UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISEÑO ENSENADA

INFORMACIÓN MÉDICA EN ESQUEMA MULTIUSUARIO

TESIS

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

INGENIERO EN ELECTRÓNICA

presenta:

ALEXIS CRESPO MICHEL

Ensenada, Baja California, México, Septiembre de 2018.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISEÑO ENSENADA

INFORMACIÓN MÉDICA EN ESQUEMA MULTIUSUARIO

TESIS

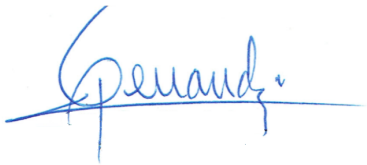
Que para obtener el grado de Ingeniero en Electrónica presenta:

ALEXIS CRESPO MICHEL

Aprobada por el siguiente comité:

Dra. Rosa Martha López Gutiérrez
Codirector del comité

Dr. Miguel Ángel Murillo Escobar
Codirector del comité



Dr. César Cruz Hernández
Miembro del comité

Dra. Liliana Cardoza Avedaño
Miembro del comité

M.C José Antonio Michel Macarty
Miembro del comité

Ensenada, Baja California, México. Septiembre de 2018.

RESUMEN de la tesis de **Alexis Crespo Michel**, presentada como requerimiento para obtener el grado de INGENIERO en ELECTRÓNICA, del programa de Licenciatura de la Universidad Autónoma de Baja California. Ensenada, Baja California, México, Septiembre de 2018.

INFORMACIÓN MÉDICA EN ESQUEMA MULTIUSUARIO

Resumen aprobado por:

Dra. Rosa Martha López Gutiérrez
Codirector de tesis

En este trabajo de tesis de licenciatura, se propone un esquema de transmisión segura de datos médicos a través de redes inalámbricas basado en caos y se implementa cada bloque funcional en sistemas embebidos de bajo costo.

Se realiza la implementación de tres mapas caóticos en un microcontrolador de 32 bits para analizar características de las secuencias caóticas generadas, como el tiempo de procesamiento y sensibilidad a condiciones iniciales para sistemas de tiempo discreto. También, se comparan las secuencias generadas en el microcontrolador con secuencias generadas en Matlab bajo las mismas condiciones iniciales para observar el desempeño de un sistema de bajo costo en comparación con un software dedicado de análisis numérico.

Finalmente se realizan pruebas de cifrado caotico utilizando el mapa logistico y se realiza la transmisión de archivos médicos de distintos tipos a través de una red local y se observan características de desempeño del esquema. También, se comprueba que los algoritmos de encriptamiento y desencriptamiento basados en cifrado de flujo funcionan correctamente para la aplicación.

Palabras clave: caos, sistema embebido, cifrado caótico, mapa logístico, IoT.

Abstract of the thesis presented by **Alexis Crespo Michel**, as a requirement to obtain the degree in ELECTRONICS ENGINEER, of the program of the Autonomous University of Baja California. Ensenada, Baja California, Mexico, September 2018.

MULTI-USER MEDICAL INFORMATION SCHEME

Abstract approved by:

Dra. Rosa Martha López Gutiérrez
Thesis co-director

In this thesis paper, a scheme for the secure transmission of medical data over wireless networks based on chaos is proposed and each functional block is implemented in low-cost embedded systems.

Three chaotic maps are implemented in a 32-bit microcontroller to analyze characteristics of the generated chaotic sequences, such as processing time and sensitivity to initial conditions for discrete time systems. Also, the sequences generated in the microcontroller are compared to sequences generated in Matlab under the same initial conditions to observe the performance of a low-cost system compared to dedicated numerical analysis software.

Finally, chaotic encryption tests are performed using the logistic map and the transmission of medical files of different types is performed through a local network and performance characteristics of the scheme are observed. Also, encryption and decryption algorithms based on flow encryption are verified to work correctly for the application.

Keywords: chaos, embedded system, chaotic encryption, logistic map, IoT.

A mi familia

Agradecimientos

A mis padres, Araceli y José. Por brindarme su amor incondicional e inculcarme valores que me permitan ser un hombre de bien. También, por haberme brindado su completo apoyo en el ámbito académico, felicitándome en mis logros y ayudándome en mis días malos, permitiéndome llevar una vida plena y feliz.

A mi hermano, Daniel por todo su cariño y apoyo. Por ser una persona que siempre se preocupa por mi y creer en que puedo llegar a hacer grandes cosas.

A la Dra. Rosa Martha López Gutiérrez, como coordinadora por todo su cariño y apoyo desde el inicio de mi carrera hasta ayudarme a avanzar al siguiente escalón. También, como directora de tesis permitiendo realizar este trabajo brindándome su ayuda durante el desarrollo de esta.

Al Dr. César Cruz Hernández, por haberme acercado por primera vez a las aplicaciones del caos lo cual me permitió desarrollar este trabajo de tesis.

A la Dr. Miguel Ángel Murillo Escobar, como codirector de tesis, por brindarme su experiencia y aconsejarme debidamente, logrando así mejorar la calidad de este trabajo y permitiéndome adquirir habilidades útiles para la siguiente etapa de mi vida.

A mi comité de tesis, por sus consejos y comentarios para la mejora de este trabajo.

A mis amigos, Estefanía, Daniel, Oscar, Eduardo, Carlos, José y David. Por brindarme su amistad y apoyo, lo cual hizo de mi carrera universitaria una experiencia mas amena.

A la Universidad Autónoma de Baja California, por brindarme un espacio íntegro donde realizarme profesionalmente. En especial a la Facultad de Ingeniería, Arquitectura y Diseño Ensenada (FIAD) por ser mi casa de estudios los últimos 4 años y brindarme espacios donde trabajar.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo económico brindado para la realización de este trabajo a través del Proyecto de Grupos de Investigación en Ciencia Básica, Referencia 166654.

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

Alexis Crespo Michel

Tabla de Contenido

Lista de Figuras

Capítulo 1

Marco teórico

1.1. Introducción

Recientemente, con el rápido avance en la instrumentación y las comunicaciones inalámbricas ha sido posible el desarrollo de nuevas técnicas que buscan facilitar el acceso a un *cuidado de la salud* de calidad comúnmente llamado *e - healthcare*. En lugar de requerir de una interacción física con el paciente, se pueden utilizar sistemas que permitan la *transmisión de datos biológicos* del paciente, a un servidor donde la información sea analizada y posteriormente almacenada en una *base de datos*. Esta información medica puede ser compartida entre varios usuarios como médicos especialistas, investigadores y agencias de seguros. Estas aplicaciones son principalmente deseables en países con *poca infraestructura médica y falta de personal bien entrenado*.

Toda la información relacionada con el paciente juega un papel importante en el diagnóstico y tratamiento médico, por lo cual es esencial proporcionar *seguridad y privacidad* a esta información, debido a que generalmente, todos estos datos sensibles se transmiten a través de *medios o canales de comunicaciones inseguros* (como Internet, redes locales de computo, etc.). Es importante remarcar que por seguridad, se refiere a la *protección del proceso de transferencia y almacenamiento de la información*, mientras que la privacidad de la información esta relacionada con los medios que proveen *autorización* al usuario para la obtención y visualización de los datos médicos [1].

Una *protección robusta* de la información debe tomar en cuenta primeramente, la *seguridad* del lado de la base de datos (servidor), donde se busca proteger la información de la modificación maliciosa, ya que esto podría llevar a un *diagnóstico o tratamiento erróneos*. También, se tiene que la información del paciente debe presentar acceso limitado a usuarios autorizados, para proteger al paciente contra *abusos de privacidad*. Una forma de asegurar este punto es mediante *métodos de encriptación* que permitan proteger la información, forzando la utilización de alguna *identificación criptográfica* (como una clave).

Para proporcionar seguridad y proveer un mecanismo de privacidad de la información, se debe tomar en cuenta los *requerimientos específicos* de la aplicación para ofrecer

un *balance* entre la seguridad, eficiencia y practicidad del sistema.

1.2. Descripción del problema

La implementación de sistemas que faciliten la *interacción paciente - médico* de manera remota a generado algunas dudas respecto a la seguridad y privacidad de la información del usuario, ya que, toda la información que viaje a través de un canal público es *susceptible* a ser interceptada por un tercero.

Otro problema es la *violación de la seguridad de datos* conocido como *("data breach")*; una vez que un tercero accede a información médica almacenada en una base de datos sin autorización, este puede copiarla para un posterior uso malicioso o incluso modificarla.

Según datos obtenidos por el *Departamento de Salud y Servicios Humanos de Estados Unidos (HSS)*, entidades y negocios asociados cubiertos por *La Ley de Transferencia y Responsabilidad de Seguro Médico (HIPPA)* reportaron en 2018 un *incremento del 19 % en violación de datos* respecto al mes anterior tal como se muestra en la figura 1.1 [2].

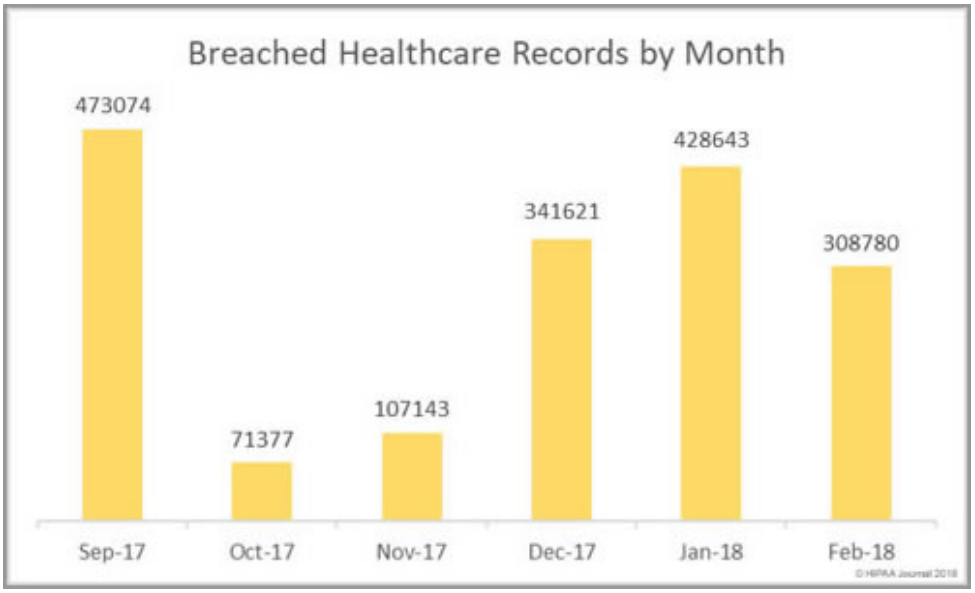


Figura 1.1: Números de historiales médicos accedidos en *violaciones de seguridad* recientes [2].

Según el sitio *hipaajournal.com*, la mayor causa de los "data breaches" que ocurrieron en febrero del 2018, fueron debido a ataques con un *acceso sin autorización*, seguido por *hacking* como la segunda causa tal como se puede observar en la figura 1.2 [2], en relación con la figura 1.1, se puede observar que con cada ataque se logró comprometer una *cantidad significativa* de información, lo cual implica que un solo descuido en la

infraestructura de los sistemas de almacenamiento de datos puede perjudicar a un gran numero de personas.

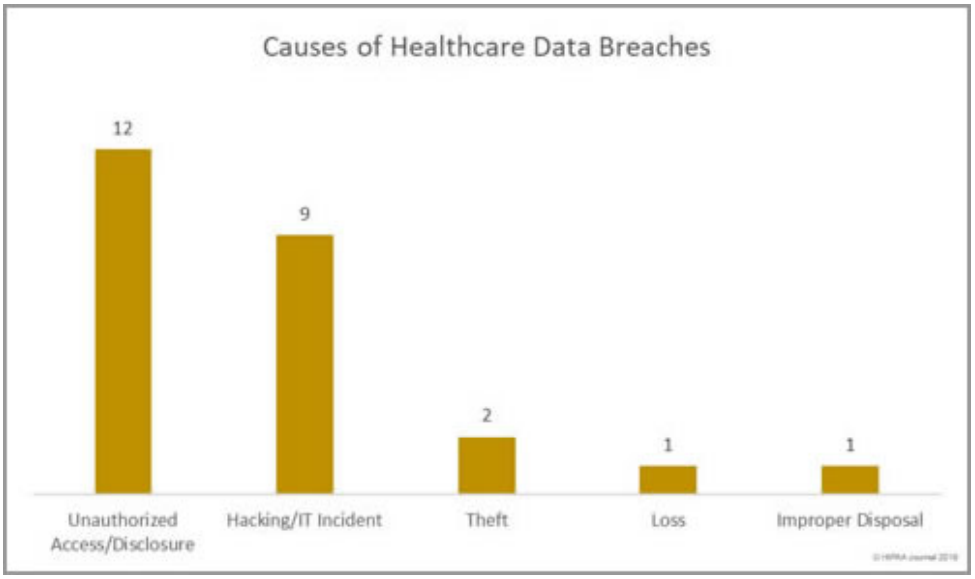


Figura 1.2: Causas de *violación de seguridad de datos* en *febrero del 2018* [2].

Las cifras de la figura 1.2 coinciden con un reporte presentado por la compañía de seguros *Beazley* en 2017, donde presentaban que el *41 % de las violaciones de seguridad de datos ocurrían por accesos sin autorización* [3]. Esto tiene sentido tomando en cuenta la *infraestructura existente* en centros de salud e instituciones relacionadas, donde la mayoría de los dispositivos médicos utilizados son *sistemas embebidos* y que debido en la manera en la que fueron desarrollados por el fabricante, es difícil o incluso imposible actualizar el sistema para parchar posibles *huecos de seguridad*.

1.3. Objetivos

Con base a los problemas presentados anteriormente y tomando en cuenta el creciente interés en el resguardo de la información sensible con aplicaciones en la *telemedicina*, *el objetivo general* de esta tesis de licenciatura es el siguiente:

Implementar físicamente un *esquema de transmisión de datos médicos* de forma segura *de bajo costo*, utilizando un protocolo de red inalámbrico.

Para cumplir con el objetivo presentado, se plantean lo siguientes *obejtivos especificos*:

1. Determinar y caracterizar hardware de bajo costo que presente características suficientes para la aplicación.

2. Implementar varios mapas caóticos en el hardware seleccionado para determinar el mapa adecuado tomando en cuenta las limitaciones de procesamiento.

3. Diseñar un algoritmo de cifrado caótico para archivos de información medica (texto, imagen, vídeo).

4. Implementacion de servidor local para el almacenamiento de informacion encriptada.

1.4. Solución propuesto

A fin de solucionar el problema de *seguridad y privacidad* de la información medica indicado anteriormente, se propuso el siguiente esquema general:

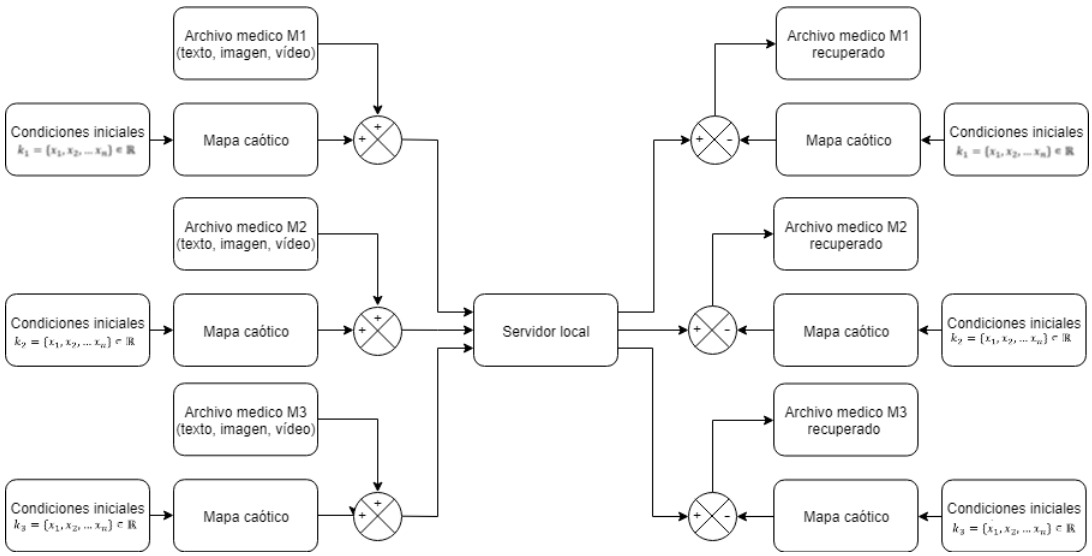


Figura 1.3: Esquema general propuesto para la *transmisión segura* de datos médicos.

