Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Sistema de criptografía y esteganografía para la confidencialidad de información en aplicaciones de transmisión para dispositivos de IdC

Tesis que presenta

Gerardo Alejandro Ruiz Avendaño

para obtener el Grado de

Maestro en Ciencias en Computación

Directores de la Tesis

Dra. Brisbane Ovilla Martínez
Dr. Amilcar Meneses Viveros

Ciudad de México **Agosto 2022**

Resumen

Internet de las Cosas (IdC) ha revolucionado la forma en cómo se realizan actividades en distintos ámbitos, desde el cotidiano hasta el industrial, por ello cada día hay una mayor cantidad de dispositivos interconectados que ayudan a tener un flujo de información que contribuyen en la realización de dichas tareas. Sin embargo, con la creciente tasa de información también se enfrentan vulnerabilidades para proteger los datos y evitar hacer mal uso de ellos, por tal motivo se busca implementar técnicas de seguridad como criptografía ligera y esteganografía de red. La primera, permite cifrar los datos para que no sean posibles de interpretar por entidades no autorizadas, y con la importante característica de ser algoritmos enfocados especialmente para dispositivos con recursos limitados. La esteganografía de red oculta la información sensible que se quiere transferir con la ayuda de protocolos de comunicación usados en IdC. En el presente trabajo se propone adoptar dos sistemas utilizando criptografía y esteganografía para dar confidencialidad y ocultamiento a la información que se quiere transmitir entre dispositivos de IdC. El primero enfocado en cifrar y ocultar la información que se quiere compartir usando un campo del protocolo de comunicación MQTT. El segundo sistema cifra la información a compartir y oculta la llave de cifrado para ser compartida de manera segura. Se ha implementado con ayuda de la tecnología FPGA el cifrador SIMON para observar a nivel de hardware su comportamiento. Y a nivel de software se utiliza el cifrador SPECK. Un broker es un dispositivo que se encarga de aceptar conexiones de red de clientes, aceptar mensajes publicados y procesar solicitudes de suscripción. En este trabajo se utilizan dos tipos de *brokers*, uno instalado en una computadora de escritorio y otro *broker* público en la nube.

Abstract

Internet of Things (IoT) has revolutionized the way in which activities are carried out in different areas, from daily to industrial, for this reason every day there is a greater number of interconnected devices that help to have a flow of information that contributes in carrying out these tasks. However, with the growing rate of information, vulnerabilities are also faced to protect the data and avoid misusing it, for this reason it's sought to implement security techniques such as light cryptography and network steganography. The first allows data is encrypted so that they are not possible to interpret by unauthorized entities, and with the important characteristic of being algorithms focused especially on devices with limited resources. Network steganography hides sensitive information that is transferred with the help of communication protocols used in IoT. In the present work, it's proposed to adopt two systems using cryptography and steganography to give confidentiality and concealment to the information that is transmitted between IdC devices. The first focused on encrypting and hiding the information to be shared using a field of the MQTT communication protocol. The second system encrypts the information is shared and hides the encryption key to be shared safely. The SIMON cipher has been implemented with the help of FPGA technology to observe its behavior at the hardware level. And at the software level, the SPECK cipher is used.A broker is a device that is responsible for accepting client network connections, accepting published messages, and processing subscription requests. Two types of *brokers* are used to implement the systems, one installed on a desktop computer and another public *broker* in the cloud.

Agradecimientos

Agradezco al CONACYT por el apoyo económico que me brindo para el estudio de esta maestría y al Cinvestav por aceptarme en su programa de maestría y por los conocimientos adquiridos en estos dos años.

A los doctores que me impartieron clases durante la maestría, especialmente a mis directores de tesis la Dra. Brisbane Ovilla Martínez y al Dr. Amilcar Meneses Viveros por su apoyo en el desarrollo de esta tesis, y a mis sinodales el Dr. Luis Gerardo de la Fraga y el Dr. Cuauhtemoc Mancillas López, por su tiempo y comentarios.

A mi madre Rosa y mis hermanos Alvaro y Esmeralda por apoyarme toda mi vida. Y a mis amigos que me acompañaron en estos años.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años Internet de las Cosas (IdC) ha aumentado su presencia en el uso cotidiano e industrial. Esto facilita, por ejemplo, procesos de fabricación de productos, que se realizan dado a la autonomía que se le brinda a los dispositivos y actuadores. Los dispositivos que se utilizan en IdC deben ser pequeños y por este motivo cuentan con recursos limitados. En las aplicaciones desarrolladas para IdC se busca tener un bajo consumo de energía y poca cantidad de almacenamiento. La información que se transmite en ocasiones es sensible, y es necesario establecer métodos de seguridad que brinden a la información confidencialidad e integridad, por mencionar algunos.

Hoy en día se han propuesto diversos protocolos para la comunicación de dispositivos de IdC tales como CoAP, XMPP, MQTT [2], entre otros, para intercambiar información. Uno de los protocolos más utilizado es MQTT, este es un protocolo de transporte de mensajes en una red tipo publicación-suscripción. Es principalmente utilizado en dispositivos que posean recursos limitados. Actualmente se trabaja con las versiones 3.1.1 y 5.0. Sin embargo, este protocolo no ofrece un método de seguridad más allá de la capa de red por estar montado en TCP/IP o SSL. En el artículo [3] se muestra que usando el motor de búsqueda Shodan en un servidor público de MQTT se puede lograr un ataque de denegación de servicios a los clientes conectados, obtener datos de esos clientes o enviar datos incorrectos. En el mismo artículo se analiza el método de autenticación que brinda MQTT, si un atacante se encuentra en la misma red solo necesita rastrear el tráfico de red y encontrar el paquete CONNECT para que pueda revelar el nombre de usuario y contraseña, debido a que el protocolo MQTT no posee un mecanismo de seguridad general. Por esta razón es importante que se brinde seguridad a la información que se transmite y así evitar fugas de información.

