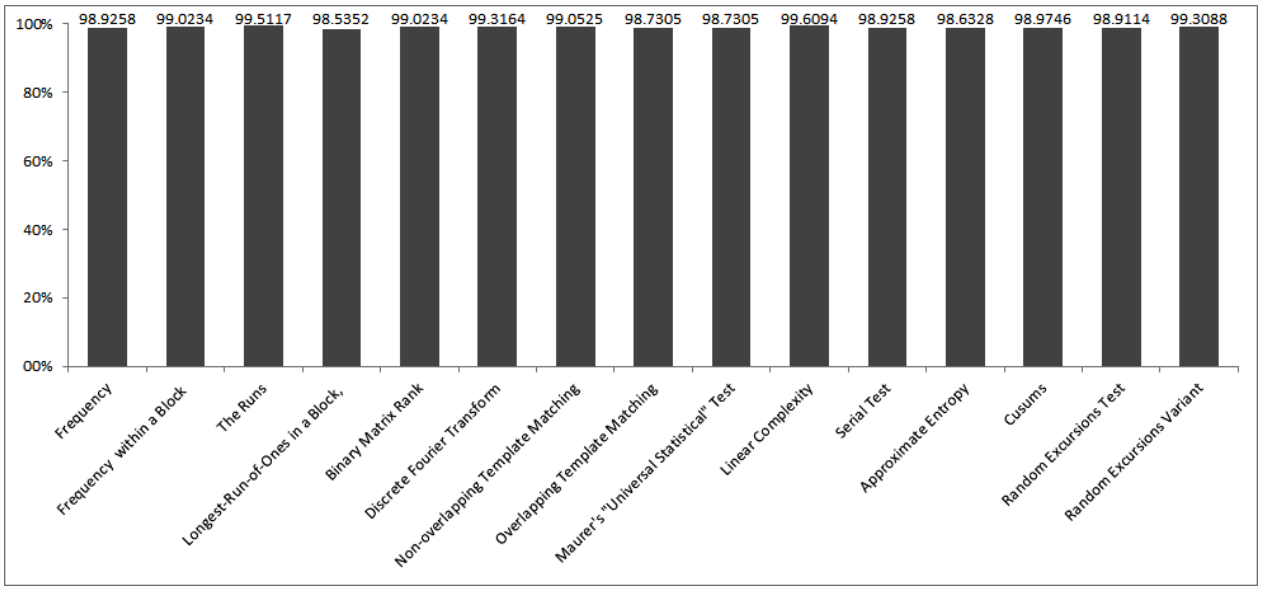

Fig. 11. Gráfica del porcentaje de secuencias binarias, después del cifrado por medio de *Salsa20 Modificado*, que aprobaron cada prueba estadística.

El porcentaje de aprobación global de *Salsa20 Modificado* es de 99.0141.

En base a los resultados de las pruebas estadísticas, las cuales fallaron en demostrar que la secuencia binaria no es aleatoria, se puede concluir que esta propuesta de algoritmo es capaz de generar secuencias que presentan características similares a las secuencias verdaderamente aleatorias, de forma que son indistinguibles de ellas. Sabiendo esto, y de acuerdo al Teorema 1, se concluye que el algoritmo de cifrado *Salsa20 Modificado* presenta seguridad en base a que la salida cifrada que provee es indistinguible de una secuencia aleatoria.

5.2 EVALUACIÓN A SALSA20

Para tener un punto de referencia, se realizaron las mismas pruebas estadísticas a una secuencia binaria pseudo aleatoria obtenida por medio del algoritmo Salsa20, manejando los mismos criterios para generarla que los utilizados en la propuesta *Salsa20 Modificado*. Se presentan los resultados de esta evaluación en la siguiente tabla.

Prueba	Porcentaje
Frequency	98.9258
Frequency within a Block	98.9258
The Runs	98.8281
Longest-Run-of-Ones in a Block,	99.2188
Binary Matrix Rank	99.805
Discrete Fourier Transform	99.0234
Non-overlapping Template Matching	99.0116
Overlapping Template Matching	99.3164
Maurer's "Universal Statistical" Test	98.6328
Linear Complexity	99.0234
Serial Test	98.7793
Approximate Entropy	98.7305
Cusums	99.0723
Random Excursions Test	99.0339
Random Excursions Variant	99.0799

Tabla 6. Porcentaje de secuencias binarias, obtenidas por medio de Salsa20, que aprobaron cada prueba estadística.