El esquema de transmisión de datos de forma segura esta basado en el *encriptamiento caótico aditivo*, donde la información es sumada a una secuencia caótica generada por condiciones iniciales dadas.

Los pasos para llevar a cabo una transmisión segura se describen a continuación:

1. El transmisor y el receptor comparten de *manera segura* una clave, la cual está basada en las condiciones iniciales y parámetros críticos elegidos para el mapa caótico correspondiente.

2. Se lee un archivo con información médica (texto, imagen, vídeo) guardado anteriormente en un dispositivo de almacenamiento compatible con el sistema.

3. Se genera una secuencia caótica basada en el mapa caótico y la clave elegida en el primer punto, la longitud de la secuencia caótica es función del tamaño del archivo, así como del *algoritmo* seleccionado para la encriptación del mismo.

4. Se realiza una *suma aritmética o binaria* dependiendo del algoritmo propuesto para la encriptación del archivo.

5. Se realiza la transmisión del archivo encriptado a un servidor, utilizando un *protocolo de comunicaciones inalámbricas* donde la información es almacenada para su uso posterior.

Los pasos para llevar a cabo una recepción segura se describen a continuación:

1. Se toma en cuenta la *clave compartida de manera segura* entre el transmisor y el receptor.

2. Mediante un *protocolo de comunicaciones inalámbricas*, se obtiene del servidor el archivo encriptado requerido y se almacena de manera temporal en el dispositivo receptor.

3. Se genera una secuencia caótica basada en el mapa caótico y la clave elegida en el punto 1, la longitud de la secuencia caótica es función del tamaño del archivo, así como del *algoritmo utilizado* para la encriptación del mismo.

4. Se realiza una *resta aritmética o binaria* dependiendo del algoritmo utilizado para la encriptación del archivo.

5. Se almacena el archivo con información medica desencriptado en un *dispositivo de almacenamiento compatible* con el sistema para su posterior análisis por parte del usuario receptor.

Con base a las características que presenta el esquema propuesto se pretende atacar los problemas de seguridad y privacidad presentados anteriormente, de una manera *directa e indirecta*.

- El problema de la *privacidad* de la información va a ser atacado directamente encriptando la información, haciendo que sea imposible la *visualización* de esta a usuarios *sin autorización*.

- El problema de la *seguridad* va a ser atacado indirectamente, ya que, el esquema como tal no previene el posible robo de la información almacenada en el servidor, sin embargo, resta valor a la información debido a que no puede ser visualizada fácilmente sin la clave, de manera que eso *desincentiva a los atacantes*.

Capítulo 2

Descripción del Hardware

En este capitulo se presenta una breve descripción de los *sistemas embebidos* y sus aplicaciones mas comunes, también se describe a detalle el Hardware utilizado para la *implementación del esquema* propuesto en el capitulo 1, y se explica brevemente el motivo de su elección y la relación que tiene con respecto a los *objetivos particulares* de diseño también indicadas en el capitulo anterior.

2.1. Sistema embebido

Un *sistema embebido* es un sistema electrónico diseñado para realizar *tareas especificas* generalmente en tiempo real (ver figura 2.1), con aplicaciones tanto en la vida diaria como a nivel industrial. En un sistema embebido la mayoría de los componentes se encuentran *integrados* a la placa base lo cual asegura la funcionalidad entre sus componentes para una tarea especifica logrando generalmente ventajas como un coste reducido, un tamaño pequeño y un consumo de energía menor con respecto a un sistema que presenta características de proposito general [4].

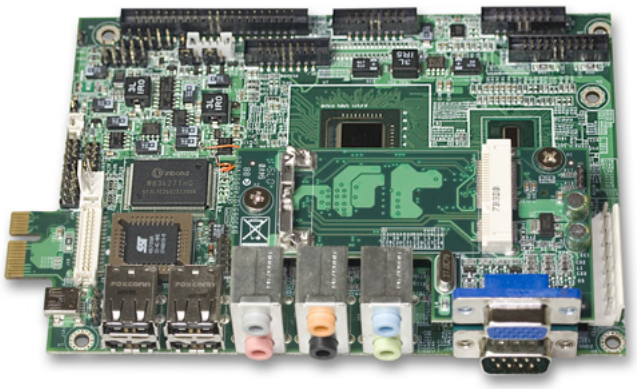


Figura 2.1: Ejemplo de un *sistema embebido*.

La mayoría de los sistemas embebidos están basados frecuentemente en *microcontroladores*, es decir, CPU con memoria integrada e interfaces de periféricos, sin embargo,

en sistemas mas complejos también esta presente el uso de *microprocesadores* utilizando electrónica externa para su funcionamiento, en la figura 2.2 se puede apreciar un diagrama que describe las características físicas encontradas comúnmente en sistemas embebidos.

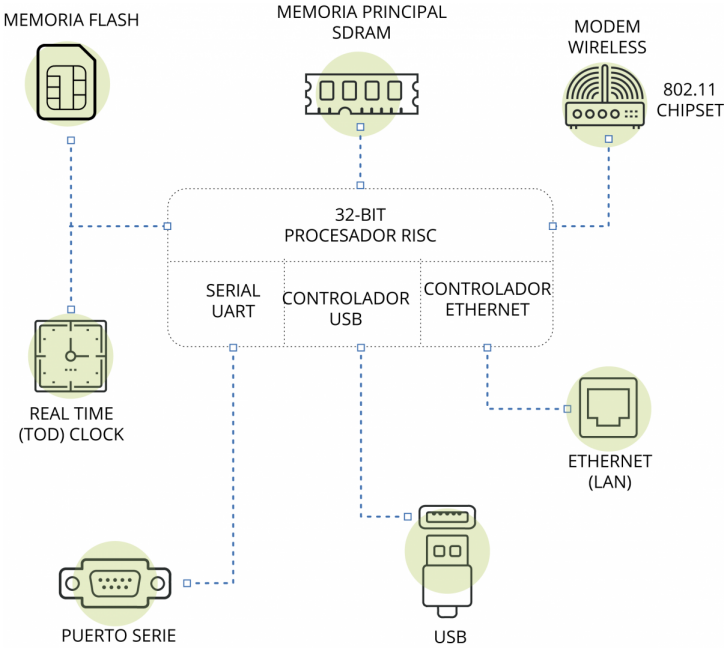


Figura 2.2: Descripción de un sistema embebido (nivel físico) [5].

Por su naturaleza, el hecho de que los sistemas embebidos sean dedicados a tareas especificas, permite que los ingenieros de diseño puedan *optimizar* el sistema para reducir costos y tamaño del producto incrementando incluso su confiabilidad y desempeño.

2.1.1. Aplicaciones de los sistemas embebidos

Los sistemas embebidos pueden encontrarse comúnmente en aplicaciones de consumo, industriales, automotriz, medicas y militares tal como se puede observar en la figura 2.3 [5].

La *electrónica de consumo* incluye la telefónica móvil, cámaras digitales, impresoras y recibidores GPS. También aplicaciones en electrodomésticos como hornos de microondas, lavadoras, hornos eléctricos.

Los *sistemas de transporte* desde aviones hasta automoviles han visto un incremento en el uso de sistemas embebidos en los ultimos años. Nuevos aviones que contienen sistemas avanzados de avionica como sistemas de dirección inercial y recibidores GPS. En los automoviles los sistemas embebidos están generalmente asociados a sistemas de asistencia de control y caracteristicas de seguridad como sistemas de freno antibloqueo

Figura 2.3: Aplicaciones de *consumo comunes* de los sistemas embebidos.

(ABS), control de traccion (TCS) y control de estabilidad.

En el *equipo medico*, los sistemas embebidos se utilizan principalmente para monitoreo de signos vitales, estetoscopios electrónicos para amplificación de sonidos del corazón, y varios sistemas de procesamiento de imágenes. El avance en la capacidad de procesamiento de estos dispositivos ha permitido incluso la transmisión en tiempo real de información medica surgiendo así el concepto de telemedicina.

2.2. System on Chip ESP8266EX

El *ESP8266EX* es un SoC del fabricante chino *Espressif Systems* que ofrece una solución Wi-Fi de bajo costo el cual esta integrado por un *stack TCP/IP* y un microcontrolador, el conjunto ofrece un desempeño suficiente para la mayoría de las aplicaciones IoT más comúnmente utilizadas.

Fue pensado originalmente como un *adaptador UART a Wi-Fi* que permitía a otros microcontroladores conectarse a una red inalambrica Wi-Fi por medio de comandos *estilo Hayes* [9].

A pesar de la falta de documentación inicial, se formo una gran comunidad alrededor de este chip, lo que permitió añadir el desarrollo de firmware para aprovechar la capacidad del SoC [6], esto lo convirtió en un integrado realmente versátil, permitiendo que pueda funcionar como un *dispositivo IoT independiente* o como un *esclavo* a otro microcontrolador de manera que le añada opciones de conectividad.

2.2.1. Especificaciones del SoC

El ESP8266EX cuenta con el conjunto de especificaciones que se puede observar en la figura 2.4, aunque el SoC es fabricado por Espressif Systems, los módulos que llevan el chip están desarrollados por diferentes fabricantes y en una variedad de tamaños y opciones de periféricos, de manera que algunas de las siguientes características pueden no estar presentes en todos los módulos/tarjetas de desarrollo.

Categories	Items	Parameters
Wi-Fi	Certification	Wi-Fi Alliance
	Protocols	802.11 b/g/n
	Frequency Range	2.4G ~ 2.5G (2400M ~ 2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: –91 dbm (11 Mbps)
		802.11 g: –75 dbm (54 Mbps)
		802.11 n: –72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit processor
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM/LED Light & Button
	Operating Voltage	2.5V ~ 3.6V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	–40°C ~ 125°C
	Storage Temperature Range	–40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
	External Interface	-
Software	Wi-Fi Mode	Station/SoftAP/SoftAP+Station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

Figura 2.4: Especificaciones del SoC ESP8266EX [8].

2.3. Modulos ESP8266

En los últimos años, el constante crecimiento en el desarrollo de aplicaciones IoT, generó interés en el ESP8266EX, y es que, su bajo costo aunado con el hecho de que no es necesario utilizar gran cantidad de componentes externos permitió que se crearan distintas *tarjetas de desarrollo* alrededor de este SoC [7], en la figura 2.5 se pueden observar todas las variantes disponibles actualmente en el mercado.

Figura 2.5: Variantes de *módulos basados* en el SoC ESP8266EX [7].

La primera serie de módulos basados en el ESP8266EX fueron desarrollados por Ai-Thinker, el fabricante de terceros que cuenta con los módulos más ampliamente utilizados actualmente.

En general, cualquier módulo ESP8266 cuenta con las siguientes características:

- *CPU base*: LX106 de la compañía Xtensa a 80 MHZ y con arquitectura RISC de 32 bit.
- 64 KB de *memoria RAM* para instrucciones, más 96 KB de *RAM para datos*.
- *Memoria flash* desde 512 KB hasta 16 MB.
- Tecnología Wi-Fi según la norma *IEEE 802.11 b/g/n Wi-Fi*.
- Hasta 16 *pines de propósito general* (GPIO) y convertidor analógico digital.
- Periféricos de comunicación *SPI, I^2C y UART*.

En la figura 2.6 se pueden observar una tabla con las *caracteristicas fisicas* de algunas variantes de módulos basados en ESP8266.

Versión	Pines	Separación	Antena	Dimensiones [mm]	Notas
ESP-01	GPIO0/2/16	0.1 [in]	Grabada en PCB	14.3 × 24.8	Este es el modulo más común, existen múltiples variantes.
ESP-02	GPIO0/2/15	0.1 [in]	Conector U-FL	14.2 × 14.2	Existen múltiples variantes.
ESP-03	GPIO0/2/12 /13/14/15/16	2 [mm]	Cerámica	17.3 × 12.1	Este es el modulo más popular y utilizado en Internet.
ESP-04	GPIO0/2/12 /13/14/15/16	2 [mm]	Ninguna	14.7 × 12.1	Tipo de antena personalizable por el usuario.
ESP-05	Ninguno	0.1 [in]	Conector U-FL	14.2 × 14.2	Existen múltiples variantes.
ESP-06	GPIO0/2/12 /13/14/15/16	Miscelánea	Ninguna	14.2 × 14.7	El escudo de metal dice contar con certificación FCC Ce.
ESP-07	GPIO0/2 /4/5/12/13 /14/15/16	2 [mm]	Ambas, Cerámica y Conector U-FL	20.0 × 16.0	Algunas versiones tienen un error en serigrafía, la tapa dice contar con certificación FCC Ce.
ESP-08	GPIO0/2/12 /13/14 /15/16	2 [mm]	Ninguna	17.0 × 16.0	Igual al ESP-07, pero con antena personalizable por el usuario.
ESP-09	GPIO0/2/12 /13/14/15	miscelánea	Ninguna	10.0 × 10.0	Es el módulo más pequeño del mercado, 1[MB] memoria Flash.
ESP-10	Ninguno	2 [mm]	Ninguna	14.2 × 10.0	Solo interfaz UART.
ESP-11	GPIO0/1	1.27 [mm]	Cerámica	17.3 × 12.1	Poca información disponible.
ESP-12	ADC + GPIO0 /2/12/13 /14/15/16	2 [mm]	Grabada en PCB	24.0 × 16.0	El escudo de metal dice contar con certificación FCC, posee 4[MB] de memoria Flash.

Figura 2.6: Características físicas de los módulos basados en SoC ESP8266EX.

2.4. Tarjetas de desarrollo ESP8266

Para el propósito de la investigación de este trabajo, era necesario contar con un sistema que permitiera el desarrollo de pruebas y la aplicación del prototipo para resolver el problema presentado en el capitulo 1, de manera que se recurrió a una de las tarjetas de desarrollo disponibles en el mercado.

Al realizar una búsqueda en los modelos que presentan distintos fabricantes, se seleccionó la opción que presenta las *características mínimas necesarias* que se buscan para el desarrollo de la aplicación de este trabajo y posteriormente se añadió la instrumentación necesaria.

2.5. NodeMCU

NodeMCU es una placa de desarrollo *totalmente abierta*, a nivel de software y de hardware la cual esta basada en el *modulo ESP-12* que a su vez contiene al SoC ESP8266 [11], en la figura 2.7 se puede observar la tarjeta de desarrollo en su version 1.0. Al igual que ocurre con las tarjetas Arduino, el NodeMCU cuenta con la instrumentación necesaria para facilitar la programación del microcontrolador, este conjunto de características permite al usuario la fácil y rápida *implementación de aplicaciones IoT*.

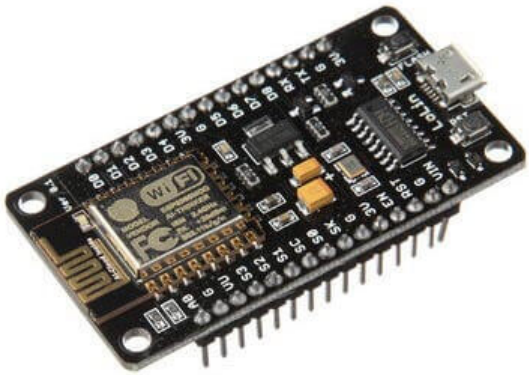


Figura 2.7: NodeMCU en su *version 1.0*.

El NodeMCU comparte características similares a las de otros módulos, en general las características importantes para la aplicación de este trabajo de tesis son:

- *Convertidor serial USB-TTL* integrado, con un socket Micro-USB.
- Voltaje de alimentación DC 4 9V en pin VIN.
- Antena PCB.
- Protocolos de comunicación *UART, SPI, I^2C*.

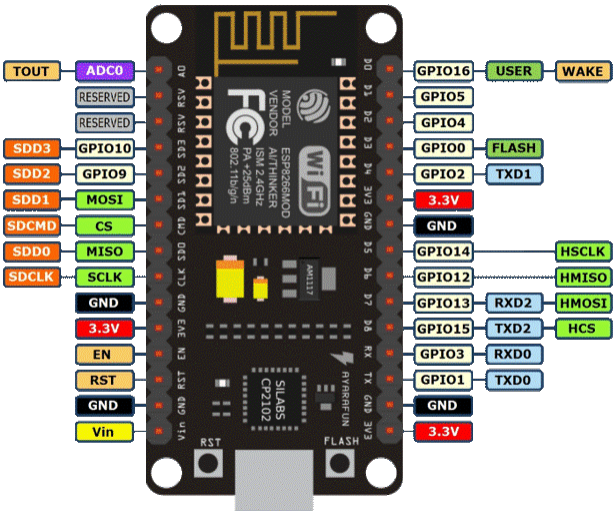


Figura 2.8: *Pin Out* del NodeMCU v1.0.