Los algoritmos convencionales de criptografía son costosos en términos de cálculos computacionales. Es indispensable optar por soluciones que no afecten el rendimiento del dispositivo de IdC. En esta tesis se abordan dos técnicas para incrementar la seguridad de la información que se desea intercambiar, la primera es la criptografía ligera que brinda confidencialidad y protege la información y la segunda es la esteganografía que oculta dicha información. La esteganografía consiste en utilizar un objeto portador, como puede ser una imagen o video, e insertar en sus datos la información que se quiere ocultar. Este método necesita de una función estego que inserte la in-

formación dentro del objeto portador y una función estego inversa para recuperar la información. La función estego debe proporcionar tres características fundamentales: robustez, imperceptibilidad y capacidad [4].

La esteganografía de red consiste en utilizar al datagrama de un protocolo de comunicación como el objeto portador. En este trabajo de tesis, los datagramas del protocolo MQTT se utilizan como objetos portadores. Los datagramas del protocolo estarán presentes siempre que haya comunicación. Por tal motivo los datagramas de MQTT se seleccionaron como objetos portadores y así aprovechar un recurso ya existente en el dispositivo y evitar el uso de más recursos.

La criptografía ligera [5] es una sub-rama de la criptografía convencional, desarrolla primitivas criptográficas que se adecuen a las necesidades de los dispositivos con recursos limitados. Las primitivas se enfocan en desarrollo de hardware y software. El algoritmo SIMON [6] se enfoca en implementación por hardware y se basa en una red de Feistel con rotaciones a la izquierda y derecha, operaciones binarias AND y XOR. El algoritmo SPECK [6] se enfoca en implementación por software y se basa en rotaciones a la izquierda y derecha, operaciones binarias XOR y una suma algebraica. Estos algoritmos fueron seleccionados debido a los tamaños de bloques pequeños y su bajo requerimiento de recursos computacionales.

1.1. Motivación

Hoy en día IdC ha tenido un gran aumento de uso en los últimos años. Por tal motivo el número de dispositivos en uso aumenta así como el tráfico de información para realizar los procesos. Es necesario proveer de seguridad a la información que sea sensible para así evitar que terceros no autorizados conozcan dicha información.

Los métodos de seguridad convencionales requieren de un poder de cómputo prohibitivo en dispositivos con recursos limitados. Existen métodos que proporcionan seguridad a la información sin utilizar un alto procesamiento, por ejemplo, la esteganografía. La esteganografía oculta información dentro de un objeto portador que puede ser una imagen, un audio, un video o un protocolo de comunicación. Los protocolos de comunicación como HTTP/HTTPS y UDP tienen un alto peso además que tienen un diseño complejo en su encabezado, por lo tanto, para dispositivos restringidos se desarrollan protocolos que sea ligeros y pocos complejos y así no utilizar más recursos de los necesarios.

En IdC se utilizan diferentes protocolos de comunicación, como son CoAP, MQTT, y XMPP, entre otros. Su principal característica es su ligereza y sencillez en su datagrama. El protocolo MQTT es el más utilizado en IdC para el intercambio de información, sin embargo, al poseer las características mencionadas anteriormente su seguridad es poco eficaz, teniendo solo usuarios y contraseñas como método de seguridad. La seguridad es de vital importancia en el intercambio de la información, por lo que se necesita agregar seguridad, pero utilizando recursos ya existentes en el dispositivo sin agregar elementos que afecten el rendimiento.

La criptografía ligera es otro método que proporciona confidencialidad a la infor-

mación, los algoritmos desarrollados se adecuan a las limitaciones de los dispositivos con recursos limitados, no utilizando mucho procesamiento ni una gran cantidad de almacenamiento.

1.2. Planteamiento del problema

IdC es una red de vehículos, dispositivos físicos, software y elementos electrónicos todos conectados para facilitar el intercambio de datos. El propósito de IdC es proporcionar la infraestructura de información tecnológica para el intercambio de "cosas". Las limitaciones de los dispositivos IdC incluyen el presupuesto energético, la conectividad y la capacidad computacional [7].

A pesar de sus limitaciones, IdC hoy en día es una tecnología capaz de implementar un amplio rango de aplicaciones. Debido al tipo de datos que manejan muchas de las aplicaciones de IdC, necesitan implementar mecanismos que protejan los datos en distintos niveles. Por ejemplo, ocultar su transmisión, o proteger su confidencialidad, por mencionar algunos. No obstante, se ha prestado poca atención a la seguridad, ya que las técnicas y algoritmos actuales para proveer seguridad no fueron diseñados para ser implementados en dispositivos con tantas limitaciones de hardware, y se debe considerar que en IdC se prioriza el consumo de energía.

Se puede plantear la siguiente hipótesis: *Existe un sistema basado en criptografía y esteganografía con características y usos de recursos adecuado para proveer confidencialidad y ocultamiento en aplicaciones de comunicación de datos utilizando un campo del protocolo de comunicación MQTT en dispositivos de IdC.*

Se plantea desarrollar un sistema que involucre esteganografía y criptografía ligera utilizando un campo del protocolo MQTT. Se debe desarrollar una función esteganográfica que permita ocultar la información dentro del campo del protocolo MQTT y poder ser transmitida por la red que MQTT maneja sin ser percibida por un usuario ajeno. Los algoritmos de criptografía ligera que se seleccionaron para utilizar son el SIMON y SPECK. El algoritmo SIMON está diseñado para una implementación en hardware, mientras que el algoritmo SPECK está diseñado para implementarse en software.

1.3. Propuesta de solución

En este trabajo se plantea diseñar un sistema que permita la ocultación de la información y al mismo tiempo provea confidencialidad en el intercambio de información entre dispositivos de IdC, desarrollando funciones esteganográficas que utilizan como objeto portador el protocolo de comunicación MQTT, y previamente la información es cifrada mediante los algoritmos SPECK, en software, y SIMON, en hardware.