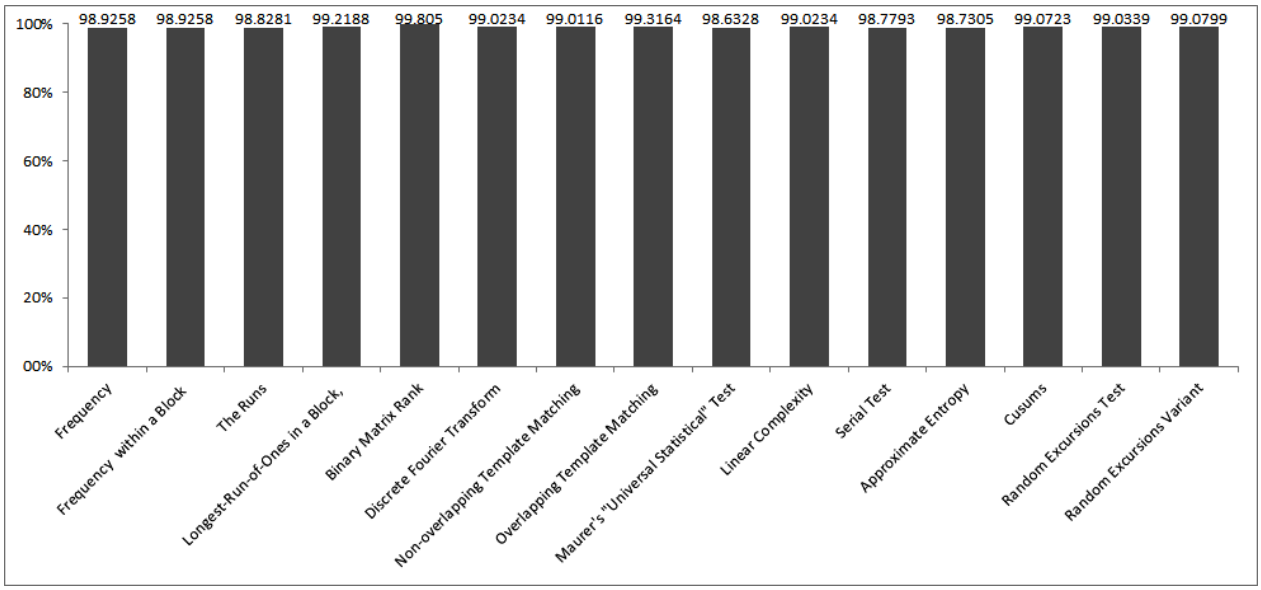

Fig. 9. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del algoritmo Salsa20, que aprobaron cada prueba estadística.

El porcentaje de aprobación global de Salsa20 es de 99.0271.

5.3 EVALUACIÓN A ACL

Para la evaluación del algoritmo *ACL* se tomó igualmente como premisa el *Teorema 1* que se presentó en la sección 2.10 *Operación XOR* y se utilizó la suite de pruebas estadísticas de NIST.

Se implementó el algoritmo en lenguaje C. La secuencia binaria pseudo aleatoria se obtuvo por medio de 8,000,000 de ejecuciones del algoritmo, alimentándolo con los mismos parámetros utilizados en la evaluación de *Salsa20 Modificado*. Cada ejecución produce una salida de tamaño 128 bits pseudo aleatorios, se concatenó la salida de cada ejecución en una sola secuencia de 1,024,000,000 bits. El pseudo código se presenta en la Fig. 12.

```
llave = 0
nonce = 0
contador = 0
desde i = 1 hasta 200
     desde j = 1 hasta 200
          desde k = 1 hasta 200
               GBPA2(llave, nonce, contador)
               salida y
               nonce = nonce + 1
          fin
          contador = contador + 1
          llave = llave + 1
     fin
fin
```

Fig. 12. Pseudo código para la generación de la secuencia pseudo aleatoria.

Para la evaluación se dividió la secuencia binaria pseudo aleatoria en 1024 secuencias de 1,000,000 de bits cada una. En la siguiente tabla se presenta el porcentaje de secuencias que aprobaron cada prueba estadística.

Prueba	Porcentaje
Frequency	98.8281
Frequency within a Block	99.3164
The Runs	99.0234
Longest-Run-of-Ones in a Block,	98.9258
Binary Matrix Rank	99.1211
Discrete Fourier Transform	98.6328
Non-overlapping Template Matching	98.999
Overlapping Template Matching	98.2422
Maurer's "Universal Statistical" Test	99.3164
Linear Complexity	98.7305
Serial Test	98.877
Approximate Entropy	98.5352
Cusums	99.0234
Random Excursions Test	99.0718
Random Excursions Variant	99.2208

Tabla 8. Porcentaje de secuencias binarias, obtenidas por medio del $GBPA_2$, que aprobaron cada prueba estadística.