2.5.1. Arduino Core

Arduino Core for ESP8266 es un proyecto de la comunidad del foro ESP8266 que ofrece soporte a cualquier módulo ESP8266 por medio del entorno de desarrollo Arduino IDE [10]. Permite escribir *sketches* mediante funciones o librerías implementadas en C, y programarlos directamente en el ESP8266 sin la necesidad de un *microcontrolador externo*.

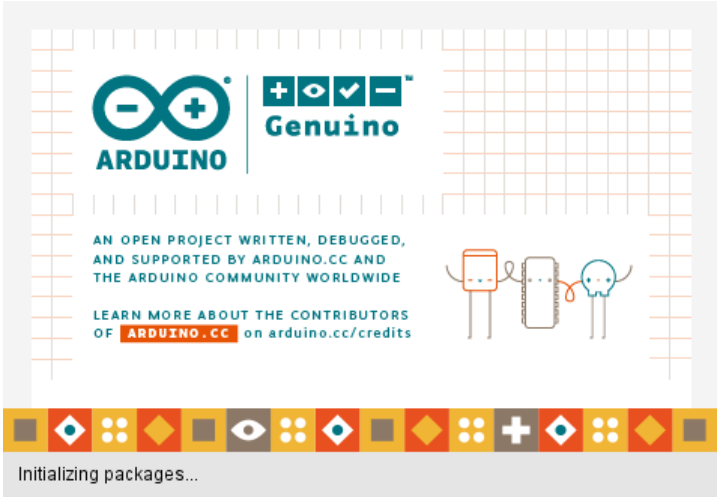


Figura 2.9: Pantalla de carga *Arduino IDE*.

Debido a que el principal enfoque del ESP8266 es en aplicaciones IoT, Arduino Core posee la librería *ESP8266WiFi.h*, la cual cuenta con una gran cantidad de *clases, métodos y propiedades* basadas en el ESP8266 SDK [12], pero utilizando la convención de nombres de las funciones y *filosofía de Arduino*.

El uso de Arduino Core facilita la configuración de las características de red del dispositivo ESP8266 permitiendo que este opere de diferentes maneras y añadiendo *versatilidad* al esquema de transmisión [12]. A continuación, se presentan los modos disponibles para el dispositivo.

2.5.2. Soft Access Point

Un *access point (AP) o punto de acceso*, es un dispositivo que provee acceso a una red Wi-Fi a otros dispositivos y los conecta a una red alámbrica adicional. El ESP8266 puede proveer una funcionalidad similar, sin embargo, no tiene una interfaz para una red alámbrica. Dicho modo de operación se conoce como soft access point (soft-AP) [13].

En la figura 2.10 se puede observar un ejemplo del modulo ESP8266 funcionando en modo de Soft Access Point permitiendo así realizar la transmisión de datos entre varios dispositivos *a través del propio modulo*. La principal aplicación de este modo es el de proveer comunicación entre distintos dispositivos haciendo posible crear una *red de dispositivos en malla*.

Figura 2.10: ESP8266 operando en modo de *Soft Access Point*.

2.5.3. Cliente

Apoyándose de la clase *Client* de Arduino Core es posible configurar el dispositivo como un cliente que accede a servicios provistos por un servidor, de manera que el dispositivo envíe, reciba y procese información [13].

En la figura 2.11 se puede observar un ejemplo del modulo ESP8266 funcionando en modo de Cliente conectándose a la red a través de un Access Point (router) permitiendo al modulo realizar una serie de *solicitudes* a servidores locales y remotos para obtener o enviar información deseada.

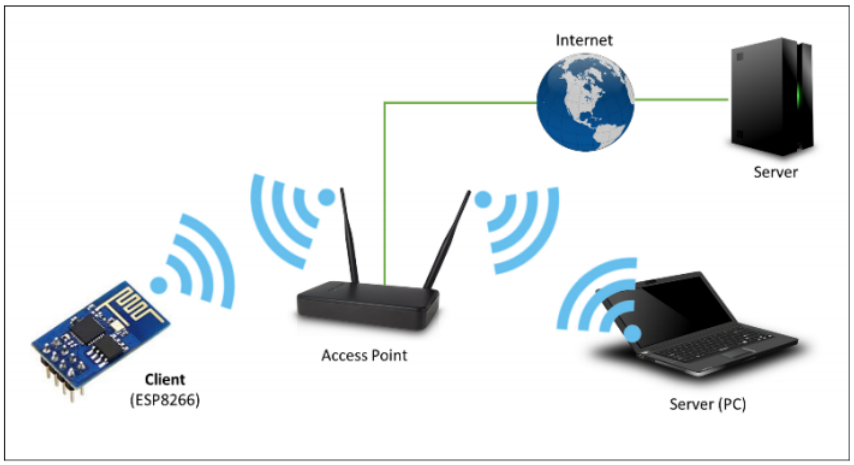


Figura 2.11: ESP8266 operando en modo *Cliente*.

2.5.4. Servidor

Apoyándose de la *clase Server* de Arduino Core es posible configurar el dispositivo como un servidor, de manera que provea funcionalidad a otros programas o dispositivos (clientes) [13].

En la figura 2.12 se puede observar un ejemplo del modulo ESP8266 funcionando en modo de Servidor conectándose a la red a través de un Access Point(router) permitiendo a distintos dispositivos realizar una serie de *solicitudes* al modulo para que este reciba o envíe información deseada.

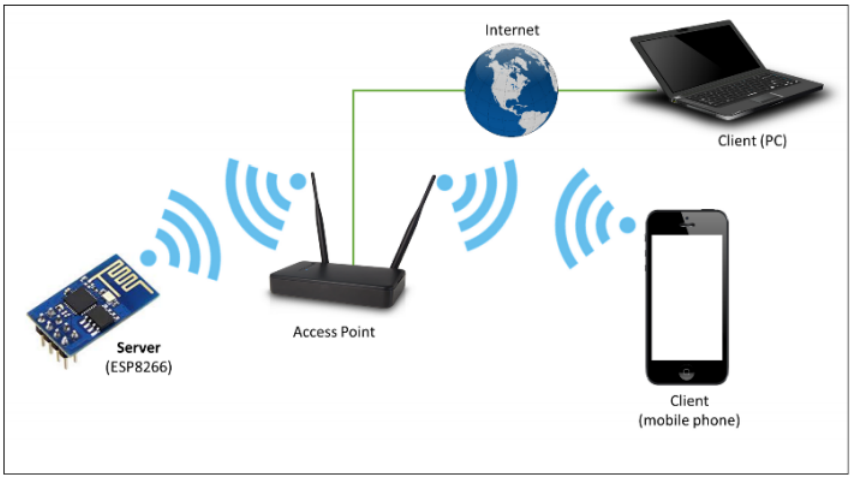


Figura 2.12: ESP8266 operando en modo *Servidor*.

2.6. Servidor LAMP

LAMP es el acrónimo usado para describir un *sistema de infraestructura de internet*, la combinación de estas tecnologías es usada principalmente para definir la infraestructura de un servidor web, utilizando un paradigma de programación para el desarrollo [14].

- *Linux*: es un familia de *sistemas operativos* de código abierto tipo Unix (multiplataforma, multiusuario y multitarea).
- *Apache*: es un *servidor web multiplataforma* de código abierto, cuya principal característica es que presenta una arquitectura modular.
- *MySQL*: es un *sistema de gestión de bases de datos relacional*.
- *PHP*: es un *lenguaje de programación de propósito general* de código del lado del servidor diseñado para el desarrollo web de contenido dinámico.

Figura 2.13: Ejemplo de *componentes* de un servidor LAMP.

Los componentes de la arquitectura LAMP son *ampliamente intercambiables* y no están limitados a la elección inicial. Cuando son combinados, representan un conjunto de soluciones que proporcionan un entorno adecuado para el diseño de sitios web dinámicos y aplicaciones web.

A pesar de que el origen de esos programas de código abierto no fue específicamente diseñado para trabajar entre si, la combinación se popularizo debido a su *bajo coste de adquisición* y *ubicuidad* de sus componentes.

2.7. Raspberry Pi

Raspberry Pi es una *serie de ordenadores de placa unica (single-board computer)* desarrollado en el Reino unido por la *Raspberry Pi Fundation* para promover la enseñanza de informática en escuelas y en países en desarrollo [15]. El modelo original se volvió mas popular de lo anticipado, vendiéndose fuera de su mercado objetivo para usos como robotica o aplicaciones IoT.

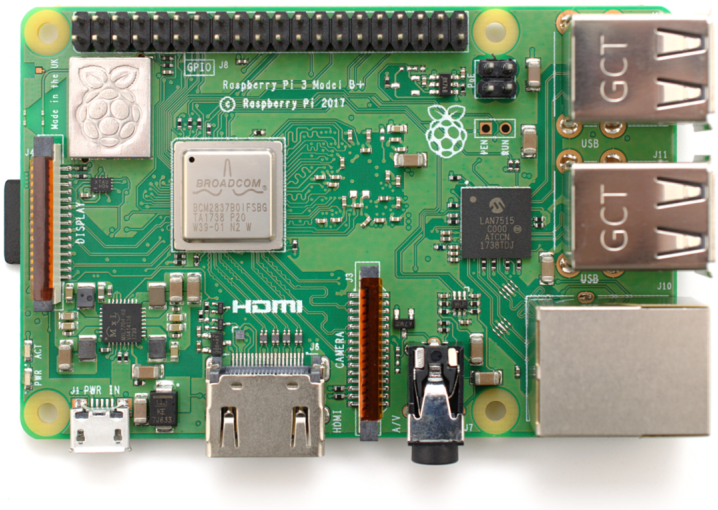


Figura 2.14: Raspberry PI 3 Model B+.

Desde su lanzamiento oficial en 2012, varias modelos han sido lanzados al mercado. Todos los modelos disponibles ofrecen un *SoC de Broadcom* que integra un CPU tipo ARM y un GPU en el mismo chip. La velocidad del procesador esta entre los rangos de 700 MHz a 1.4 GHz dependiendo el modelo, así como sus *características de memoria* que se encuentran entre los 256 MB y 1 GB de RAM. Todas las variantes cuentan con un *lector de tarjetas SD* (Secure Digital) debido a que se utilizan estos dispositivos de memoria para almacenar el sistema operativo y configuraciones de programa. El dispositivo tambien ofrece varias opciones de *interfaces de entrada/salida* como *puertos USB*, *puerto Ethernet*, *salida HDMI y video compuesto* (en modelos antiguos), *jack de 3.5 mm* como salida de audio y varios *pines de proposito general* (GPIO) que permiten interactuar con dispositivos externos y tambien proveen de *protocolos de comunicación* como I^2C o SPI. A partir del modelo Raspberry Pi 3, el dispositivo cuenta con interfaces para *comunicaciones inalambricas* como Wi-Fi y Bluetooth. En la figura 2.15 se pueden observar algunas de las caracteristicas importantes de los modelos lanzados al mercado en su s variantes principales [15].

El Raspberry Pi usa principalmente *sistemas operativos GNU/Linux*. *Raspbian*, es la distribución oficial derivada de *Debian* la cual está optimizada para el Hardware de Raspberry Pi. Raspbian es un sistema operativo gratuito que incluye un conjunto

	Raspberry Pi 3 Model B	Raspberry Pi Zero W	Raspberry Pi Zero	Raspberry Pi 2 Model B	Raspberry Pi Model B
Introduction Date	2/29/2016	2/28/2017	11/25/2015	2/2/2015	4/14/2012
SoC	BCM2837	BCM2835	BCM2835	BCM2836	BCM2835
CPU	Quad Cortex A53 @ 1.2 GHz	ARM 11 @ 1GHz	ARM 11 @ 1GHz	Quad Cortex A7 @ 900 MHz	ARM11 @ 700 MHz
Instruction Set	ARMv8-A (64/32-bit)	ARMv6 (32-bit)	ARMv6 (32-bit)	ARMv7-A (32-bit)	ARMv6 (32-bit)
GPU	400MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	1GB SDRAM	512 MB SDRAM	512 MB SDRAM	1GB SDRAM	256 MB SDRAM
Storage	micro-SD	micro-SD	micro-SD	micro-SD	SD
Ethernet	10/100	none	none	10/100	10/100
Wireless	802.11n / Bluetooth 4.1	802.11n / Bluetooth 4.1	ALGO	none	none
Video Output	HDMI / Composite (GPIO)	HDMI / Composite (GPIO)	HDMI / Composite (GPIO)	HDMI / Composite (GPIO)	HDMI / Composite (GPIO)
GPIO	40	40	40	40	40
Price	$35	$10	$5	$35	$35

Figura 2.15: Características de *modelos de Raspberry Pi* [17][18][19].

básico de programas y herramientas para lograr que la computadora funcione con miles de otros paquetes relacionados al entorno de Linux [16].

Debian tiene una reputación dentro de la comunidad de Linux como una distribución de alta calidad y que presenta una estabilidad robusta. Tambien cuenta con una comunidad bastante grande que ofrece soporte a problemas comunes que encuentran los nuevos usuarios, lo cual lo hace el sistema ideal para el Raspberry Pi tomando en cuenta su público objetivo.

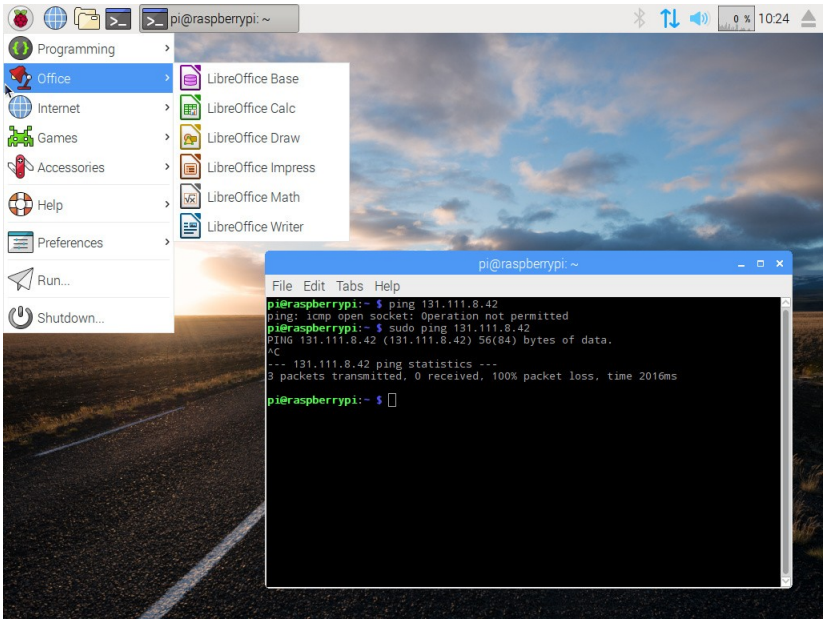


Figura 2.16: Entorno del escritorio de Raspbian Stretch.

2.7.1. Raspberry Pi Zero W

Tomando en cuenta los *objetivos específicos* para resolver el problema propuesto en el capitulo 1, se eligió una variante de Raspberry Pi que cumpliera con las características suficientes para la aplicación. En este caso, se busca que el Raspberry Pi lleve a cabo la función de *servidor local* para implementar el esquema propuesto en el capitulo 1 el cual consiste en la transmisión de datos de un sistema ESP8266 a otro. Puesto que no es necesario una cantidad significativa de poder de procesamiento se recurrió a utilizar la variante "Zero" de la familia de Raspberry Pi, logrando así incluso mantener un costo bajo en todo el prototipo. Para eliminar la necesidad de instrumentar el equipo para una conexión alámbrica a Internet, dentro de la variante "Zero" se eligió el modelo "Zero W" el cual cuenta con una interfaz inalambrica 802.11n / Bluetooth 4.1 [18].

En la figura 2.17 se puede observar el modelo de Raspberry Pi elegido para este trabajo de tesis.

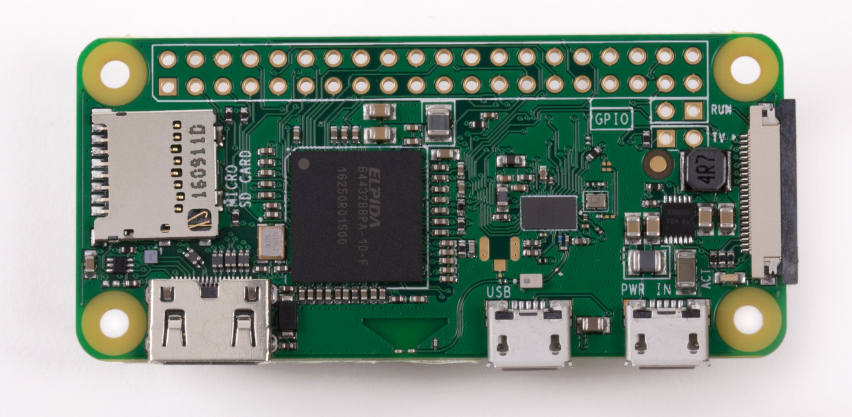


Figura 2.17: Raspberry PI Zero W.