1.4. Objetivos

General

Obtener un sistema de bajo consumo de recursos para el ocultamiento y cifrado de información transmitida entre dispositivos de IdC a través del protocolo de comunicación MQTT.

Particulares

1. Ocultar información en uno de los campos del protocolo MQTT.

2. Proveer confidencialidad a la información mediante un algoritmo criptográfico adecuado para IdC.

3. Desarrollar funciones esteganográficas para el ocultamiento de la información.

4. Validar la solución propuesta en torno a las limitaciones de los dispositivos de IdC, considerando sus capacidades de hardware y software.

1.5. Organización de la tesis

En el capítulo 2 se presenta los preliminares sobre IdC, su definición, su arquitectura y los protocolos que se manejan es esta red de dispositivos. Se analiza el protocolo MQTT v5.0, la arquitectura en la que trabaja, la forma del datagrama, los tipos de paquetes de control que maneja, la seguridad en la que trabaja y cómo funciona el intercambio de información.

En el capítulo 3 se describe el estado del arte, que presenta los trabajos relacionados con seguridad en IoT, uso de esteganografía en un ámbito en general, el uso de esteganografía desarrollado en un FPGA, el uso de esteganografía usando como objeto portador un protocolo de comunicación con un enfoque general, uso de esteganografía en general con un enfoque a IdC y los trabajos de esteganografía usando protocolos de comunicación con un enfoque en IdC.

En el capítulo 4 se describen los dos esquemas que se manejan en la tesis, ocultando la información que se va a compartir y ocultando la llave que se utilizó para cifrar la información. Además, se describen dos funciones esteganográficas que se diseñaron e implementaron para ocultar la información en el protocolo MQTT.

En el capítulo 5 se describe como se implementó el sistema, mencionando los dispositivos y software que se utilizaron para el desarrollo de las aplicaciones que se necesitan para el intercambio de información. Se describe el funcionamiento del cliente-publicador que oculta y cifra la información para posteriormente ser enviada por el protocolo MQTT, también se describe el funcionamiento del cliente-suscriptor que obtiene la información oculta y la descifra para que se pueda visualizar dicha

información. Se describen los descifradores SIMON y SPECK que se implementaron en software y hardware.

En el capítulo 6 se muestran los resultados obtenidos durante las pruebas de la implementación de los clientes en software y hardware.

En el capítulo 7 se presentan las conclusiones y trabajo a futuro.

Capítulo 2

Preliminares

2.1. Internet de las Cosas

En los últimos años internet ha sido una gran herramienta para el desarrollo tecnológico ya sea en cuestiones de entretenimiento, laboral o industrial. Cada año se busca incrementar la capacidad de datos que se pueden intercambiar, lo cual se ha logrado gracias al aumento en la interconexión de pequeños dispositivos, a lo que se le ha denominado como Internet de las Cosas (IdC) [8]. IdC es una red de vehículos, dispositivos físicos, software y elementos electrónicos conectados para facilitar el intercambio de información. Su propósito es proveer la infraestructura de información tecnológica para el intercambio de "cosas" [7]. Este modelo de red logra que dispositivos con recursos limitados y que poseen una capacidad de comunicación puedan ser conectados a Internet para poder trabajar en ese entorno [8].

IdC permite que los dispositivos logren adquirir cierta inteligencia ya que pueden llevar a cabo varias operaciones y de esas operaciones tomar ciertas acciones que son necesarias para realizar el trabajo con base en la información que se recopila del entorno. El sistema de IdC ha sido preparado para soportar el aumento en el intercambio de datos, los recursos informáticos que se necesiten y las infraestructuras de la red. IdC puede ser visto como el sistema nervioso y las decisiones que se deben tomar son proporcionadas por las diversas tecnologías que se pueden encontrar, tales como computación en la nube, computación paralela, el análisis de big data, inteligencia artificial, por mencionar algunas. Una perfecta combinación de estas tecnologías nos asegura un sistema capaz de efectuar cualquier trabajo.

Para tener una conectividad confiable para una gran cantidad de dispositivos conectados y en comunicación, IdC debe tener una arquitectura de capas que permita ser flexible. A pesar de numerosos trabajos que hay relacionados a la arquitectura de IdC, no se ha establecido un modelo de referencia por lo que el modelo actual se basa en el estándar del modelo OSI (por sus siglas en ingles *Open Systems Interconnection*), con modificaciones en las capas de enlace de datos, red y transporte. IdC trabaja con el modelo de tres capas, que son percepción, red y aplicación, como se muestra en la figura 2.1a.

(a) Arquitectura de 3 capas. (b) Arquitectura de 5 capas.

Figura 2.1: Arquitecturas para el sistema de IdC

En la capa de percepción se efectúa la interacción de los objetos y componentes físicos, es decir, se encarga de adquirir, procesar y transmitir hacia otras capas la información de los dispositivos conectados. La capa de red nos brinda el enrutamiento y la transmisión de los datos teniendo como recursos para lograr esta transmisión los dispositivos de conexión, tales como concentradores o enrutadores, las distintas redes de comunicación como Bluetooth, WiFi, entre otras, y los protocolos de comunicación como el IEEE 802.15.4, Zigbee, 6LoWPAN, etc. La capa de aplicación es la que se encarga de las operaciones, todo esto en relación a los datos que fueron analizados y procesados.

Otra variante de la arquitectura es el modelo de cinco capas (ver figura 2.1b), la cual consta de agregar dos capas más, la capa de middleware y la capa de gestión. La capa de middleware se encarga de permitir la gestión de servicios, es decir, recibe la información de la capa de red, procesa y realiza cálculos con la información y permite el enlace a una base de datos. La capa de gestión se encarga, como su nombre lo indica, de gestionar el sistema de IdC donde determina la forma de lanzar y cargar las aplicaciones que se usarán.