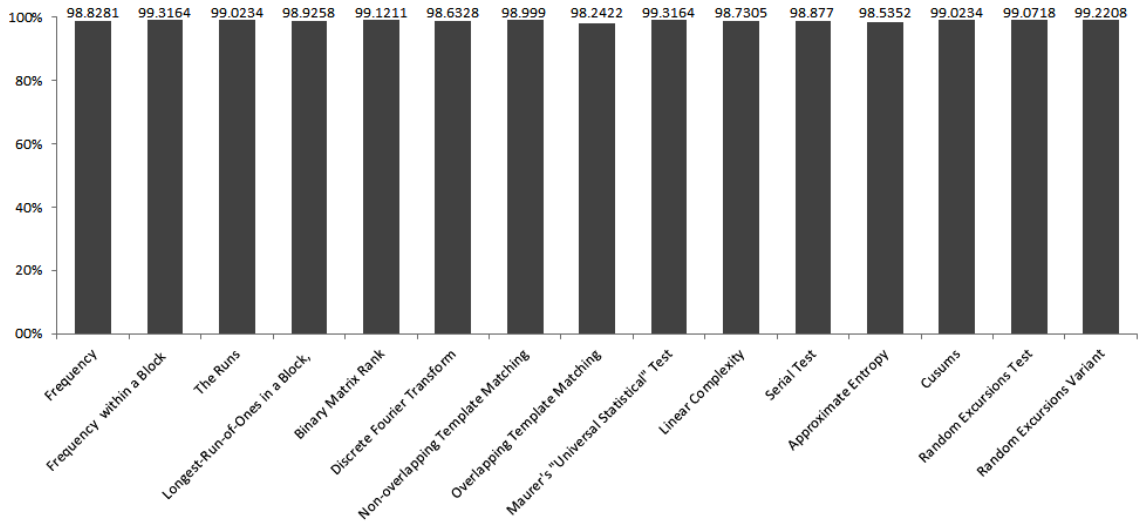

Fig. 13. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del $GBPA_2$, que aprobaron cada prueba estadística.

En base a estos resultados, se tiene un porcentaje global de aprobación del 98.9243.

Para esta propuesta también se realizó una evaluación de la salida que se obtiene tras realizar la operación xor con la salida del $GBPA_2$, y los resultados fueron aprobatorios tal como se esperaban debido a lo que indica el Teorema 1.

El mensaje que se utilizó para realizar el cifrado se define como $m = (m_0, m_1, m_2, m_3)$ donde $m = \{0,1\}^{128}$, y los valores particulares de inicialización para cada elemento son $m_n = (8000000)$. Después de cada cifrado, se decremento en uno el valor de cada elemento de m.

```
llave = 0
nonce = 0
contador = 0
desde i = 1 hasta 200
     desde j = 1 hasta 200
          desde k = 1 hasta 200
               ACL(llave, nonce, contador)
               nonce = nonce + 1
               m = m - 1
          fin
          contador = contador + 1
          llave = llave + 1
     fin
fin
```

Fig. 14. Pseudo código para la generación de la secuencia pseudo aleatoria.

Prueba	Porcentaje
Frequency	98.5352
Frequency within a Block	99.1211
The Runs	99.2188
Longest-Run-of-Ones in a Block,	98.9258
Binary Matrix Rank	99.1211
Discrete Fourier Transform	99.2186
Non-overlapping Template Matching	99.0597
Overlapping Template Matching	99.0234
Maurer's "Universal Statistical" Test	99.0234
Linear Complexity	99.0234
Serial Test	98.6816
Approximate Entropy	98.7305
Cusums	98.7305
Random Excursions Test	99.1588
Random Excursions Variant	99.2871

Tabla 9. Porcentaje de secuencias binarias, después del cifrado por medio de *ACL*, que aprobaron cada prueba estadística.