Capítulo 3

Encriptación caótica

3.1. Introducción

En las últimas décadas, ha habido un creciente interés en la posibilidad de utilizar el *caos* en los sistemas de comunicaciones para el diseño e implementación de algoritmos criptográficos que permitan una transmisión segura de datos.

En la era de la información, con el rápido desarrollo y la utilización de diversas aplicaciones web, y debido que, por su naturaleza, Internet habilita la transmisión de datos en tiempo real, la seguridad de la información *"sensible"* se volvió un asunto importante, ya que durante la transmisión, la privacidad de la información puede ser comprometida por un tercero. La utilización del caos en la criptografía emergió como una solución potencial a estos problemas debido a 3 características fundamentales de los sistemas caóticos [20]:

1. Son sistemas dinámicos no lineales determinísticos.
2. Exhiben sensibilidad y dependencia a condiciones iniciales.
3. Generan dinámicas de pseudoaleatorias.

3.2. Criptografía

A lo largo del desarrollo de la humanidad, ha existido la necesidad de ocultar información, incluso mucho antes de que existieran los primeros sistemas electrónicos. La *criptografía moderna* es el estudio de técnicas y algoritmos matemáticos para la protección de información, lo cual permite una comunicación segura entre un transmisor y un receptor [22]. En general, se busca mantener oculta la información de un mensaje enviado a través de un canal público fácilmente observable por un tercero, algo muy importante en una sociedad con gran intercambio de información.

Con los avances en la tecnología y los sistemas de comunicaciones gran parte de nuestra información se encuentra moviéndose de un lugar a otro, muchas veces a través

de medios o canales no confiables, lo cual hace de la criptografía una necesidad para la construcción y el análisis de protocolos seguros.

Existen 4 funciones primarias de la criptografía moderna [24]:

1. *Privacidad*: asegura que nadie pueda leer el mensaje excepto el receptor destinado.
2. *Autenticación*: permite probar la identidad de un individuo.
3. *Integridad*: asegura que el mensaje recibido no a sido alterado de ninguna manera.
4. *No rechazo*: permite probar que el transmisor realmente envió ese mensaje.

3.3. Encriptación de clave simétrica

Como se mencionó anteriormente, la criptografía se interesa en la construcción y análisis de esquemas de encriptación para proveer comunicaciones secretas entre dos partes. El esquema de *clave simétrica* ofrece un escenario donde las dos partes comparten información secreta llamada clave mediante un canal seguro y utilizan esta clave cuando desean comunicarse secretamente el uno al otro. La parte que envía el mensaje utiliza la clave para encriptar o *"esconder"* la información antes de enviarla, mientras que el receptor utiliza la clave para desencriptar o *"revelar"* el mensaje recibido [23].

Se asume que en cualquier sistema de clave simétrica, hay alguna manera en la que inicialmente ambas partes pueden compartir la clave de forma segura. En muchos escenarios actuales, es imposible que ambas partes puedan organizar este intercambio de información. Por lo cual es necesaria la implementación otro tipo de esquemas de encriptación como lo son los esquemas asíncronos [23], sin embargo el esquema de encriptación de clave simétrica es suficiente para muchas de las aplicaciones actuales.

3.3.1. Implementación del esquema de clave simétrica

El esquema de encriptación de clave simétrica está compuesto por tres partes principales, la primera es un procedimiento para la generación de claves, la segunda es el procedimiento para la encriptación y la tercera es el procedimiento para la desencriptación. Las partes que componen el esquema tienen la siguiente funcionalidad:

1. La *elección de la clave secreta* se basa en las características que presenta el algoritmo utilizado para la encriptación/desencriptación de los datos, por lo que la *longitud y el rango de valores* que puede tomar la clave varían de una implementación a otra.
2. El *algoritmo de encriptamiento* toma como entrada una *clave* k y un *mensaje* m, y entrega como salida un mensaje cifrado también llamado *criptograma* c.

3. El *algoritmo de desencriptamiento* toma como entrada una *clave* k y el *criptograma* c, y entrega como salida un *mensaje* $\hat{m}$.

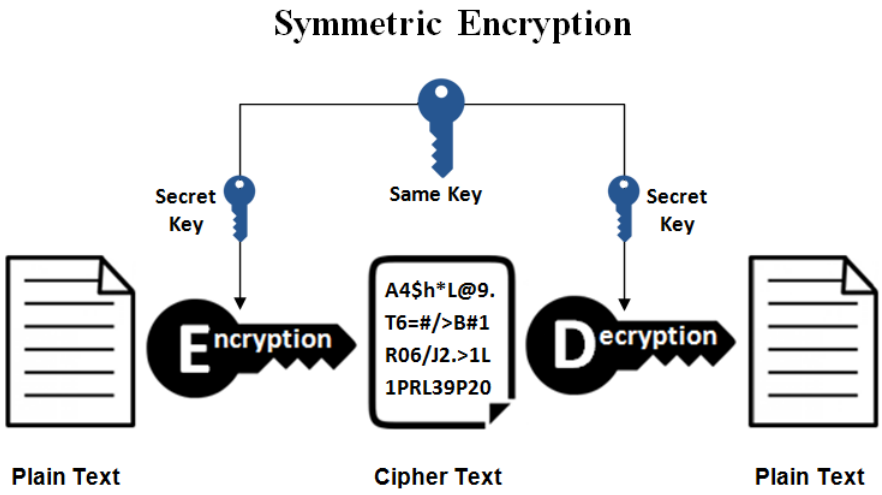


Figura 3.1: Esquema de encriptación de clave simétrica.

3.4. Teoría del caos

La teoría del caos es una rama de las matemáticas que se enfoca en el comportamiento de *sistemas dinámicos no lineales* que presentan gran *sensibilidad* a condiciones iniciales. Es una teoría que establece que, dentro de la aparente aleatoriedad de los sistemas complejos caóticos, hay patrones subyacentes, ciclos constantes de retroalimentación, repetición y fractales entre otras características [25].

El caos está presente en muchos sistemas complejos naturales como el clima y la astronomía; también ocurre en sistemas artificiales como la economía y el tráfico; el estudio del comportamiento de estos sistemas se realiza a través de su modelo matemático caótico.

A pesar de su simplicidad determinística, pequeñas diferencias en las condiciones iniciales, provocan que estos sistemas pueden producir a través del tiempo resultados ampliamente divergentes, generalmente haciendo la predicción a largo plazo de su comportamiento imposible.

Edward Lorenz, el padre de la teoría del caos describe a este como: " Cuando el presente determina el futuro, pero el presente aproximado no determina el futuro aproximado [25]"

Todos los sistemas caóticos presentan las siguientes características:

- *Deterministas*: su modelo matemático presenta una evolución única, es decir, un estado dado del modelo siempre es seguido por la misma historia de transiciones de estados.
- *No lineales*: el comportamiento de su modelo matemático no es expresable como la suma de los comportamientos de estados independientes, es decir, no cumple con el principio de superposición.
- *Sensibles*: cada punto en un sistema caótico ésta arbitrariamente aproximado a otros puntos con trayectorias futuras significativamente diferentes. De manera que, un cambio arbitrariamente pequeño (perturbación) puede llevar a un comportamiento futuro completamente diferente.

3.5. Generación del caos en el sistema NodeMCU

El primer paso para la implementación del esquema propuesto en el capítulo 1, es verificar que la secuencia caótica generada en el microcontrolador cumpla con las características necesarias para la aplicación. Primeramente, tomando en cuenta las limitaciones en cuanto al hardware utilizado, se busca utilizar un mapa caótico lo suficientemente sencillo para mantener la generación de la secuencia caótica dentro de límites prácticos de tiempo.

También, es necesario comprobar que el mapa caótico presente sensibilidad a condiciones iniciales, tomando en cuenta que se está intentando generar una secuencia caótica en una máquina de precisión finita, lo cual puede producir a una degradación de las propiedades dinámicas caóticas [21].

En la figura 3.2 se puede observar una muestra de las capacidades del microcontrolador, el atractor se generó al realizar 1000 iteraciones con 1000 condiciones iniciales distintas, el tiempo de procesamiento de los 2,000,000 de puntos fue de 59.92 segundos, un tiempo no muy grande tomando en cuenta las limitaciones del sistema y la complejidad de las ecuaciones que presenta el mapa de Ikeda.

Se seleccionaron 3 mapas caóticos ampliamente utilizados actualmente en sistemas de recursos limitados, para comparar sus características y seleccionar la opción óptima para la aplicación de este trabajo de tesis.

Para obtener el tiempo de cálculo de una muestra, se realiza una prueba generando 10 000 iteraciones de cada mapa caótico en el microcontrolador, se utiliza la diferencia entre el tiempo de inicio y el tiempo final para cada mapa. Finalmente, se envía por comunicación serial las secuencias generadas en el microcontrolador para su posterior análisis en MATLAB.

Se generan las secuencias caóticas para los 3 mapas con las mismas condiciones iniciales utilizadas en el microcontrolador en MATLAB para verificar si existen diferencias entre las dos plataformas.

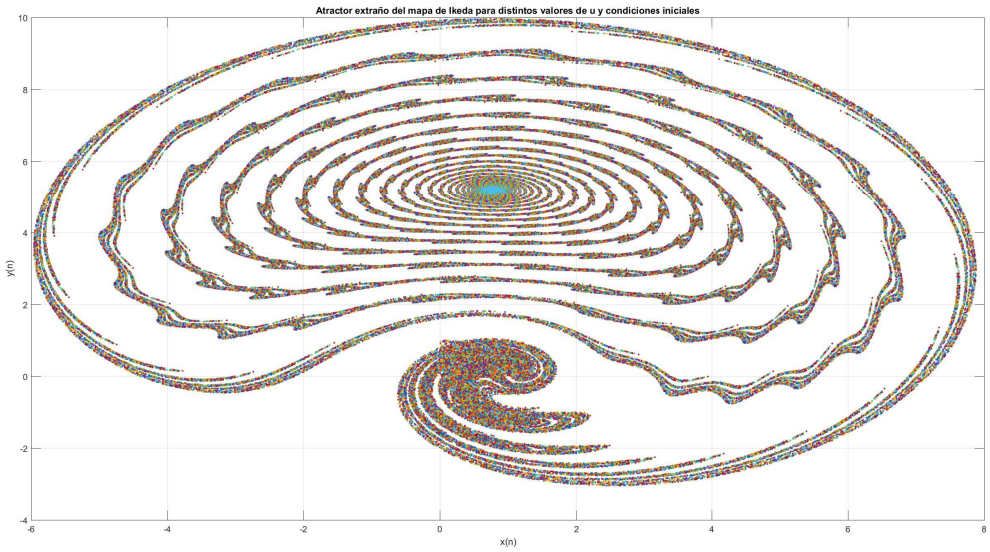


Figura 3.2: Atractor del Mapa de Ikeda para distintas condiciones iniciales.

3.5.1. Mapa Logístico

El mapa logístico es un modelo basado en la función logística de curva en forma de S, que muestra como una población crece lentamente y luego tiene un crecimiento rápido a medida de que alcanza su capacidad de carga. A diferencia de la función logística que utiliza una ecuación diferencial en tiempo continuo, el mapa logístico en cambio, usa una ecuación no lineal de diferencias que toma valores de tiempo discretos [27].

El dinámica del mapa logístico está definido por la siguiente ecuación:

$$x_{(n+1)} = r x_n (1 - x_n) \tag{3.1}$$

Donde x es el estado del mapa logístico y r es el parámetro de control.

La figura 3.3 muestra el estado del mapa caótico para las primeras 200 iteraciones, se utilizó como parámetro de control $r = 3.762954$ y como condición inicial $x_n = 0.683929$.

La figura 3.4 muestra el error entre la secuencia generada en el microcontrolador y la secuencia generada en Matlab, se puede observar que el error entre las dos secuencias es de cero, lo cual implica que las dos plataformas presentan la misma dinámica caótica.

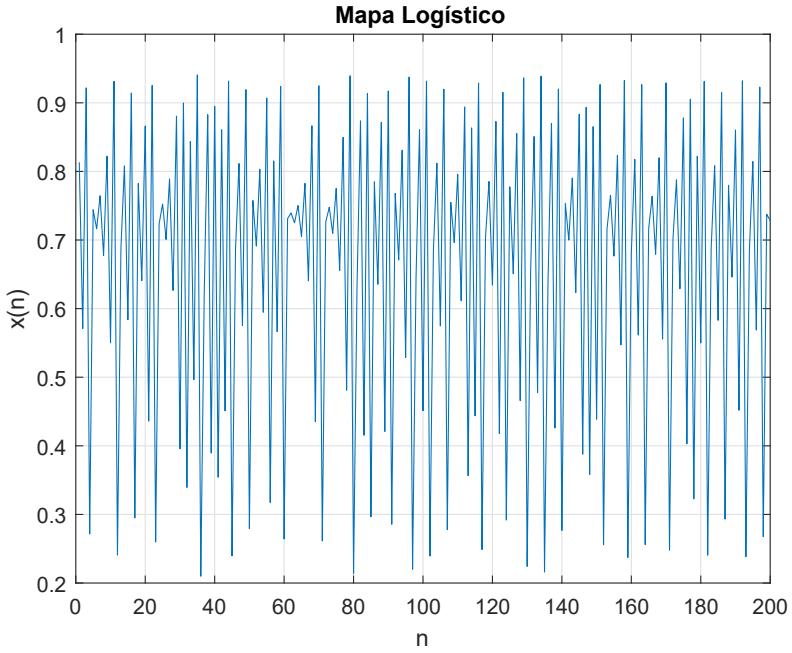


Figura 3.3: Estado del mapa logístico.

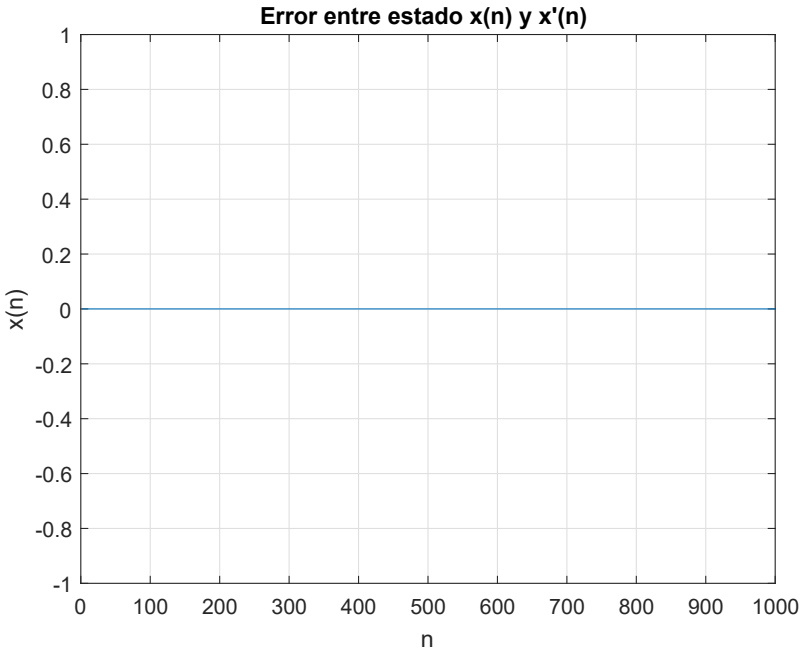


Figura 3.4: Error entre las 2 secuencias generadas del mapa logístico.

3.5.2. Mapa de Hénon

El mapa de Hénon es un sistema dinámico no lineal de tiempo discreto que exhibe dinámica caótica en dos dimensiones [26].

El mapa de Hénon está definido por el siguiente conjunto de ecuaciones:

$$x_{(n+1)} = 1 - ax_n^2 + y_n \tag{3.2a}$$
$$y_{(n+1)} = b_{x_n} \tag{3.2b}$$

Donde x y y representan los estados del mapa de Hénon, mientras que a y b son los parámetros de control los cuales permiten definir las distintas dinámicas del sistema.

La figura 3.5 muestra los estados del mapa caótico así como su atractor para las primeras 100 iteraciones, se utilizaron como parámetros de control $a = 1.4$ y $b = 0.3$ los cuales corresponden a la configuración básica del mapa de Hénon y como condiciones iniciales se tomaron $x_n = 0.2$ y $y_n = 0.2$.