2.2. Vulnerabilidad en IdC

Debido a las capacidades limitadas de los dispositivos de IdC, muchos de ellos tienen vulnerabilidades que los hacen propensos a varios ataques. Un dispositivo de IdC vulnerable puede ser un riesgo en cualquier red, independiente a su nivel de seguridad [9]. Muchos ataques han implicado aprovechar las vulnerabilidades de los dispositivos de IdC, incluidas acciones como ataques de repetición, ataque de día cero y ataques de suplantación de identidad. Se ha observado un aumento en los ataques de *botnets*. La *botnets* Mirai es un ejemplo conocido, ataca dispositivos explotando las credenciales predeterminadas [10, 11, 12].

Una gran cantidad de dispositivos de IdC se corrompieron y se utilizaron para lanzar ataques de denegación de servicio (DoS) en servidores críticos. Estos ataques utilizan el servicio de nombres de dominio (DNS) y el protocolo de tiempo de red (NTP) como una forma de ataque DoS distribuido (DDoS). Un estudio informó que la razón principal por la que la *botnets* Mirai es tan efectiva es el uso de dispositivos de Idc de bajo costo y fáciles de instalar, desarrollados con poca o ninguna preocupación por la seguridad [13, 12].

En IdC hay dos tipos de amenazas: amenazas contra IdC y amenazas de IdC.

Amenazas contra IdC: el 21 de octubre de 2016, se implementó un gran ataque DDoS contra los servidores Dyn DNS y cerró muchos servicios web, incluido Twitter. Los causantes explotaron las contraseñas y los nombres de usuario predeterminados de las cámaras web y otros dispositivos de IdC, e instalaron la *botnets* Mirai [14] en los dispositivos comprometidos. La enorme red de *bots* se usó para implementar el ataque DDoS contra los servidores Dyn DNS. Las cámaras IP pueden ser vulnerables mediante ataques de desbordamiento de búfer. Las lámparas Phillips Hue fueron vulneradas a través de su protocolo de enlace ZigBee. Amenazas de IdC: los investigadores también encontraron ataques de secuencias de comandos en sitios cruzados (XSS) que explotaron la aplicación Belkin WeMo y se acceden a los datos y recursos a los que la aplicación puede acceder [15].

Para asegurar la seguridad y privacidad de un sistema de IdC, debemos considerar cinco perspectivas: hardware, sistema operativo/firmware, software, redes, datos generados y mantenidos dentro del sistema. Un sistema de IdC tiene pocos componentes, los cuales deben ser inspeccionados a través de estas cinco perspectivas [15].

Seguridad del hardware: la seguridad del hardware es fundamental cuando los atacantes pueden acceder físicamente a los dispositivos de IdC. Casi todos los dispositivos de IdC tienen vulnerabilidades de hardware que pueden ser explotadas por atacantes, incluidos los puertos de depuración UART/JTAG, múltiples opciones de arranque y memoria flash sin cifrar [16, 17]. A través de las puertas traseras de hardware, ya sea deshabilitado la funcionalidad de verificación o iniciando el sistema a través de una imagen de firmware inyectada [15].

Seguridad y privacidad del sistema operativo (SO)/firmware y software: dadas las funcionalidades a menudo limitadas de un dispositivo de IdC, se puede implementar un sistema operativo confiable [18] en el dispositivo si el costo lo permite. Los proble-

mas de seguridad del software son similares a los sistemas informáticos tradicionales. Las puertas traseras y los pares de claves SSL públicas y privadas se descubren realizando un análisis estático en una gran cantidad de firmwares desempaquetados [19]. Una explotación del desbordamiento de búfer se encuentra analizando el protocolo de administración de red doméstica (HNAP) [20] para que pueda usarse para ejecutar cualquier código en el dispositivo. Un desbordamiento de búfer basado en pila de la biblioteca general glibc [21] se aprovecha para atacar varios concentradores domésticos [22, 15].

Seguridad y privacidad de la red: un sistema de IdC es un sistema en red y todo el sistema debe estar protegido [23, 24]. La comunicación debe cifrarse para evitar la fuga de información confidencial. La autenticación debe implementarse cuidadosamente. En el proceso de emparejamiento, el controlador debe conectarse al dispositivo de IdC para configurarlo. La mayoría de los dispositivos de IdC permiten cualquier controlador en las proximidades para el emparejamiento. Para una implementación a gran escala en un entorno público, cualquier persona con acceso a los dispositivos puede reconfigurar el sistema y puede acceder al sistema. La autenticación debe configurarse de manera adecuada. Un sistema de IdC puede estar compuesto por un gran número de nodos con capacidades de detección y técnicas de seguridad para redes de sensores que se pueden aplicar [25, 26, 15].

El ataque Mirai DDoS [14] fue posible debido a las contraseñas débiles en varios dispositivos de IdC. Rouf et al. [23] explotan el protocolo de comunicación inalámbrica insegura de la lectura automática de contadores. Dhanjani vulnera el sistema de lámparas Phillips Hue y descubre que los mecanismos de autenticación no son sólidos. Molina [27] aprovecha el KNX, un protocolo de comunicación de automatización del hogar estandarizado, y descubre que la falta de autenticación y cifrado permite que un atacante controle de forma remota los electrodomésticos de un hotel. Rahman et al. [28] encuentran las vulnerabilidades del protocolo de comunicación del dispositivo portátil (Fitbit) [15].