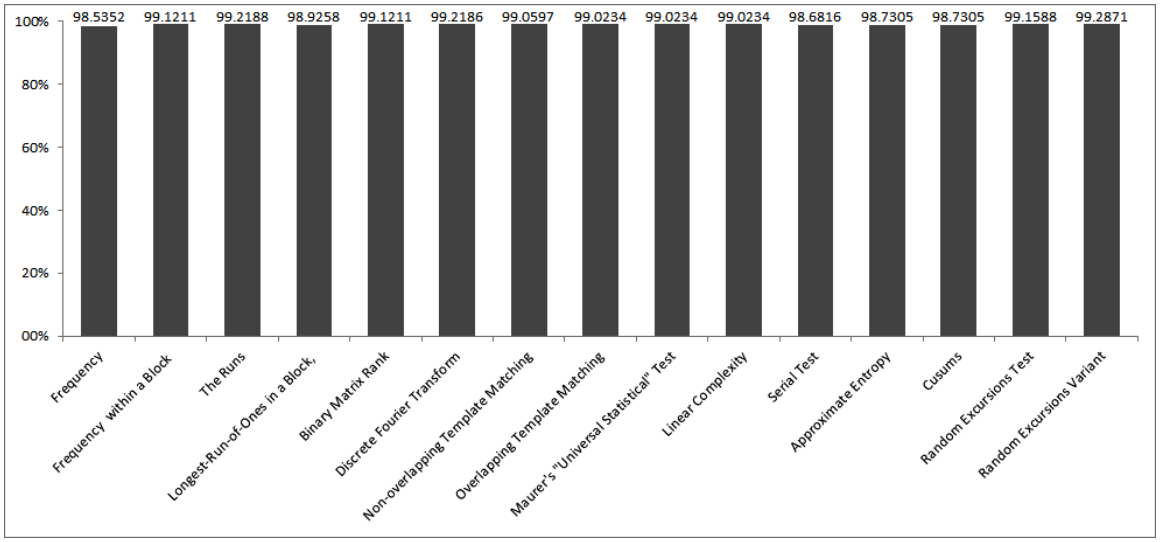

Fig. 15. Gráfica del porcentaje de secuencias binarias, después del cifrado por medio de *ACL*, que aprobaron cada prueba estadística.

El porcentaje de aprobación global es de 98.9906.

En base a los resultados de las pruebas estadísticas, las cuales fallaron en demostrar que la secuencia binaria no es aleatoria, se puede concluir que el algoritmo propuesto es capaz de generar secuencias que presentan características similares a las secuencias verdaderamente aleatorias, de forma que son indistinguibles de ellas. Sabiendo esto, y de acuerdo al Teorema 1, se concluye que el algoritmo de cifrado *ACL* presenta seguridad en base a que la salida cifrada que provee es indistinguible de una secuencia aleatoria.

5.4 REQUERIMIENTOS DE CÓMPUTO

Debido a que este trabajo de tesis está enfocado en proveer un algoritmo de cifrado a sistemas con bajo poder de cómputo, se presentan los requerimientos de memoria y la cantidad de ciclos de CPU utilizados para cifrar un mensaje por medio de *Salsa20 Modificado* y de *ACL*. Se presenta además una comparativa respecto a Salsa20.

En los tres análisis se utilizó el IDE, por sus siglas en inglés *Integrated Development Environment*, AVR Studio 4 de la compañía Atmel [34]. La implementación de los tres algoritmos se hizo para el microcontrolador Atmega644p [35] y se utilizó la optimización 3.

Respecto a ciclos de CPU necesarios para los algoritmos, se presenta la comparativa considerando el cifrado de un mensaje de 512 bits debido a que es el tamaño de salida de Salsa20, se presentan también los ciclos requeridos para un cifrado de 128 bits que es el tamaño de salida de ambas propuestas.

Salsa20

Memoria:	**Ciclos del CPU**:
9,610 bytes de memoria de programa	181,351 para cifrar 512 bits
12,362 bytes de memoria de datos	

Tabla 10. Requerimientos de Salsa20 de memoria de programa y de datos y de ciclos de CPU para cifrar un mensaje de 512 bits.

Salsa20 Modificado

Memoria:	**Ciclos del CPU**:
2,218 bytes de memoria de programa	13,185 para cifrar 128 bits
16 bytes de memoria de datos	52,740 para 512 bits

Tabla 11. Requerimientos de *Salsa20 Modificado* de memoria de programa y de datos y de ciclos de CPU para cifrar un mensaje de 128 y 512 bits.

ACL

Memoria:	**Ciclos del CPU**:
1,840 bytes de memoria de programa	5,614 para cifrar 128 bits
16 bytes de memoria de datos	22,456 para 512 bits

Tabla 12. Requerimientos de *ACL* de memoria de programa y de datos, y de ciclos de CPU para cifrar un mensaje de 128 y 512 bits.

En la Fig. 16 se presenta una gráfica comparativa de los requerimientos de memoria de los tres algoritmos, y en la Fig. 17 los requerimientos de ciclos de CPU para el cifrado de 512 bits.