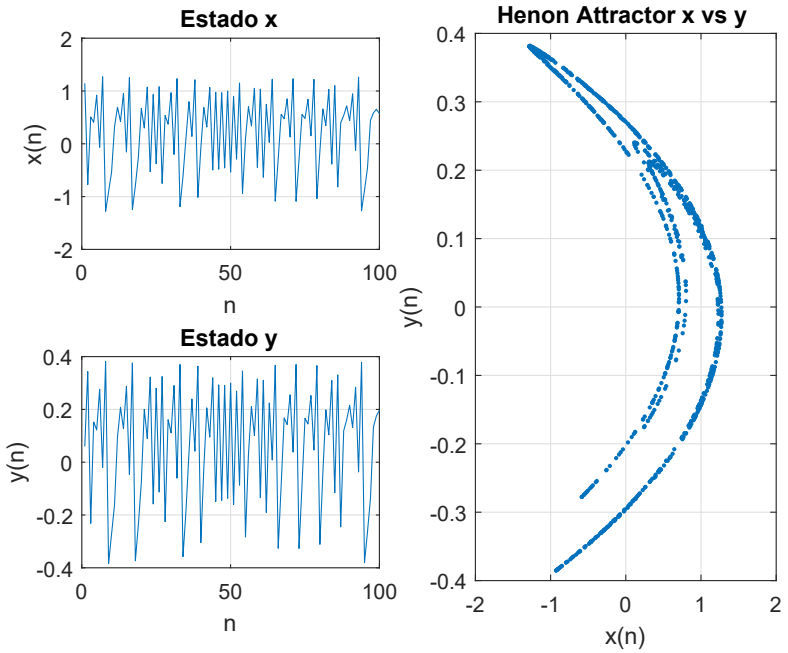


Figura 3.5: Estados y atractor del mapa de Hénon.

La figura 3.6 muestra el error entre las secuencias generadas en el microcontrolador y la secuencias generadas en Matlab, se puede observar que el error entre las secuencias es de cero, lo cual implica que las dos plataformas presentan la misma dinámica caótica.

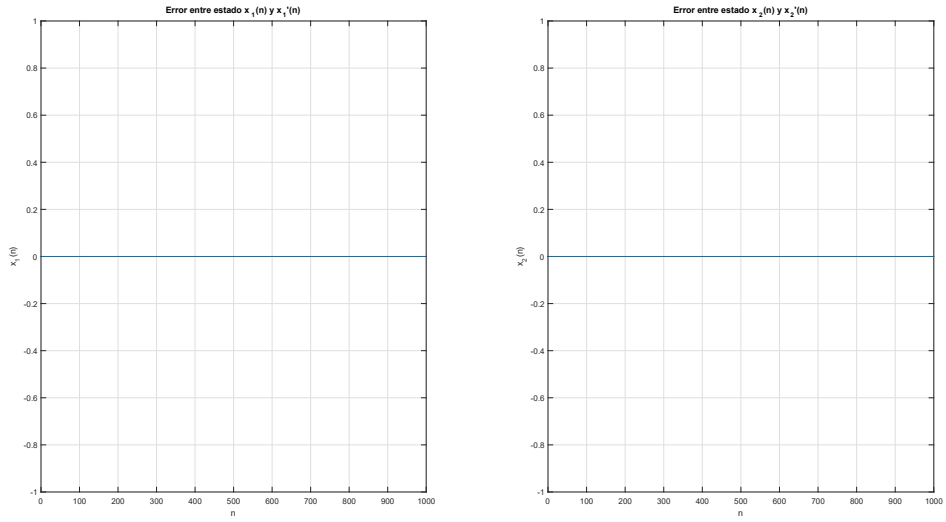


Figura 3.6: Error entre los estados de las 2 secuencias generadas del mapa de Hénon.

3.5.3. Mapa de Ikeda

El mapa de Ikeda es un sistema dinámico no lineal de tiempo discreto que exhibe dinámica caótica en dos dimensiones, el mapa original fue propuesto primeramente como un modelo de la luz que pasa atravez de un resonador optico no lineal.

El mapa de Ikeda está definido por el siguiente conjunto de ecuaciones:

$$x_{(n+1)} = q - u(x_n cost_n - y_n sint_n) \tag{3.3a}$$
$$y_{(n+1)} = u(x_n sint_n + y_n cost_n) \tag{3.3b}$$
$$t_{(n+1)} = 0.4 - \frac{6}{1 + x_n^2 + y_n^2} \tag{3.3c}$$

Donde x y y representan los estados del mapa de Ikeda, mientras que q y u son los parámetros de control los cuales permiten definir las distintas dinámicas del sistema.

La figura 3.7 muestra los estados del mapa caótico así como su atractor para las primeras 100 iteraciones, se utilizaron como parámetros de control $a = 1.0$ y $u = 0.84$ los cuales corresponden al mapa de Ikeda con comportamiento caótico y como condiciones iniciales se tomaron $x_n = 0.35$ y $y_n = 0.28$.

La figura 3.8 muestra el error entre las secuencias generadas en el microcontrolador y la secuencias generadas en Matlab, se puede observar que el error entre las secuencias es de cero, lo cual implica que las dos plataformas presentan la misma dinámica caótica.

Para el caso particular del mapa de Ikeda, los resultados presentados en la figura 3.8 son interesantes, tomando en cuenta la complejidad del conjunto de ecuaciones que

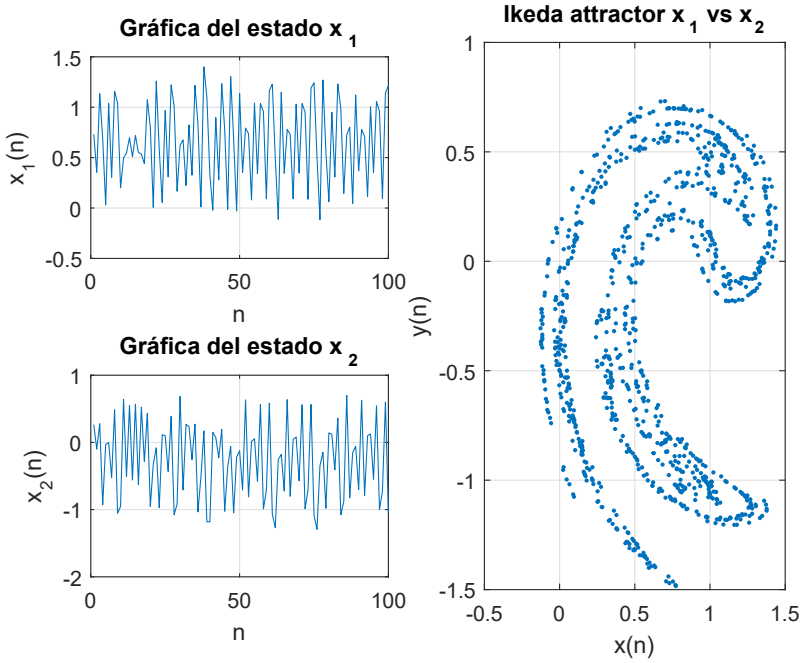


Figura 3.7: Estados y atractor del mapa de Ikeda.

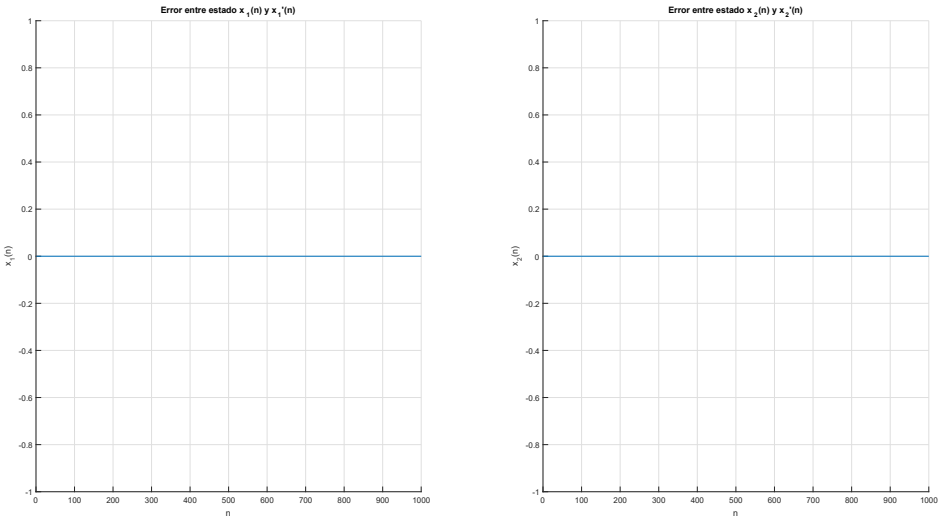


Figura 3.8: Error entre los estados de las 2 secuencias generadas del mapa de Ikeda.

representan el mapa de Ikeda. El microcontrolador realizó las operaciones matematicas seno, coseno e incluso una división de punto flotante de la misma manera que se realizaron en Matlab.

3.6. Resultados

Se implementaron los conjuntos de ecuaciones que definen a los tres mapas caóticos presentados anteriormente en un programa del NodeMCU (ver Apéndice X) y se observaron los siguientes resultados:

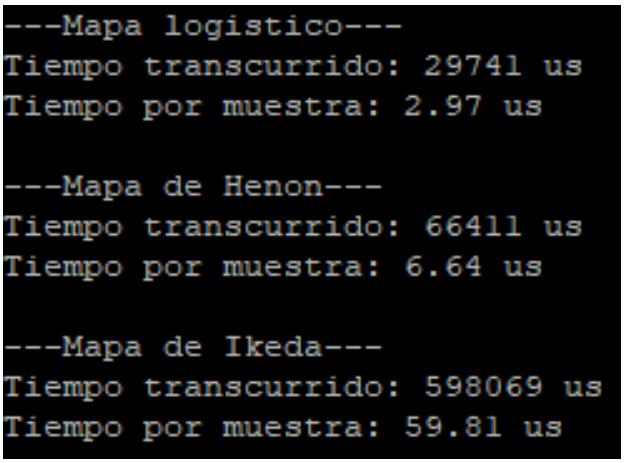


Figura 3.9: Tiempo de procesamiento para cada uno de los 3 mapas caóticos.

Se observa que el mapa logístico presenta el menor tiempo por muestra en comparación con los otros mapas, lo cual es evidente debido a que el mapa caótico presenta el menor número de ecuaciones, reduciendo así la cantidad de operaciones que el microcontrolador debe realizar.

Un tiempo por muestra bajo es una característica deseable para la aplicación de este trabajo de tesis, ya que un tiempo por muestra grande afecta negativamente el desempeño del sistema a la hora de realizar la encriptación de archivos grandes (imágenes de alta calidad, vídeos, etc...).

3.7. Sensibilidad a condiciones iniciales del Mapa Logístico

Una vez que se selecciono el mapa caótico adecuado para la aplicación de este trabajo de tesis, se realizaron algunas pruebas para determinar sus propiedades caóticas, la mas sencilla de ellas implica generar dos secuencias caóticas del mapa logístico con condiciones iniciales ligeramente diferentes para comprobar si el sistema presenta sensibilidad.

En la figura 3.10 se muestran las secuencias generadas del mapa caótico logístico para las primeras 200 iteraciones con condiciones ligeramente diferentes, se utilizó como parámetro de control $r = 3.762954$ y como condiciones iniciales $x_n = 0.683929$ y $x'_n = 0.683930$. es decir un cambio de 0.000001 entre ambas condiciones.

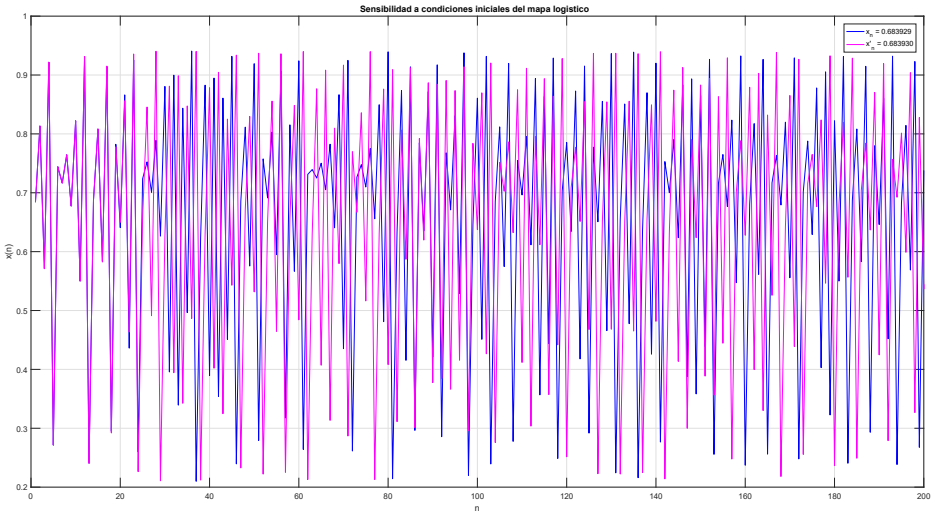


Figura 3.10: Estado del mapa logístico para dos condiciones iniciales cercanas.

En la figura 3.11 se muestra el cálculo del error es decir, la diferencia entre las dos secuencias caóticas generadas en el microcontrolador. Se puede observar que para las primeras 20 iteraciones, el error es cercano a cero, sin embargo a partir de ese punto el error entre las dos secuencias se hace notorio lo cual demuestre que el mapa logístico implementado en el microcontrolador presenta sensibilidad a condiciones iniciales.

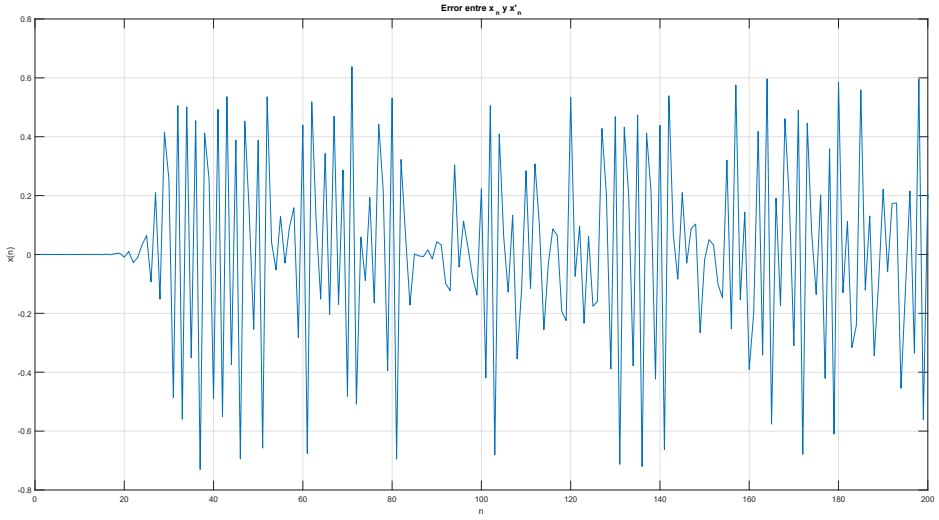


Figura 3.11: Mapa logístico, error entre dos condiciones iniciales cercanas.

Capítulo 4

Implementación y resultados experimentales

4.1. Introducción

En los capítulos anteriores, se introdujo el hardware utilizado para la implementación del esquema propuesto en este trabajo de tesis. También, se mostró que es posible la generación de secuencias basadas en mapas caóticos en un microcontrolador de 32 bits manteniendo un tiempo de procesamiento relativamente bajo y se comprobó que la secuencias generadas en el dispositivo son idénticas a las generadas por plataformas comúnmente utilizadas para el análisis numérico de sistemas complejos. La implementación presentada en este capítulo, es el resultado de realizar una aproximación física a la solución propuesta, en donde también fueron considerados los objetivos específicos y las limitaciones del diseño.

En el desarrollo de este capítulo se expone la implementación tanto de hardware como de software de cada uno de los bloques funcionales del sistema y se indica su interacción entre ellos. También, se presentan los resultados de la fase de experimentación constituida por dos pruebas. Cada experimento consiste en realizar la encriptación de un archivo de texto y una imagen en el microcontrolador y transmitir el criptograma generado al servidor local para su almacenamiento mediante una comunicación Wi-Fi previamente establecida. Posteriormente, el usuario receptor solicita los archivos encriptados y una vez que son recibidos por este, se realiza el proceso de desencriptacion obteniéndose así el archivo original. Finalmente, se realizan una serie de pruebas para observar si el sistema cumple con los objetivos particulares expuestos en el capitulo 1.

4.2. Bloque transmisor

El bloque transmisor consiste en un sistema para realizar la encriptación de un archivo obtenido desde un dispositivo de almacenamiento micro-SD (Secure Digital). Primeramente, se genera una secuencia caótica en el microcontrolador y se realiza el encriptamiento aditivo con el archivo obtenido desde el dispositivo de almacenamiento. El encriptamiento se realiza mediante un algoritmo de cifrado de flujo. Finalmente, el sistema transmite el archivo de manera inalámbrica a un servidor local, donde el archivo encriptado es almacenado para su posterior uso.