Análisis de Big Data: dado que la nube se encuentra entre el controlador y los dispositivos de IdC, puede recopilar todos los datos. Tenemos que cuestionarnos ¿debe la nube saberlo todo y recopilar datos sobre nosotros y nuestras pertenencias? Sin embargo, los grandes datos recopilados por la nube pueden ayudar a vencer los ataques. Por ejemplo, un sistema de detección de intrusos adecuado en la nube puede evitar otra ronda de ataques de Mirai. Dado que las cosas suelen ser muy específicas, la detección de intrusos se puede simplificar [15].

Ye et al. realizaron un estudio de caso sobre la seguridad de August Smart Lock [29], la exposición de la clave de protocolo de enlace del dispositivo y los datos de la cuenta y la información personal del propietario, así como la susceptibilidad a los ataques de denegación de servicio (DoS). En otro estudio, Ly y Jin [30] analizaron el problema de la fuga de información del usuario. Examinaron el firmware de las pulseras tecnológicas, incluidas Nike+ Fuelband, la banda Huawei, la banda Xiaomi Mi y la banda Codoon, y encontraron seguridad insuficiente que provocó la fuga de información del usuario [12].

La seguridad de las cerraduras inteligentes también ha llamado la atención de los investigadores [31, 32]. Algunas de las cerraduras inteligentes bajo escrutinio expusieron información confidencial del usuario, mientras que otras podrían ser controladas por dispositivos no autorizados. Kim et al. [31] sugirió que las cerraduras inteligentes modernas deberían tener los siguientes niveles de control: completo, restringido, parcial y mínimo. Chistiakov et al. [32] desarrollaron un nuevo diseño de seguridad para cerraduras inteligentes utilizando un chip de memoria de solo lectura programable borrable electrónicamente (EEPROM) [12].

El Smart Nest Learning Thermostat es otro dispositivo inteligente para el hogar que ha sido analizado por investigadores. Hernandez et al. [17] probaron el dispositivo iniciando una imagen maliciosa a través de un puerto USB. Oren et al. [33] descubrió ataques a televisores inteligentes que tenían como objetivo los protocolos de comunicación de los dispositivos [12]. La tecnología de hogar inteligente permite el control inalámbrico de puertas, luces y otros electrodomésticos. Denning et al. [34] mencionan que este tipo de electrodomésticos son vulnerables a los ataques debido a la falta de un administrador profesional. Denning et al. [35] y Ur et al. [36] analizaron las políticas de control de acceso y las amenazas asociadas a este tipo de dispositivos [12].

A medida que aumenta la cantidad de dispositivos de IdC implementados en los hogares, el control de estos dispositivos se vuelve cada vez más complicado porque cada dispositivo usa una aplicación móvil separada. Para eso están diseñados SmartThings de Samsung o HomeKit de Apple [12].

El análisis de Samsung SmartThings realizado por Fernandes [37] identificó cuatro posibles ataques que podrían lanzarse contra aplicaciones móviles, la indagación de los códigos PIN de las cerraduras de las puertas, la desactivación de las configuraciones de protección y la generación de alarmas falsas. Gyory y Chauah [38] encontraron errores de seguridad en SmartThings que otorgaban acceso privilegiado al sistema a un tercero [12].

Fernández et al. [39] estudiaron patrones de ataque DoS en redes VoIP. Alghamdi et al. [40] examinaron los inconvenientes de seguridad del Protocolo de aplicación restringida (CoAP), que es una capa de aplicación para dispositivos IdC restringidos [12].

Cyr et al. [41] realizó un análisis de red y un análisis de firmware en relojes inteligentes, al mismo tiempo que verificaban las vulnerabilidades de las aplicaciones móviles. Los autores rastrearon la dirección privada del usuario desde el dispositivo IdC, capturaron el intercambio de claves, aplicaron ingeniería inversa a la aplicación móvil, monitorearon el tráfico entre la aplicación y el servidor Fitbit y usaron el tráfico proxy Transport Layer Security (TLS) para interceptar y extraer datos [12].

2.3. Protocolos de comunicación para dispositivos con recursos limitados

Con el auge de IdC se ha tratado de establecer un estándar en los protocolos que se deben de utilizar, pero a pesar de varios esfuerzos todavía no hay un estándar fijo con el cual trabajar, es por ello que se utilizan diversos protocolos para establecer comunicación entre los dispositivos que conforman el sistema de IdC.

- Protocolo de Aplicación Restringida (CoAP). Es el más utilizado para la capa de aplicación, este protocolo es un subconjunto de las funciones que se establecen en HTTP, con la diferencia que el encabezado es de baja carga y con un análisis reducido en la complejidad, de tal forma que se pueda utilizar en dispositivos con poca capacidad de almacenamiento y una reducida capacidad computacional.

- Transporte de Telemetría de Cola de Mensajes (MQTT). Tiene dos funciones por una parte proporciona la conectividad entre las aplicaciones que se utilizan en el sistema de IdC, y por otro se enfoca en las redes y comunicaciones.

- MQTT-SN. Está diseñado específicamente para redes de sensores, ya que se adapta a la dinámica de la comunicación inalámbrica.

- Extensible de Mensajería y Presencia (XMPP). Su funcionamiento original era ser utilizado en las aplicaciones de chat y posteriormente se utilizó en IdC.

- Cola de Mensajes Avanzado (AMQP). Está dirigido para la capa de aplicación y es abierto para IdC y es funcional en entornos que son orientados a mensajes, algunas características a mencionar de este protocolo son la orientación de los mensajes, las colas y el enrutamiento.

- 6LowPAN. Permite transportar paquetes de IPv6 por medio de redes IEEE 802.15.4, el cual tiene un tamaño de paquete de 127 bytes, este estándar permite lograr una compresión de los encabezados de IPv6 y UDP.

- IEEE 802.15.4. Posibilita especificar una subcapa para el Control de Acceso al Medio (MAC) y para el medio Físico (PHY), y define un formato para la trama y el encabezado.

- Bluetooth de Baja Energía (BLE). Comparado con la versión normal de Bluetooth, este protocolo nos brinda un rango más amplio, una latencia más baja y un gasto energético más bajo.