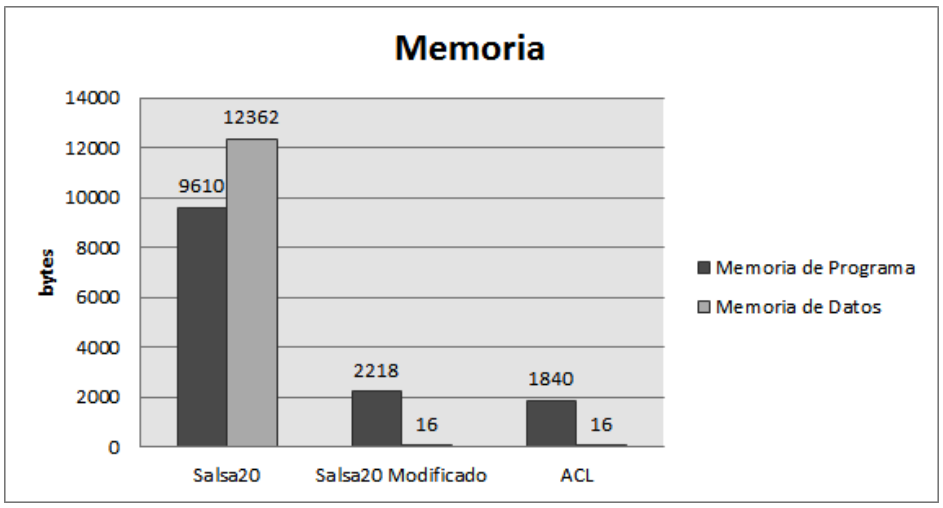

Fig. 16. Gráfica con una comparativa entre la cantidad de memoria de programa y de datos requerida por los tres algoritmos: Salsa20, *Salsa20 Modificado* y *ACL*.

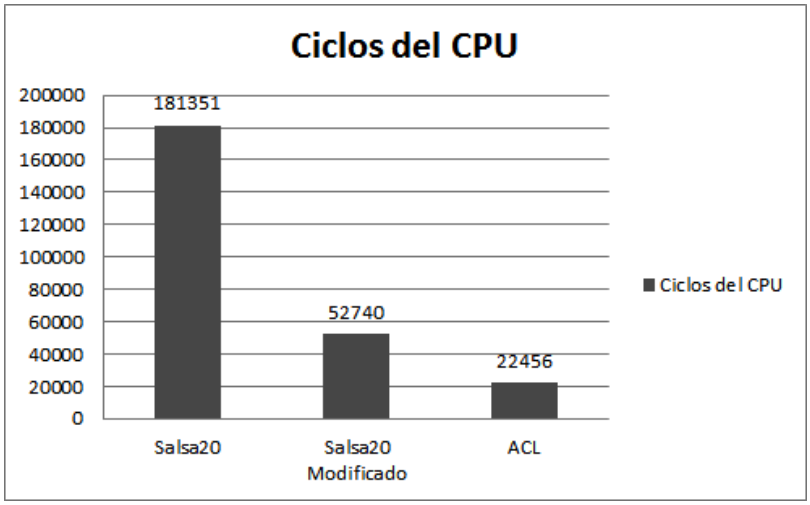

Fig. 17. Gráfica con una comparativa entre la cantidad de ciclos de CPU requeridos por los tres algoritmos: Salsa20, *Salsa20 Modificado* y *ACL* para cifrar un mensaje de 512 bits.

El algoritmo *Salsa20 Modificado* requiere un 76.9199% menos de memoria de programa y un 99.8706% menos de memoria de datos que el algoritmo Salsa20. Y utiliza un 70.9183% menos de ciclos de CPU que Salsa20 para cifrar un mensaje de 512 bits.

El algoritmo *ACL* requiere un 17.0424% menos de memoria de programa que el algoritmo *Salsa20 Modificado*. En cuanto a memoria de datos, utilizan la misma cantidad de bytes. *ACL* utiliza un 57.4213% menos de ciclos de CPU que *Salsa20 Modificado* para cifrar un mensaje de 512 bits.

6. Generador de Números Verdaderamente Aleatorios

El objetivo de esta tesis es el diseño de un algoritmo de cifrado viable para un sistema embebido de bajo poder computacional. Tal como se describió en los antecedentes los algoritmos de cifrado utilizan una llave secreta, que en el caso de los algoritmos de flujo se le conoce también como semilla, la cual consiste en un TRN. En ese trabajo de tesis se quiso ir más allá del objetivo y experimentar en la generación de TRN llegando a una posible propuesta de TRNG. En este capítulo se describe el trabajo realizado y los resultados obtenidos.

A diferencia de la propuesta que se describió en la sección 2.17 *Circuitos Caóticos en RNGS* donde se requiere de otros componentes electrónicos además de los necesarios para la implementación del circuito caótico, en esta propuesta se buscó que la generación de TRN se realizará directamente con los datos que provienen el circuito caótico y el único otro requisito fuera el software de tratamiento de estos.

El circuito caótico que se utilizó fue el circuito de Chua que se muestra en la Fig. 18 [36].