En la figura 4.1 se puede observar el esquema electrónico que representa la implementación del bloque transmisor. El bloque consiste en un modulo lector de tarjetas micro-SD conectada a una tarjeta de desarrollo NodeMCU. La lectura de la informacion en la tarjeta se realiza mediante el protocolo de datos SPI utilizando los pines que corresponden a la interfaz SPI del hardware del microcontrolador.

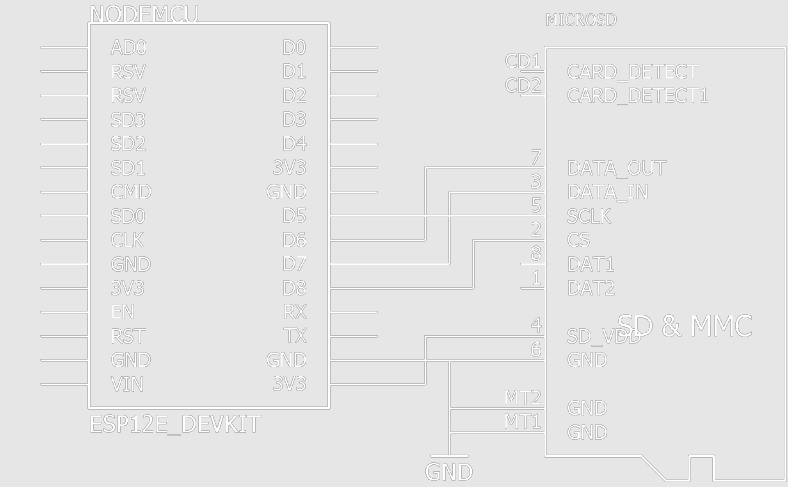


Figura 4.1: Esquema electrónico del bloque transmisor.

Para poder desempeñar las tareas requeridas por el bloque transmisor, es necesario programar el microcontrolador mediante el lenguaje de programación C. Esto se logra utilizando el software Arduino IDE y haciendo uso de la implementacion del "Arduino Core for ESP8266". En el apéndice A se puede observar el codigo con el que se programa el bloque transmisor.

A continuación se enumeran los pasos que debe realizar el microcontrolador para realizar una transmisión exitosa en el esquema propuesto:

1. Inicializar el sistema y realizar una conexión con el servidor.
2. Leer el archivo con información médica desde tarjeta micro-SD.
3. Generar una secuencia mediante un mapa caótico por medio de una clave compartida con el receptor.

4. Encriptar aditivamente utilizando el archivo leído y la secuencia caótica generada.
5. Transmitir del archivo encriptado (o criptograma) a el servidor LAMP local.
6. Imprimir la respuesta del servidor, velocidad de transmisión e información relevante de la conexión realizada.

4.2.1. Algoritmo de encriptación

El algoritmo de encriptación esta basado en el cifrado de flujo y se describe como sigue:

1. Se lee una serie de bytes desde el archivo M hasta llenar el buffer clientBuf de longitud L.
2. Se genera un valor de la secuencia mediante un mapa caótico y se almacena como una variable del tipo Double (8 Bytes).
3. Se descompone el valor obtenido anteriormente en un buffer b de longitud de 8 bytes.
4. Se suma el buffer b a clientBuf empezando en la posición i de cada buffer.
5. Se realizan los pasos del 2 al 4 iniciando en los siguientes 8 bytes del clientBuf hasta llegar al último byte del dicho buffer.
6. Se realiza la transmisión del buffer clientBuf al servidor.
7. Se realizan los pasos del 1 al 6 iniciando en los siguientes L bytes del archivo hasta que se llegue a la ultimo byte del archivo.

4.3. Bloque receptor

El bloque receptor consiste en un sistema para realizar la desencriptación de un archivo obtenido desde un servidor local micro-SD. Primeramente, el sistema recibe un archivo de forma inalámbrica desde un servidor local, se genera una secuencia caótica en el microcontrolador y se realiza el desencriptamiento aditivo con el archivo obtenido. El desencriptamiento se realiza mediante un algoritmo de cifrado de flujo. Finalmente, el sistema almacena el archivo en el dispositivo de almacenamiento micro-SD para su posterior uso. El archivo con información medica es idéntico al que se encuentra del lado del transmisor.

En la figura 4.2 se puede observar el esquema electrónico que representa la implementación del bloque receptor. El bloque consiste en un modulo lector de tarjetas micro-SD conectada a una tarjeta de desarrollo NodeMCU. La escritura de la informacion en la tarjeta se realiza mediante el protocolo de datos SPI utilizando los pines que corresponden a la interfaz SPI del hardware del microcontrolador.

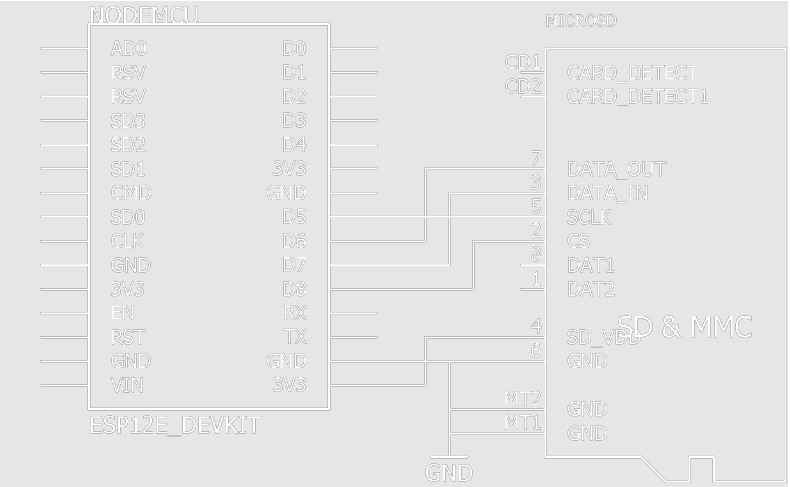


Figura 4.2: Esquema electrónico del bloque receptor.

Para poder desempeñar las tareas requeridas por el bloque receptor, es necesario programar el microcontrolador mediante el lenguaje de programación C. Esto se logra utilizando el software Arduino IDE y haciendo uso de la implementacion del "Arduino Core for ESP8266". En el apéndice B se puede observar el codigo con el que se programa el bloque transmisor.

A continuación se enumeran los pasos que debe realizar el microcontrolador para realizar una recepción exitosa en el esquema propuesto:

1. Inicializar el sistema y realizar una conexión con el servidor.
2. Descargar el archivo con información médica encriptado desde el servidor LAMP local.
3. Generar una secuencia mediante un mapa caótico por medio de una clave compartida con el receptor.
4. Desencriptar aditivamente utilizando el archivo descargado y la secuencia caótica generada.
5. Guardar el archivo con información médica en la tarjeta micro-SD.
6. Imprimir la respuesta del servidor, velocidad de recepción e información relevante de la conexión realizada.

4.3.1. Algoritmo de desencriptación

El algoritmo de desencriptación esta basado en el cifrado de flujo y se describe como sigue:

1. Se lee una serie de bytes desde el servidor hasta llenar el buffer clientBuf de longitud L.

2. Se genera un valor de la secuencia mediante mapa caótico y se almacena como una variable del tipo Double (8 bytes).

3. Se descompone el valor obtenido anteriormente en un buffer b de longitud de 8 bytes.

4. Se resta el buffer b a clientBuf empezando en la posición i de cada buffer.

5. Se realizan los pasos del 2 al 4 iniciando en los siguientes 8 bytes del clientBuf hasta llegar al último byte de dicho buffer.

6. Se realiza el grabado del buffer clientBuf en un archivo generado en la tarjeta SD.

7. Se realizan los pasos del 1 al 6 iniciando en los siguientes L bytes del archivo hasta que se llegue al ultimo byte de la respuesta del servidor.

4.4. Servidor LAMP

En este trabajo de tesis se realiza la implementacion de un servidor LAMP en un Raspberry Pi W. La función del servidor es la de almacenar archivos recibidos por medio del bloque transmisor y enviar dichos archivos hacia el bloque receptor cuando este lo solicite. Recordando el acrónimo LAMP, la implementación se presenta como sigue:

- L: se utiliza Raspbian, la distribucion oficial del sistema operativo para Raspberry Pi la cual está basada en linux debian.
- A: se utiliza Apache 2 como servidor web.
- M: para la aplicación no se hace uso de un servicio de base de datos, los archivos recibidos son alamcenados directamente en carpetas.
- P: se utiliza PHP 5.0 como lenguaje de programación del lado del servidor.

En la figura 4.3 se puede observar la pantalla principal que muestra el servidor cuando se accede desde cualquier navegador web.

Como se mencionó antes, la interacción entre los clientes (transmisor y receptor) y el servidor montado en el Raspberry Pi se lleva a cabo mediante el lenguaje PHP. El servidor realiza el procesamiento de las solicitudes de los dos bloques mediante dos scripts (programas) en PHP los cuales se muestran a detalle en el Anexo 3.

El sistema transmisor realiza una POST request al servidor, la cual es procesada mediante un script que consiste en un código simple para el manejo de subida de archivos, el formato y el nombre del archivo son obtenidos del header de la misma POST request.

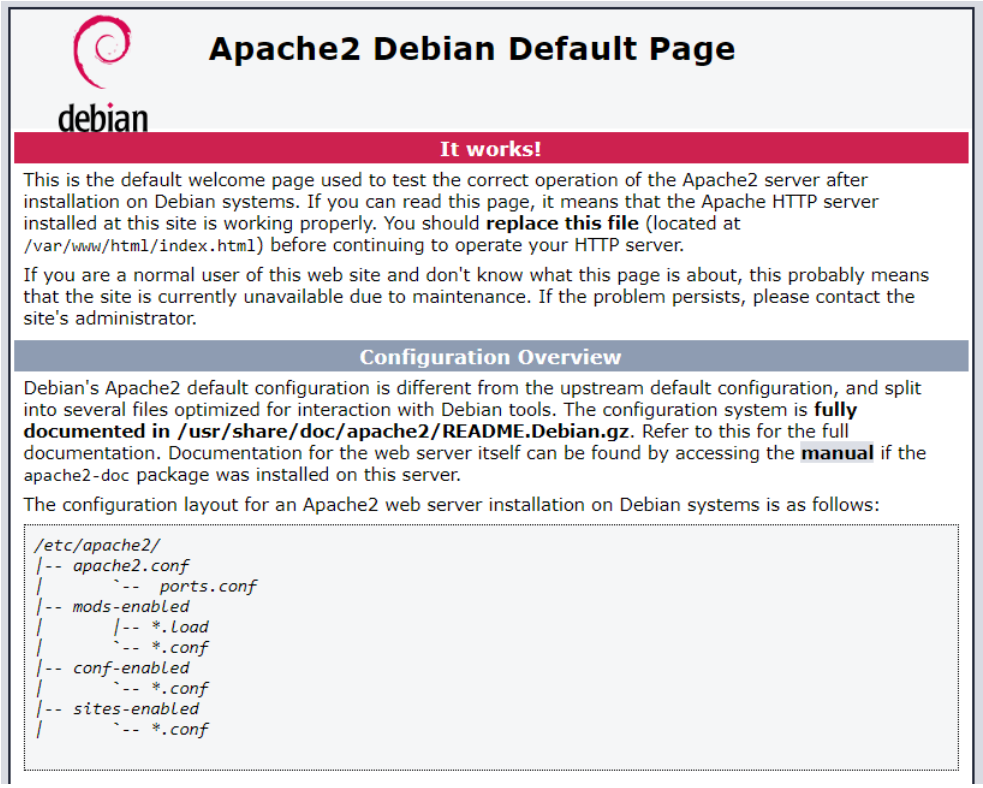


Figura 4.3: Pantalla principal del servidor LAMP.

El sistema receptor realiza una GET request al servidor, la cual es procesada mediante un script que consiste en un código simple para el manejo de bajada de archivos, el nombre y formato del archivo solicitado por el cliente se obtienen a través de la respuesta del servidor.

4.5. Resultados experimentales

Una vez caracterizado el hardware y con los algoritmos desarrollados para la implementación del esquema propuesto, se procede a realizar dos pruebas con dos tipos de archivos. Uno de los archivos corresponde a un archivo csv de tipo texto, mientras que el otro archivo es un archivo tipo ppm el cual corresponde a una imagen. En la figura 4.4 se observa el sistema físico montado. Se pueden observar los dos bloques transmisor y receptor con su respectiva electrónica necesaria y el Raspberry Pi que corresponde al servidor.

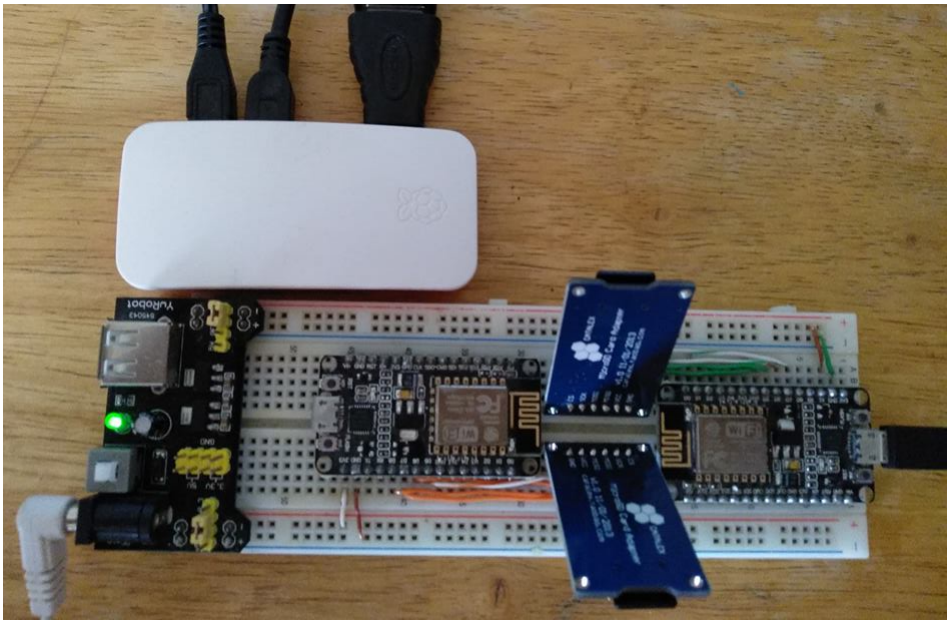


Figura 4.4: Implementación final del esquema propuesto para este trabajo de tesis.

4.5.1. Encriptación de archivo de texto

Para el experimento con el archivo de texto se realiza una prueba con el transmisor donde se utiliza como información a encriptar un archivo CSV que contiene una secuencia ECG de 10,000 muestras, las cuales fueron obtenidas por el módulo ADS129R, con una frecuencia de muestreo de 512 Hz y una resolución de 16 bits.

En la figura 4.5 se puede observar la señal ECG a encriptar la cual se tiene como un archivo tipo CSV.

Con un editor hexadecimal se analiza el archivo CSV de la señal ECG para observar su contenido a nivel byte. La figura 4.6 muestra la pantalla principal del editor, en donde el lado izquierdo se observa la representación hexadecimal de un fragmento inicial de bytes del archivo y en la parte derecha muestra su representación en código ASCII.

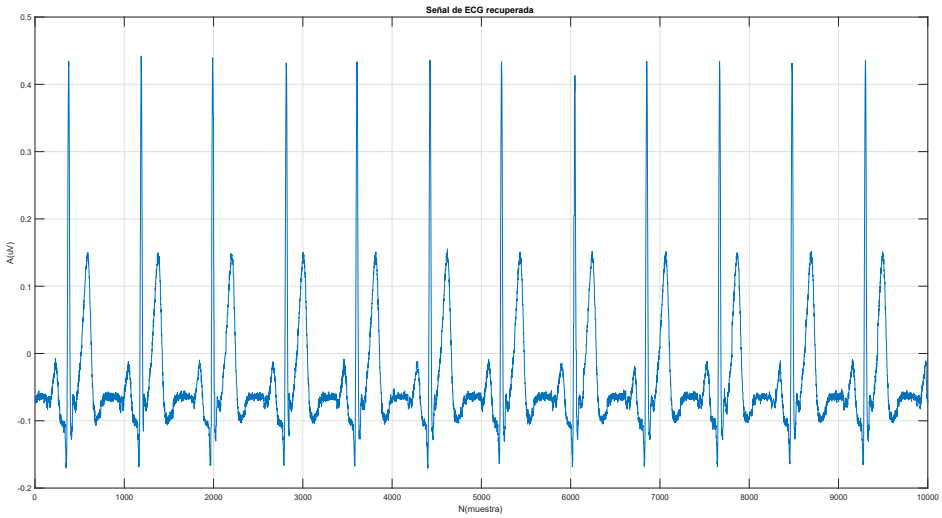


Figura 4.5: Señal de ECG a encriptar.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	2D	30	2E	30	36	35	36	33	30	36	38	32	0D	0A	2D	30	-0.065630682..-0
00000010	2E	30	36	35	36	33	30	36	38	32	0D	0A	2D	30	2E	30	.065630682..-0.0
00000020	36	34	35	34	38	38	35	38	0D	0A	2D	30	2E	30	36	34	64548858..-0.064
00000030	39	30	39	34	36	36	0D	0A	2D	30	2E	30	36	36	33	35	909466..-0.06635
00000040	31	38	39	38	0D	0A	2D	30	2E	30	36	38	38	37	36	31	1898..-0.0688761
00000050	35	35	0D	0A	2D	30	2E	30	36	39	32	33	36	37	36	34	55..-0.069236764
00000060	0D	0A	2D	30	2E	30	37	33	39	32	34	36	36	39	0D	0A	..-0.073924669..
00000070	2D	30	2E	30	37	32	38	34	32	38	34	35	0D	0A	2D	30	-0.072842845..-0
00000080	2E	30	36	33	38	32	37	36	34	31	0D	0A	2D	30	2E	30	.063827641..-0.0
00000090	36	36	37	31	32	35	30	37	0D	0A	2D	30	2E	30	36	35	66712507..-0.065

Figura 4.6: Fragmento del archivo ecg.csv en un editor hexadecimal.