- Z - Wave. Fue diseñado primeramente para redes de automatización de casas inteligentes, pero por su bajo consumo también es utilizado para sistemas de IdC [8].

En la figura 2.2 [42] se muestra como se dividen los diferentes protocolos en cada capa que conforma el modelo OSI para IdC. Se puede observar que el protocolo MQTT, CoAP, XMPP y AMQP se encuentran en la capa de aplicación, mientras 6LoWPAN en la capa de red, Z-Wave en la capa de enlace de datos y el protocolo IEEE 802.15.4 en la capa física.

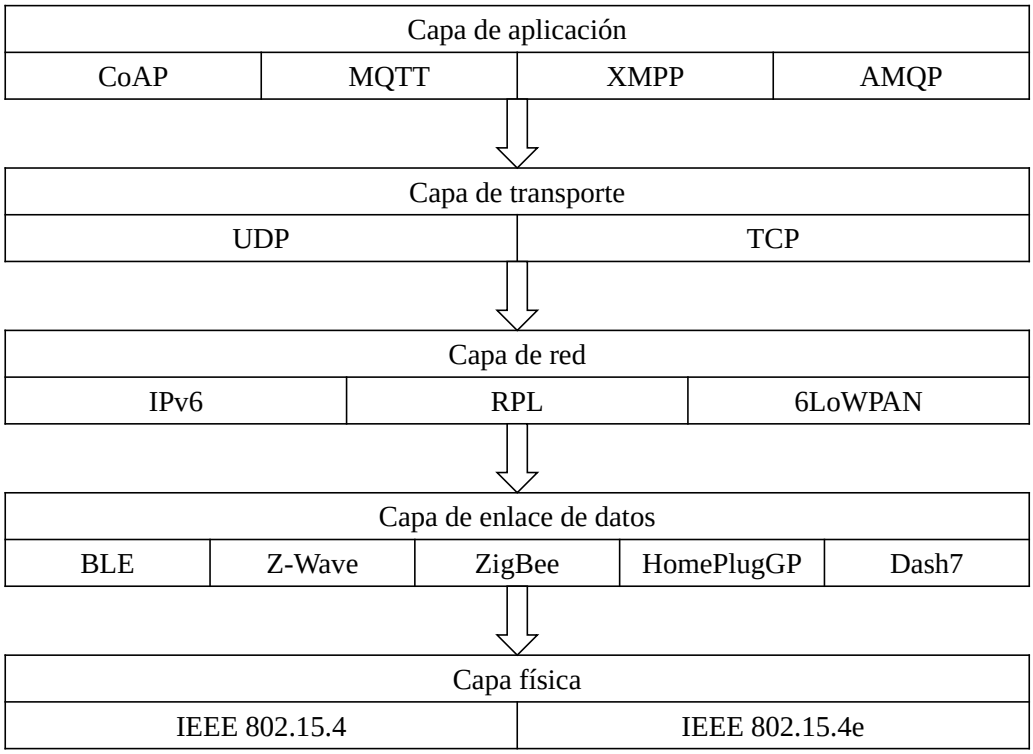


Figura 2.2: Protocolos en cada capa del modelo OSI para IdC

Todos estos protocolos se usan en las redes de IdC para el intercambio de información. Sin embargo, algunos protocolos son utilizados más que otros, ya sea porque son sencillos de utilizar o por su bajo consumo energético. En el resumen [43] que recopila los datos de una encuesta realizada a 502 participantes para poder comprender como los desarrolladores crean soluciones de IdC, se observa en la figura 2.3 una gráfica que muestra los porcentajes de los protocolos IdC con más tendencia en el año 2018. En la gráfica se puede observar que el protocolo MQTT lidera con un 62.51 %, mientras CoAP se encuentra con un 22.92 %, AMQP con un 18.24 % y XMPP con un 4.26 %.

En el resumen que trata sobre el panorama de la industria IdC, los desafíos que se enfrentan los desarrolladores de IdC y las oportunidades de IdC para un ecosistema de código abierto [44], se encuentra la figura 2.4 que muestra una gráfica con los protocolos más utilizados en el 2020. En esta gráfica se puede observar que en el ámbito general HTTP/HTTPS lidera con un 51 % del uso, mientras que MQTT se encuentra en segundo lugar con un 41 %. El protocolo MQTT es utilizado en gran medida en el ámbito general solo superado por el protocolo HTTP/HTTPS que es