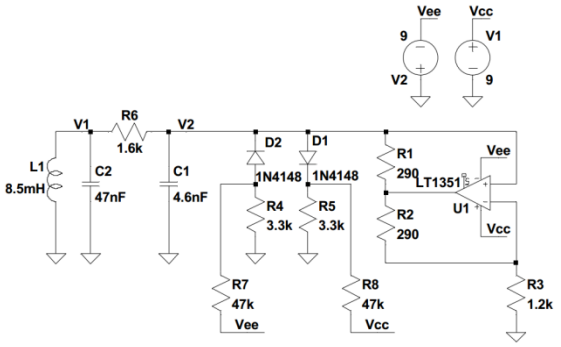

Fig. 18. Circuito de Chua usado en la propuesta de TRNG.

Debido a que no se contó con todos los componentes para poder realizar una implementación física del circuito, este trabajo se realizó utilizando la herramienta de simulación de circuitos LTSpice [37], la cual permite conocer los valores de voltaje y corriente de los componentes conforme al tiempo.

La salida del circuito obtenida en la simulación se muestra en la Fig. 19. La salida es el graficado del voltaje del nodo V1 contra el voltaje en el nodo V2.

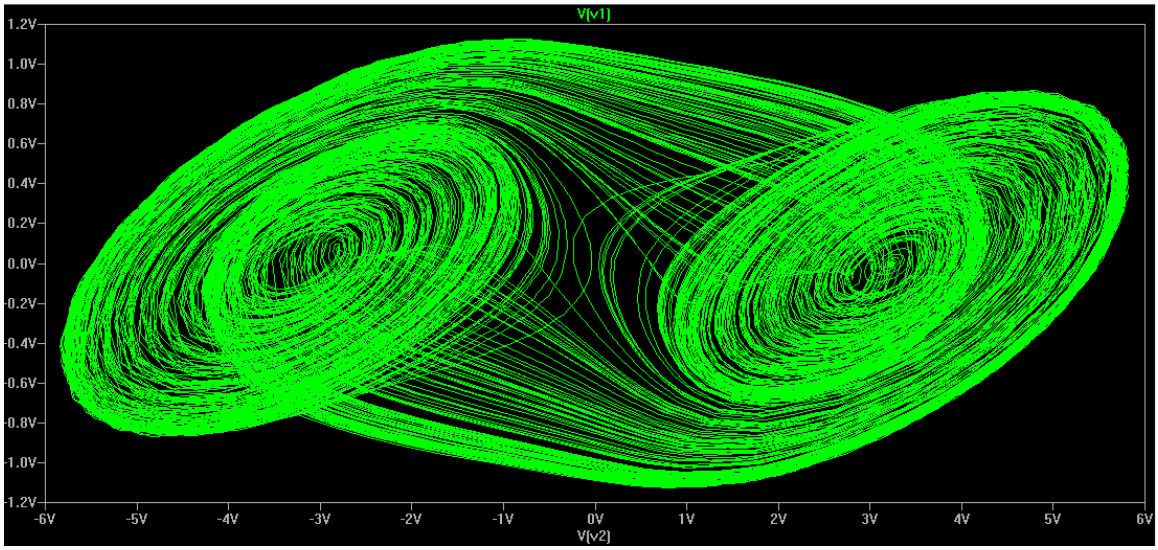

Fig. 19. Grafica de los nodos V1 y V2 del Circuito de Chua.

La propuesta que se presenta utiliza únicamente el voltaje del nodo V1, y consiste en solo dos pasos para obtener los TRNs:

1. Escalar la parte decimal de la medición del voltaje en V1, y posteriormente
2. Tomar los 6 bits menos significativos del dato obtenido en el paso 1

Los seis bits obtenidos en el paso 2 son la salida de este TRNG.

Por medio de la herramienta LTSpice se hizo la simulación del circuito por 50 segundos obteniendo los valores del voltaje en V1, sobre esos datos se realizó el procesamiento descrito anteriormente y se obtuvieron 46 secuencias de 1 millón de bits cada una. No se pudieron obtener más secuencias binarias ya que al solicitarle a la herramienta simulaciones de mayor tiempo, ésta dejaba de responder.

Se realizaron las pruebas estadísticas sobre estas 46 secuencias y la mayoría de ellas aprobó. En la siguiente tabla se presenta el porcentaje de secuencias que aprobaron cada prueba estadística.

Prueba	Porcentaje
Frequency	100
Frequency within a Block	100
The Runs	100
Longest-Run-of-Ones in a Block,	100
Binary Matrix Rank	97.8261
Discrete Fourier Transform	97.8261
Non-overlapping Template Matching	99.0452
Overlapping Template Matching	100
Maurer's "Universal Statistical" Test	100
Linear Complexity	100
Serial Test	96.7391
Approximate Entropy	95.6522
Cusums	100
Random Excursions Test	97.9167
Random Excursions Variant	99.8148

Tabla 13. Porcentaje de secuencias binarias, obtenidas por medio del *TRNG* del circuito de Chua, que aprobaron cada prueba estadística.