Una vez que se verifican las características del archivo, este se almacena en una memoria micro-SD y se introduce al sistema transmisor para realizar una prueba. La figura 4.7 muestra la información obtenida a través de la comunicación serial con el dispositivo, se puede apreciar información relevante de la conexión realizada con el servidor, características del archivo a encriptar y velocidad de transmisión.

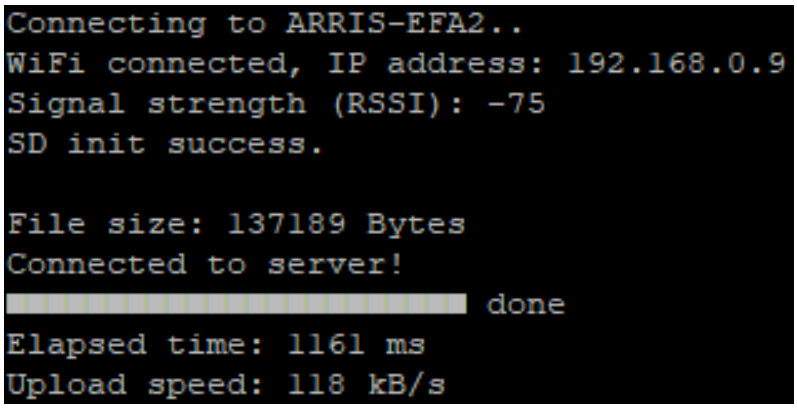


Figura 4.7: Resultado del programa transmisor en monitor serial.

Con un editor hexadecimal se analiza el archivo CSV encriptado el cual se encuentra almacenado en el servidor local. En la figura 4.8 se muestra un fragmento inicial de bytes del archivo encriptado. En la parte derecha donde se muestra la representación en código ASCII, se puede apreciar que los símbolos son completamente diferentes a los presentados en el archivo original (ver figura 4.6), lo cual implica que el sistema realmente realiza un cambio en la información del archivo.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	13	4C	4D	A6	70	9B	1B	72	2F	4C	43	7A	03	FA	16	6F	.LM¦p›.r/LCz.ú.o
00000010	B9	03	98	15	68	2C	11	75	63	E8	1C	8F	93	04	1A	6F	¹.˜.h,.ucè..“..o
00000020	55	14	88	7B	A4	1F	09	77	78	78	D6	D0	7E	F1	1F	73	U.ˆ{¤..wxxÖÐ~ñ.s
00000030	D3	1C	29	FE	BA	9A	EF	49	C9	38	0A	5B	07	D4	1F	74	Ó.)þºšïIÉ8.[.Ô.t
00000040	77	91	24	4C	5E	29	03	6F	DC	B3	52	F0	8C	B3	20	70	w‘$L^).oÜ³RðŒ³ p
00000050	D0	02	F9	71	D1	E3	0E	6F	FB	17	62	D0	34	6E	23	73	Ð.ùqÑã.oû.bÐ4n#s
00000060	57	77	9B	CB	43	C6	09	72	C4	C1	C6	3E	C5	5A	F5	49	Ww›ËCÆ.rÄÁÆ>ÅZõI
00000070	D6	DB	83	3D	1E	E8	1D	73	CA	7F	02	99	7B	94	16	6F	ÖÛƒ=.è.sÊ..™{”.o
00000080	D4	63	47	23	D7	0F	19	75	CC	4B	53	42	2B	85	1A	6F	ÔcG#×..uÌKSB+….o
00000090	52	F2	B1	37	2D	F3	07	76	D5	A3	DE	FF	F9	83	21	74	Rò±7-ó.vÕ£Þÿùƒ!t

Figura 4.8: Fragmento del archivo ecg.csv en un editor hexadecimal, encriptado con un enmascaramiento aditivo a nivel byte.

Una vez que se verifican las características del archivo encriptado, se realiza una prueba con el sistema receptor. La figura 4.9 muestra la información obtenida a través de la comunicación serial con el dispositivo, se puede apreciar información relevante de la conexión realizada con el servidor, y el tiempo transcurrido entre el solicitud y la

finalización de la descarga y desencriptación del archivo.

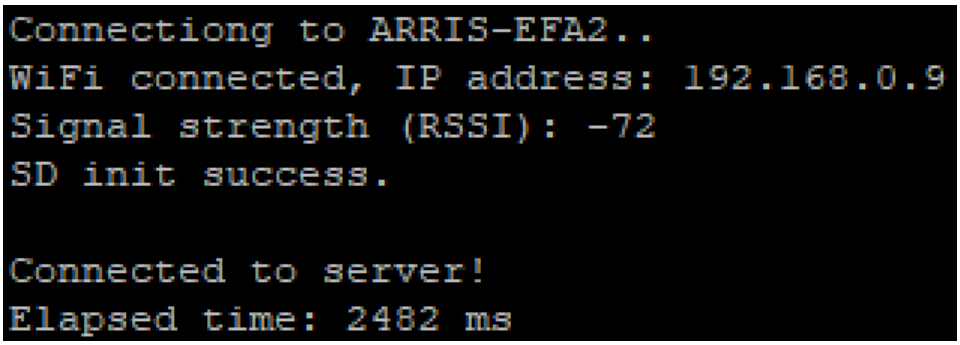


Figura 4.9: Resultado del programa receptor en monitor serial.

Con un editor hexadecimal se analiza el archivo CSV desencriptado el cual se encuentra almacenado en la memoria micro-SD del receptor. En la figura 4.10 se muestra un fragmento inicial de bytes del archivo desencriptado. En la parte derecha donde se muestra la representación en código ASCII, se puede apreciar que los símbolos son idénticos a los presentados en el archivo original (ver figura 4.6), lo cual implica que el sistema realmente recupera el archivo original.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	2D	30	2E	30	36	35	36	33	30	36	38	32	0D	0A	2D	30	-0.065630682..-0
00000010	2E	30	36	35	36	33	30	36	38	32	0D	0A	2D	30	2E	30	.065630682..-0.0
00000020	36	34	35	34	38	38	35	38	0D	0A	2D	30	2E	30	36	34	64548858..-0.064
00000030	39	30	39	34	36	36	0D	0A	2D	30	2E	30	36	36	33	35	909466..-0.06635
00000040	31	38	39	38	0D	0A	2D	30	2E	30	36	38	38	37	36	31	1898..-0.0688761
00000050	35	35	0D	0A	2D	30	2E	30	36	39	32	33	36	37	36	34	55..-0.069236764
00000060	0D	0A	2D	30	2E	30	37	33	39	32	34	36	36	39	0D	0A	..-0.073924669..
00000070	2D	30	2E	30	37	32	38	34	32	38	34	35	0D	0A	2D	30	-0.072842845..-0
00000080	2E	30	36	33	38	32	37	36	34	31	0D	0A	2D	30	2E	30	.063827641..-0.0
00000090	36	36	37	31	32	35	30	37	0D	0A	2D	30	2E	30	36	35	66712507..-0.065

Figura 4.10: Fragmento del archivo ecg.csv en un editor hexadecimal, recuperado realizando un desencriptado caótico.

En la figura 4.11 se puede observar la señal ECG recuperada la cual se tiene como un archivo tipo CSV.

La figura 4.12 muestra el error entre la señal ECG obtenida del archivo recuperado y la señal ECG del archivo original, se puede observar que el error es de 0, lo cual demuestra que el archivo original y el recuperado son idénticos, esto implica que la transmisión segura de datos fue exitosa.

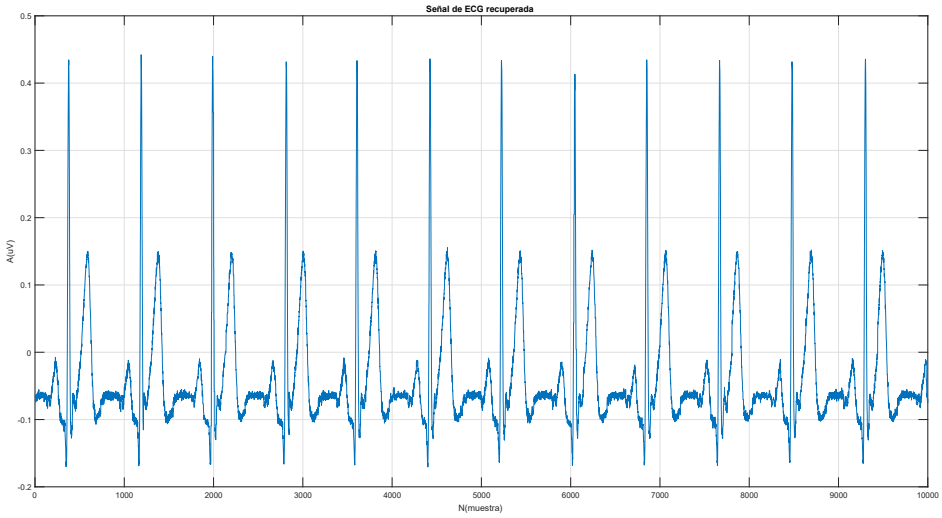


Figura 4.11: Señal de ECG recuperada.

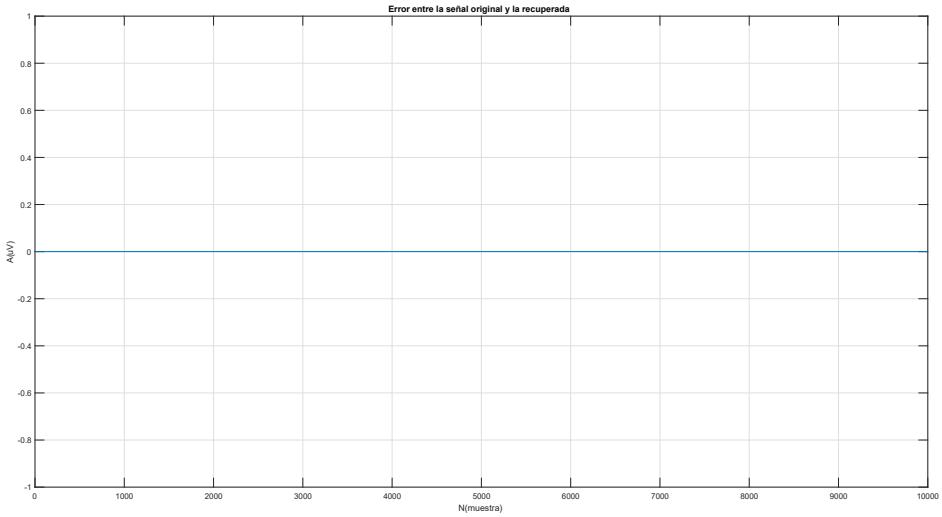


Figura 4.12: Error entre la señal ECG original y la recuperada.

4.5.2. Encriptación de archivo de imagen

Para el experimento con el archivo de imagen se realiza una prueba con el transmisor donde se utiliza como información a encriptar un archivo de imagen que representa un imagen de resonancia magentica [28].

En la figura 4.13 se puede observar la imagen de resonancia magnética la cual se tiene como un archivo tipo ppm.

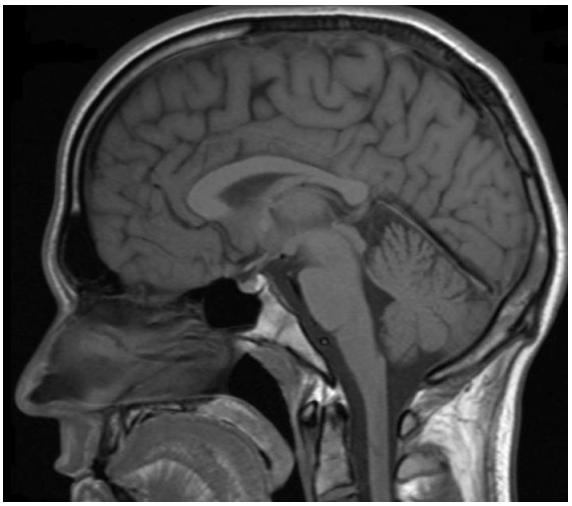


Figura 4.13: Imagen de resonancia magnética a encriptar [28].

Una vez que se verifican las características del archivo, este se almacena en una memoria micro-SD y se introduce al sistema transmisor para realizar una prueba. La figura 4.14 muestra la información obtenida a través de la comunicación serial con el dispositivo, se puede apreciar información relevante de la conexión realizada con el servidor, características del archivo a encriptar y velocidad de transmisión.

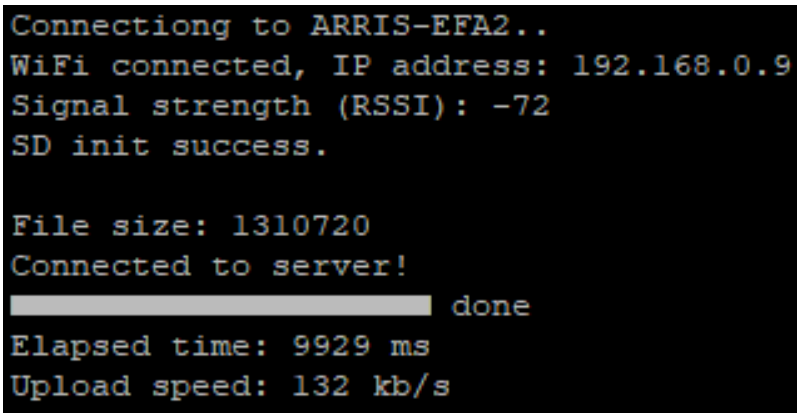


Figura 4.14: Resultado del programa transmisor en monitor serial.

La figura 4.15 muestra el archivo ppm encriptado el cual se encuentra almacenado

en el servidor local. A pesar de que se puede notar el contorno presente en la imagen original la mayoría de la información importante de la imagen se encuentra cubierta por ruido aparentemente aleatorio.

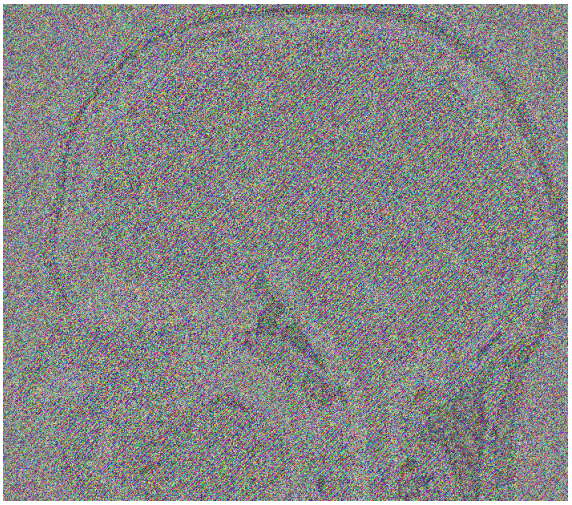


Figura 4.15: Imagen de resonancia magnética encriptada.

Una vez que se verifican las características del archivo encriptado, se realiza una prueba con el sistema receptor. La figura 4.16 muestra la información obtenida a través de la comunicación serial con el dispositivo, se puede apreciar información relevante de la conexión realizada con el servidor, y el tiempo transcurrido entre el solicitud y la finalización de la descarga y desencriptación del archivo.

```
Connectiong to ARRIS-EFA2..
WiFi connected, IP address: 192.168.0.9
Signal strength (RSSI): -72
SD init success.

Connected to server!
Elapsed time: 19324 ms
```

Figura 4.16: Resultado del programa receptor en monitor serial.

La figura 4.17 se muestra la imagen recuperada por el bloque receptor, es decir, del archivo desencriptado. Para comprobar que el archivo original y el archivo recuperado son iguales se realiza una prueba utilizando un algoritmo de comparación byte por byte para comprobar si existe un error entre los dos archivos.