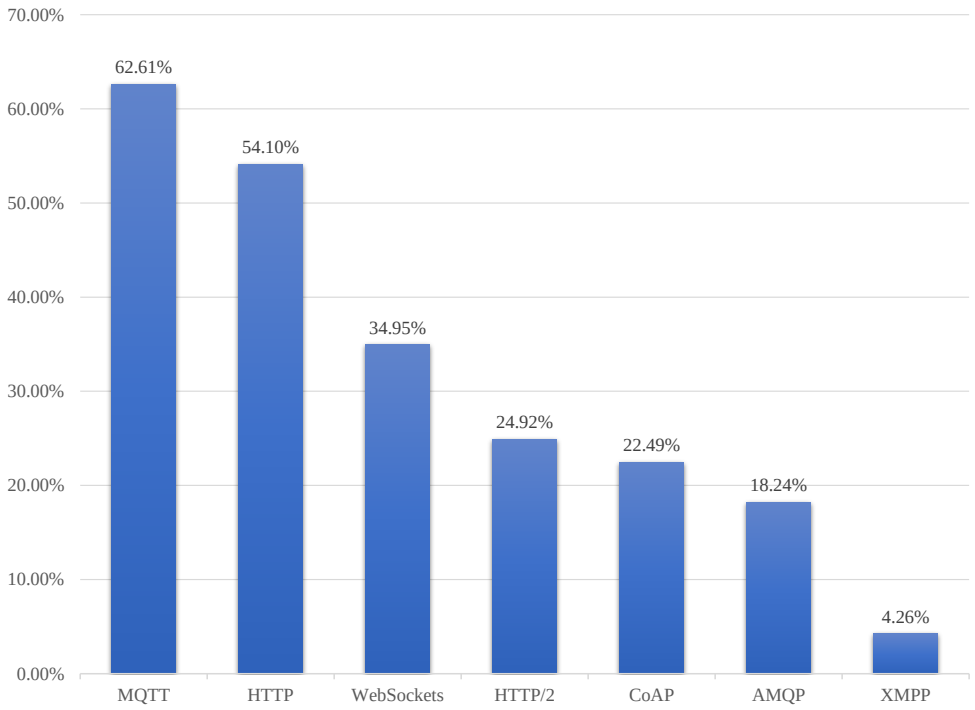


Figura 2.3: Tendencias de protocolos de comunicación en 2018

ampliamente utilizado en el internet. En IdC MQTT es mayormente utilizado en comparación con los protocolos CoAP, AMQP y XMPP.

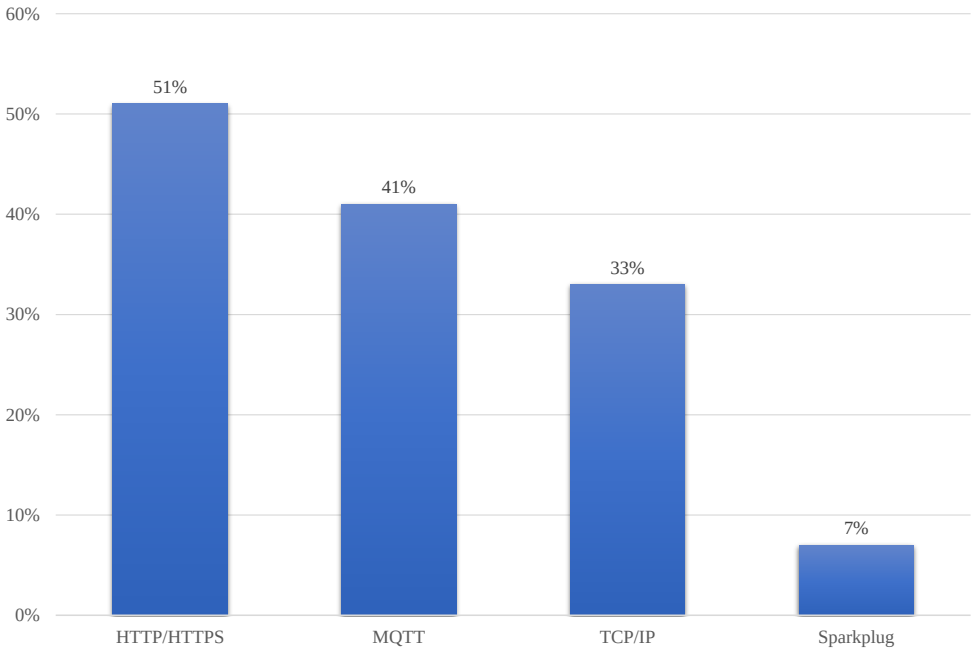


Figura 2.4: Principales protocolos de comunicación en 2020

2.4. Protocolo MQTT

MQTT es un protocolo de transporte de mensajería cliente-servidor del tipo publicación/suscripción mediante un tema en específico. Es ligero, abierto, simple y diseñado para ser fácil de implementar. Estas características lo hacen ideal para su uso en muchas situaciones, incluidos entornos limitados, como la comunicación en contextos máquina a máquina (M2M) e Internet de las cosas (IdC), donde se requiere un código pequeño y/o un escaso ancho de banda [2].

El protocolo MQTT se dio a conocer en 1999 y fue diseñado originalmente por Andy Stanford-Clark y Alen Nipper [3]. En 2014 la versión 3.1.1 de MQTT [45] se convirtió en un estándar de OASIS y en 2016 en el estándar de ISO/IEC 20922:2016. En marzo del 2019 se lanzó la versión 5.0 de MQTT [2], y de igual manera que su versión anterior acabo convirtiéndose en estándar de OASIS. Este protocolo ha sido ampliamente utilizado como ejemplos se pueden mencionar Facebook Messenger, Amazon IoT, OpenStack, Microsoft Azure IoT Hub, FloodNet, entre otros [46].

MQTT tiene las siguientes características.

- Uso del patrón de mensajes de publicación/suscripción, proporciona distribución de mensajes de uno a muchos y desacoplamiento de aplicaciones.
- Un transporte de mensajería que es independiente del contenido de la carga útil.
- Tres calidades de servicio para la entrega de mensajes.
- Una sobrecarga de transporte pequeña y los intercambios de protocolo minimizados para reducir el tráfico de red.
- Y un mecanismo para avisar a los interesados cuando se produzca una desconexión anómala [2].

MQTT ofrece tres niveles de calidad de servicio (QoS) para la entrega de mensajes: como máximo una vez (QoS 0), donde los mensajes se entregan de acuerdo con los mejores esfuerzos del entorno operativo y puede ocurrir la pérdida de mensajes; al menos una vez (QoS 1), donde se asegura que los mensajes llegarán, pero pueden ocurrir duplicados; y exactamente una vez, donde se asegura que los mensajes lleguen exactamente una vez[2].

El protocolo MQTT se ejecuta sobre TCP/IP o sobre otros protocolos de red que proporcionan conexiones bidireccionales ordenadas y sin pérdidas. Para el transporte sobre TCP/IP se utiliza el puerto 1883 y para SSL/TSL se usa el 8883 [47].