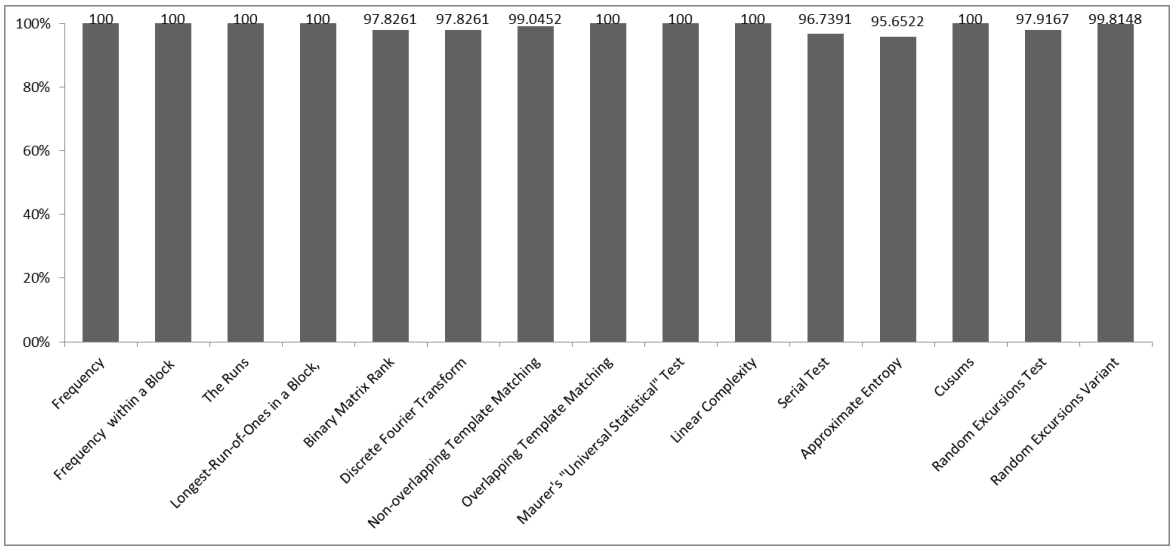

Fig. 20. Gráfica del porcentaje de secuencias binarias, obtenidas por medio del *TRNG* del circuito de Chua, que aprobaron cada prueba estadística.

En base a estos resultados, se tiene un porcentaje global de aprobación del 98.9880.

Debido a que no se pudo contar con el circuito físico y se realizaron las pruebas estadísticas con relativamente pocos datos obtenidos de una simulación, no se puede afirmar que este TRNG sea seguro para cualquier tipo de aplicación. Sin embargo al haber obtenido buenos resultados en las pruebas que fue posible realizar, se concluye que es una propuesta viable y quedando para el autor como trabajo futuro.

7. Conclusiones y trabajo futuro

Se concluye que es posible proveer seguridad a los datos que son enviados a través de un sistema embebido de bajo poder computacional por medio de un esquema de cifrado que fue diseñado considerando las características de este tipo de sistemas y de los datos que van a ser transmitidos por ellos.

En este documento se presentó la descripción de dos esquemas de cifrado de flujo que fueron diseñado considerando la capacidad de almacenamiento y de procesamiento de los sistemas embebidos de bajo poder de cómputo, y tomando en cuenta las características de los datos que son transmitidos a través de estos sistemas, donde en las redes de sensores los paquetes de datos suelen ser de corta longitud. Se sometió cada algoritmo a un conjunto de pruebas estadísticas y los resultados fueron aprobatorios.

Respecto al TRNG la idea que se quiso llevar a cabo fue la generación de los TRN con la menor cantidad de elementos a adicionar a los sistemas embebidos, o más específicamente a las redes de sensores. El TRNG descrito en el capítulo 6 sigue esta idea ya que para su implementación se requieren únicamente los componentes del circuito de Chua y el software para la obtención y tratamiento de sus valores de voltaje.

Como se explicó en el capítulo 6, la propuesta de TRNG no está enteramente terminada y como trabajo futuro lo que se pretende es ya sea el lograr obtener los componentes del circuito o trabajar con algún otro diseño del circuito de Chua que podamos implementar físicamente para de esta manera trabajar y realizar pruebas con datos reales y a su vez no tener la limitación de la cantidad de datos que se pueden obtener.