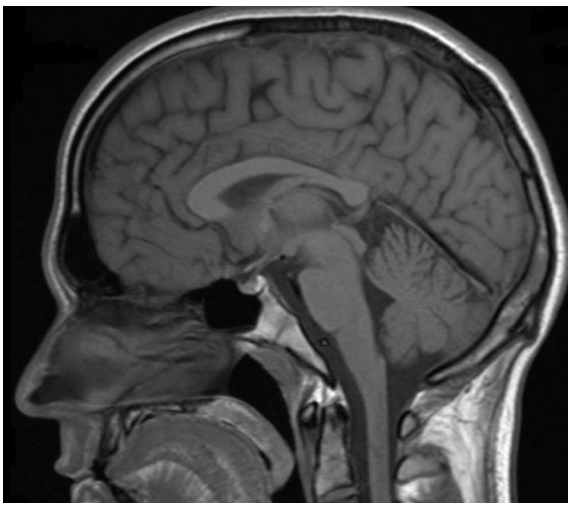


Figura 4.17: Imagen de resonancia magnética recuperada.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

En este trabajo de tesis de licenciatura, se diseñó e implementó un esquema de transmisión de datos médicos a través de redes inalámbricas en sistemas embebidos de bajo costo. Para implementar el caos en el esquema de encriptacion propuesto, se analizaron tres mapas caóticos cuyas ecuaciones eran lo suficientemente sencillas para implementarse en dispositivos de bajo costo. También, se implementó la arquitectura de un servidor local sencillo en un sistema Raspberry Pi.

El esquema propuesto, se basó en un esquema de cifrado de clave simétrica, donde la clave compartida entre los usuarios pertenece a las características del mapa caótico utilizado. El algoritmo criptográfico se basó en una implementación del encriptamiento aditivo y el cifrado de flujo debido a la simplicidad de operaciones que lo conforman, lo cual fue útil tomando en cuenta los recursos limitados que presentaba el microcontrolador utilizado.

El esquema desarrollado puede ser implementado en hospitales, hogares o en otros lugares donde se requiera proteger información sensible dentro de una red local.

5.2. Trabajo a futuro

Como trabajo a futuro se plantean las siguientes actividades:

- **Implementar el esquema en un servidor en red**: Lo cual permita transmitir la información a lugares distantes, aplicando el concepto de telemdicina.
- **Diseñar un sistema dedicado**: Instrumentar el modulo ESP-012 de acuerdo a las necesidades especificas, prescindiendo así de elementos no necesarios que están presentes en la tarjeta de desarrollo NodeMCU, lo cual lograría reducir el costo del sistema.

- **Realizar pruebas incrementando el numero de sistemas transmisor/receptor**: Lo cual permita observar el desempeño del sistema para condiciones de uso real (en un establecimiento medico).
- **Implementar algoritmo de cifrado mas robusto**: El algoritmo de encriptamiento propuesto en este trabajo de tesis, es un algoritmo muy sencillo basado en cifrado de flujo y encriptamiento aditivo, ademas, no se realizaron pruebas de seguridad para verificar la confiabilidad de este.
- **Implementar un mapa caótico con mejores caracteristicas**: Investigar en la literatura mapas caoticos que presenten mejores caracteristicas dinamicas.

  Es importante que los últimos dos puntos solo se pueden lograr sacrificando el desempeño del sistema, debido a que implicaría que se incrementara el tiempo de procesamiento. Sin embargo, es importante tomar en cuenta cuales son las restricciones de diseño del sistema, si el tiempo no es vital para la aplicación deseada, entonces aumentar las características de seguridad del esquema de transmisión es una beneficio importante.

Bibliografía

[1] Ming Li, Worcester Polytechnic Institute (2010). Data Security and Privacy in Wirless Body Area Networks *IEEE Wireless Communications*, pp. 51-58.

[2] HIPPA Journal (2018). *Analysis of February 2018 Healthcare Data Breaches.*

[3] Beazley(2017). Beazley Breach Insights, *Healthcare Special Edition*, pp 2-5.

[4] Heath, Steve (2003). Embedded systems design. *EDN series for design engineers. Newnes.* p. 2.

[5] INCIBE (2018). Introducción a los sistemas embebidos. *CERT de seguridad e industria.*

[6] Espressif Systems (2017). *ESP8266 overview.* https://www.espressif.com/en/products/hardware/esp8266ex/overview.

[7] ESP8266 Community (2015). *ESP8266* module family. https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

[8] ESP8266EX (2012). ESP8266EX Espressif Systems datasheet rev 5.8, p. 2.

[9] Alasdair Allan (2015). Getting Started with the ESP8266. https://medium.com/@aallan/getting-started-with-the-esp8266-270e30feb4d1

[10] ESP8266 Community (2014). Arduino core for ESP8266 WiFi chip. https://github.com/esp8266/

[11] Michel Yuan (2017). *Getting to know NodeMCU and its DEVKIT board.*

[12] Ivan Grokhotkov (2017). *ESP8266 library documentation.* https://arduino-esp8266.readthedocs.io/en/latest/libraries.html

[13] Rabbit and Dynamic (2007) *An Introduction to Wi-Fi*, , pp. 20-22.

[14] Jack Wallen (2010). *Easy LAMP Server Installation.*

[15] Raspberry Pi Foundation (2014). *"What is a Raspberry Pi".* https://www.raspberrypi.org/help/faqs/

[16] Mike Thompson (2012). *What is Raspbian.* https://www.raspbian.org/RaspbianFAQ

[17] The MagPi Magazine (2016). *Raspberry Pi 3 is out now! Specs, Benchmarks & more"*. https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[18] PC World (2017). *"The $10 Raspberry Pi Zero W brings Wi-Fi and Bluetooth to minuscule micro"*.

[19] RS componentes (2017). *Raspberry Pi2, Model B Technical Specifications"*.

[20] Mohammed Alsaedi (2010). *Colored Image Encryption and Decryption Using Chaotic Lorenz System and DCT.*

[21] Wenjing Lou, Worcester Polytechnic Institute (2010). Data Security and Privacy in Wirless Body Area Networks *IEEE Wireless Communications*, pp. 51-58.

[22] Daniel-Ioan Curiac, Daniel Iercan (2007). *Chaos-Base Cryptography: End of the Road?*

[23] Bellare, Mihir; Rogaway, Phillip (2005). *Introduction to Modern Cryptography.* pp. 3-5.

[24] Gary C. Kessler (2013). *An Overview of Cryptography.*

[25] Robert Bishop (2008). Stanford Encyclopedia of Philsofy: *Chaos.*

[26] Wadia Faid Hassan (2012). *Dynamical Properties of the Hénon Map.*

[27] Daan van den Berg (2005). *Discrete dynamical systems and the logistic map.*

[28] Reznek R, Takahashi S (2006). Commonly used imaging techniques for diagnosis and staging. *Journal of Clinical Oncology.* 2006 Jul 10;24(20):3234-44

Apéndice A

Programa para bloque transmisor

En este apéndice, se presenta el software de encriptacion propuesto para el esquema de este trabajo de tesis. Se realizo en lenguaje C con el software Arduino IDE para el sistema NodeMCU basandose en el "Arduino Core for ESP8266".

```
1  #include <ESP8266WiFi.h>
2  #include <SD.h>
3
4  #define MTU_Size  2*1460
5
6  const char *ssid     = "************";
7  const char *password = "********************";
8  const char *servername = "***********";
9
10 unsigned long currentTime;
11
12 byte clientBuf[MTU_Size];
13 int clientCount, count, chunks;
14 char symbol[] = {0xE2, 0x96, 0xA0};
15
16 char fileName[13] = "ecg.csv";
17 long fileSize;
18
19 //Condiciones iniciales
20 double seq;
21 double xn = 0.683928;
22 double r = 3.65938;
23
24 File medFile;
25 WiFiClient client;
26
27 void setup() {
28   Serial.begin(230400);
```

```
29    Serial.println();
30    delay(5000);
31    Serial.print("Connecting␣to␣");
32    Serial.print(ssid);
33
34    WiFi.begin(ssid, password);
35
36    while(WiFi.status() != WL_CONNECTED)
37    {
38      delay(100);
39      Serial.print(".");
40    }
41
42    Serial.println();
43    Serial.print("WiFi␣connected,␣");
44    Serial.print("IP␣address:␣");
45    Serial.println(WiFi.localIP());
46
47    long rssi = WiFi.RSSI();
48    Serial.print("Signal␣strength␣(RSSI):␣");
49    Serial.println(rssi);
50
51    if(SD.begin(SS,50000000) == true)
52    {
53      Serial.println("SD␣init␣success.");
54    }
55    else
56    {
57      Serial.println("SD␣init␣failure.");
58    }
59    Serial.println();
60
61    pinMode(LED_BUILTIN, OUTPUT);
62    led_blink();
63  }
64
65  void loop() {
66    currentTime = millis();
67    medFile = SD.open(fileName, FILE_READ);
68    fileSize = medFile.size();
69    chunks = fileSize/(16*MTU_Size);
70    count = 0;
71
72    Serial.print("File␣size:␣");
73    Serial.print(fileSize);
```

```
74  Serial.println("␣Bytes");
75
76  if(client.connect(servername, 80))
77  {
78    client.setNoDelay(true);
79    Serial.println("Connected␣to␣server!");
80
81    //HTTP POST request:
82    client.println("POST␣/upload.php?up=1&ACM=1␣HTTP/1.1");
83    client.print("Host:␣");
84    client.println(servername);
85    client.println("Connection:␣close");
86    client.println("Content-Type:␣multipart/form-data;␣
        boundary=--84989444e2484915a216e1718e0f93f0");
87    client.print("Content-Length:␣");client.println(medFile.
        size()+188);
88    client.println();
89
90    client.println("----84989444e2484915a216e1718e0f93f0");
91    client.println("Content-Disposition:␣form-data;␣name=\"
        FileGDF\";␣filename=\"ecg.csv\"");
92    client.println("Content-Type:␣application/octet-stream");
93    client.println();
94
95    while(medFile.available())
96    {
97      if(medFile.available() >= MTU_Size)
98      {
99        clientCount = MTU_Size;
100     }
101     else
102     {
103       clientCount = medFile.available();
104     }
105
106     medFile.read(&clientBuf[0],clientCount);
107
108     for(int i = 0; i < MTU_Size/8; i++)
109     {
110       seq = r*xn*(1-xn);
111       xn = seq;
112       byte * b = (byte *) &seq;
113
114       for(int j = 0; j < 8; j++)
115       {
```

```
116             clientBuf[(8*i+j)] += b[j];
117           }
118         }
119
120         client.write((const uint8_t *)&clientBuf[0],
              clientCount);
121         count++;
122         if(count == chunks)
123         {
124           //Serial.print(".");
125           Serial.write(symbol);
126           count = 0;
127         }
128       }
129       client.println();
130       client.println("----84989444e2484915a216e1718e0f93f0--");
            //form end
131       client.println();
132       medFile.close();
133       Serial.println("␣done");
134     }
135
136     currentTime = millis() - currentTime;
137     Serial.print("Elapsed␣time:␣");
138     Serial.print(currentTime);
139     Serial.println("␣ms");
140
141     Serial.print("Upload␣speed:␣");
142     Serial.print(fileSize/currentTime);
143     Serial.println("␣kB/s");
144     Serial.println();
145
146     serverResponse();
147
148     client.stop();
149
150     led_blink();
151
152     delay(100000);
153 }
154
155 void serverResponse()
156 {
157     while(client.available())
158     {
```

```
159     char c = client.read();
160     //Serial.print(c);
161   }
162 }
163
164 void led_blink()
165 {
166   boolean state = false;
167   for(int i = 0; i < 4; i++)
168   {
169     digitalWrite(LED_BUILTIN, state);
170     state = !state;
171     delay(500);
172   }
173 }
```

Apéndice B

Programa para bloque receptor

En este apéndice, se presenta el software de desencriptacion propuesto para el esquema de este trabajo de tesis. Se realizo en lenguaje C con el software Arduino IDE para el sistema NodeMCU basandose en el "Arduino Core for ESP8266".

```
1 #include <ESP8266WiFi.h>
2 #include <SD.h>
3
4 #define SD_Buf  512
5
6 const char *ssid     = "************";
7 const char *password = "********************";
8 const char *servername = "***********";
9
10 unsigned long currentTime;
11
12 String url = "/download.php/";
13
14 int clientCount = 0;
15
16 byte clientBuf[SD_Buf];
17
18 char fileName[13] = "uff.csv";
19
20 //Condiciones iniciales
21 double seq;
22 double xn = 0.683928;
23 double r = 3.65938;
24
25 File medFile;
26 WiFiClient client;
27
28 void setup() {
```

```
29   Serial.begin(230400);
30   Serial.println();
31   delay(5000);
32   Serial.print("Connecting to ");
33   Serial.print(ssid);
34
35   WiFi.begin(ssid, password);
36
37   while(WiFi.status() != WL_CONNECTED)
38   {
39     delay(100);
40     Serial.print(".");
41   }
42
43   Serial.println();
44   Serial.print("WiFi connected, ");
45   Serial.print("IP address: ");
46   Serial.println(WiFi.localIP());
47
48   long rssi = WiFi.RSSI();
49   Serial.print("Signal strength (RSSI): ");
50   Serial.println(rssi);
51
52   if(SD.begin(SS,50000000) == true)
53   {
54     Serial.println("SD init success.");
55   }
56   else
57   {
58     Serial.println("SD init failure.");
59   }
60   Serial.println();
61 }
62
63 void loop() {
64   currentTime = millis();
65   medFile = SD.open(fileName, FILE_WRITE);
66
67   if(client.connect(servername, 80))
68   {
69     client.setNoDelay(true);
70     Serial.println("Connected to server!");
71   }
72   //HTTP GET request
```

```
73  int bytesSent = client.print(String("GET␣") + url + "␣HTTP
       /1.1\r\n" +
74                  "Host:␣" + servername + "\r\n" +
75                  "Connection:␣close\r\n\r\n");
76
77  while(true) //Header loop
78  {
79    char c = client.read();
80    if(c == '\t')
81    {
82      break;
83    }
84  }
85  unsigned long timeout = 0;
86  while(true)
87  {
88    if(client.available())
89    {
90      clientBuf[clientCount] = client.read();
91      timeout = millis();
92      clientCount++;
93      if(clientCount == SD_Buf)
94      {
95        for(int i = 0; i < SD_Buf/8; i++)
96        {
97          seq = r*xn*(1-xn);
98          xn = seq;
99          byte * b = (byte *) &seq;
100         for(int j = 0; j < 8; j++)
101         {
102           clientBuf[(8*i+j)] -=  b[j];
103         }
104       }
105       clientCount = 0;
106       medFile.write((const uint8_t *)&clientBuf[0], SD_Buf)
             ;
107     }
108   }
109   else
110   {
111     if((millis() - timeout) >= 1000)
112     {
113       if(clientCount > 0)
114       {
115         for(int i = 0; i <= clientCount/8; i++)
```

```
116             {
117               seq = r*xn*(1-xn);
118               xn = seq;
119               byte * b = (byte *) &seq;
120               for(int j = 0; j < 8; j++)
121               {
122                 clientBuf[(8*i+j)] -=  b[j];
123               }
124             }
125             medFile.write((const uint8_t *)&clientBuf[0],
                   clientCount);
126           }
127           break;
128         }
129       }
130     }
131     medFile.close();
132     client.stop();
133     currentTime = millis() - currentTime;
134
135     Serial.print("Elapsed time: ");
136     Serial.print(currentTime);
137     Serial.println(" ms");
138     Serial.println();
139
140     delay(100000);
141 }
```

Apéndice C

Programas para el manejo de archivos del servidor

En esta apéndice, se presenta el software del lado de servidor el cual ayuda al Raspberry Pi a responder las solicitudes realizadas por los clientes transmisor y receptor.

```php
<?php
$target_dir = "uploads/";
$filename = basename(@$_FILES["FileGDF"]["name"]);
$target_file = $target_dir . $filename;
$FileType = pathinfo($target_file,PATHINFO_EXTENSION);

if(isset($_GET["up"])&&(isset($_GET["ACM"]))) {
  if (move_uploaded_file(@$_FILES["FileGDF"]["tmp_name"], $target_file))
  {
    echo "The file ". basename($filename). " has been uploaded.";
  }
  else
  {
    echo "Sorry, there was an error uploading your file.";
  }
}
?>
```

```php
<?php
$target_dir = "uploads/";
$filename = 'ecg.csv';
$target_file = $target_dir . $filename;


if (file_exists($target_file)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($target_file).'"');
    header('Content-Length: ' . filesize($target_file));
    echo("\t");
    readfile($target_file);
}
?>
```