Referencias

[1] H. M. Elbehiery and M. Ghada Abdelmouez, "Implementation of new symmetric ciphering on ATMEGA 32," in *Computer Engineering Conference (ICENCO), 2010 International*, 2010, pp. 1–8.

[2] X. Luo, K. Zheng, Y. Pan, and Z. Wu, "Encryption algorithms comparisons for wireless networked sensors," in *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004, vol. 2, pp. 1142–1146 vol.2.

[3] O. Krejcar, "Using of Ubiquitous Computing principles to develop a mobile user adaptive system framework," in *Roedunet International Conference (RoEduNet), 2010 9th*, 2010, pp. 352–357.

[4] C. O'Driscoll, "Privacy in context: Privacy issues in Ubiquitous Computing applications," in *Third International Conference on Digital Information Management, 2008. ICDIM 2008*, 2008, pp. 827–837.

[5] I. Vajda and L. Buttyán, "Lightweight authentication protocols for low-cost RFID tags," in *Second Workshop on Security in Ubiquitous Computing–Ubicomp 2003*, 2003.

[6] M. Weiser, "The computer for the 21st Century," *IEEE Pervasive Comput.*, vol. 1, no. 1, pp. 19–25, 2002.

[7] S. E. Sarma, S. A. Weis, and D. Engels, "Radio-frequency-identification security risks and challenges," *Cryptobytes*, vol. 6, no. 1, 2003.

[8] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 9, no. 4, pp. 545–557, 2001.

[9] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Science & Business Media, 2010.

[10] Shibu, *Intro To Embedded Systems 1E*. Tata McGraw-Hill Education.

[11] I. A. D. V. S. Bagad, *Cryptography And Network Security*. Technical Publications, 2008.

[12] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[13] C. Paar and J. Pelzl, *Understanding Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[14] G. J. Chaitin, *Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory*. World Scientific, 1990.

[15] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo, "A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 403–409, Apr. 2003.

[16] D. Knuth, *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*. Addison–Wesley, 1981.

[17] B. Schneier, *Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C*. Wiley Computer Publishing, John Wiley & Sons, Inc, 1996.

[18] M. Bucci, “Random Bit Generator,” in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Springer US, 2011, pp. 1022–1026.
[19] F. Koeune, “Pseudorandom Number Generator,” in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Springer US, 2011, pp. 995–996.
[20] R. L. Graham, *Handbook of Combinatorics*. Elsevier Science, 1995.
[21] M. Robshaw, “The eSTREAM Project,” in *New Stream Cipher Designs*, M. Robshaw and O. Billet, Eds. Springer Berlin Heidelberg, 2008, pp. 1–6.
[22] ECRYPTII, *eSTREAM project*. ECRYPTII, 2008.
[23] D. J. Bernstein, “Salsa20 design,” Apr. 2005.
[24] D. J. Bernstein, “Salsa20 specification,” Apr. 2005.
[25] D. J. Bernstein, “Salsa20 security,” Apr. 2005.
[26] D. J. Bernstein, “Salsa20 speed,” Apr. 2005.
[27] S. JuanSoto, JamesNechvatal, Miles Smid, ElaineBarker, Stefan Leigh, MarkLevenson, Mark Vangel, DavidBanks, AlanHeckert, JamesDray AndrewRukhin, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST, 2010.
[28] T. S. Parker and L. O. Chua, “Chaos: A tutorial for engineers,” *Proc. IEEE*, vol. 75, no. 8, pp. 982–1008, Aug. 1987.
[29] L. Kocarev and S. Lian, Eds., *Chaos-Based Cryptography*, vol. 354. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
[30] M. P. Kennedy, “Three steps to chaos. I. Evolution,” *IEEE Trans. Circuits Syst. Fundam. Theory Appl.*, vol. 40, no. 10, pp. 640–656, Oct. 1993.
[31] V. Tavas, A. S. Demirkol, S. Ozoguz, S. Kilinc, A. Toker, and A. Zeki, “An IC random number generator based on chaos,” in *2010 International Conference on Applied Electronics (AE)*, 2010, pp. 1–4.
[32] F. Koeune, “Timing Attack,” in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Springer US, 2011, pp. 1303–1305.
[33] C. E. Shannon, “Communication theory of secrecy systems,” *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.
[34] “Atmel Corporation - Microcontrollers, 32-bit, and touch solutions.” [Online]. Available: http://www.atmel.com/default.aspx. [Accessed: 23-Nov-2015].
[35] “ATmega644.” [Online]. Available: http://www.atmel.com/devices/ATMEGA644.aspx. [Accessed: 23-Nov-2015].
[36] juho-eric, *Chua’s circuit implementation*. 2012.
[37] L. Technology, *LTSpice IV Simulator*. 2015.