La arquitectura MQTT se puede observar en la figura 2.5. Esta se compone de tres componentes, un cliente-publicador, un cliente-suscriptor y un servidor o *broker*. Un cliente MQTT es un programa o dispositivo que utiliza el protocolo MQTT para la transferencia de información. Un cliente es responsable de abrir la conexión de red al servidor, crear mensajes para ser publicados, publicar mensajes de aplicación en el servidor, suscribirse para solicitar mensajes de aplicación que sea de interés

para su recepción, cancelar la suscripción para eliminar una solicitud de mensajes de aplicación y cerrar la conexión de red al servidor [47].

Un servidor MQTT o *broker* es un programa o dispositivo basado en el protocolo MQTT que actúa como una oficina postal entre editores y suscriptores. Un *broker* de MQTT es responsable de aceptar conexiones de red de cliente, aceptar mensajes de aplicaciones publicados por clientes, procesar solicitudes de suscripción y cancelación de suscripción de clientes, enviar mensajes de aplicación a clientes según sus suscripciones y cerrar la conexión de red del cliente [47].

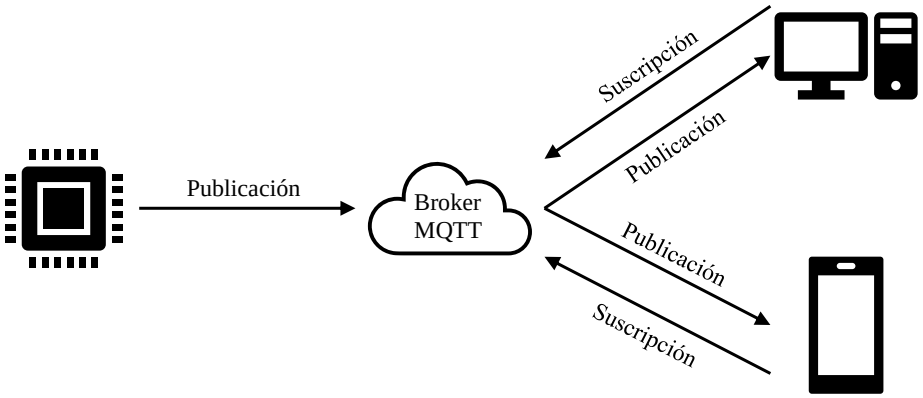


Figura 2.5: Diagrama de la arquitectura de red MQTT

Para el protocolo MQTT los paquetes de control tienen la misma estructura, esto se puede observar en la tabla 2.1. La estructura general está conformada en tres partes, el encabezado fijo, el encabezado variable y la carga útil que puede estar o no presente. La primera parte es el encabezado fijo, que se muestra en la tabla 2.2, esta parte se encuentra presente en todos los paquetes de control y se conforma por dos bytes, en la primera mitad del primer byte (bits 0 al 3) son indicadores específicos para cada paquete de control, en la segunda mitad del primer byte (bits 4 al 7) se indica el tipo de paquete de control. El segundo byte indica la longitud restante del paquete de control [2].

Encabezado fijo, presente en todos los paquetes de control MQTT	Byte 1 Byte 2
Encabezado variable, presente en algunos paquetes de control MQTT	Byte 3 Byte n
Carga útil, presente en algunos paquetes de control MQTT	Byte $n+1$ Byte m

Tabla 2.1: Estructura de un paquete de control MQTT

Bit	7	6	5	4	3	2	1	0
Byte 1	Tipo de control de paquete MQTT				Indicadores específicos para cada tipo de paquete de control MQTT			
Byte 2	Longitud restante							

Tabla 2.2: Encabezado fijo para un paquete de control MQTT

2.4.1. MQTT v5.0

MQTT 5.0 fue liberado en marzo del 2019. Utiliza 15 paquetes de control, uno más respecto a su versión anterior, estos paquetes son enumerados del 1 al 15 para ser identificados, como se muestran en la tabla 2.3. Cada paquete puede tener un tamaño de 2 Bytes hasta 256 MB. MQTT v5.0 agrega nuevas características con respecto a su versión anterior. Mejora en la escalabilidad y sistemas a gran escala, un informe de errores mejorado, una formalización de patrones comunes, incluido el descubrimiento de capacidades y la respuesta a solicitudes, mecanismos de extensibilidad que incluyen propiedades de usuario, mejoras de rendimiento y soporte para pequeños clientes, y la creación y adición de nuevos campos en los paquetes de control [2].

Nombre	**Valor**	**Dirección de flujo**	**Descripción**	**Carga útil**
Reservado	0	-	Reservado	No hay
CONNECT	1	Cliente → Servidor	Petición de conexión	Obligatorio
CONNACK	2	Cliente ← Servidor	Confirmación de conexión	No hay
PUBLISH	3	Cliente ↔ Servidor	Mensaje de publicación	Opcional
PUBACK	4	Cliente ↔ Servidor	Confirmación de publicación	No hay
PUBREC	5	Cliente ↔ Servidor	Recepción de publicación (entrega asegurada I)	No hay
PUBREL	6	Cliente ↔ Servidor	Lanzamiento de publicación (entrega asegurada II)	No hay
PUBCOMP	7	Cliente ↔ Servidor	Publicación completada (entrega asegurada III)	No hay
SUBSCRIBE	8	Cliente → Servidor	Petición de suscripción	Obligatorio
SUBACK	9	Cliente ← Servidor	Confirmación de suscripción	Obligatorio
UNSUBSCRIBE	10	Cliente → Servidor	Petición de cancelación de suscripción	Obligatorio
UNBSUBACK	11	Cliente ← Servidor	Confirmación de cancelación de suscripción	Obligatorio
PINGREQ	12	Cliente → Servidor	Solicitud de PING	No hay
PINGRESP	13	Cliente ← Servidor	Respuesta de PING	No hay
DISCONNECT	14	Cliente ↔ Servidor	Notificación de desconexión	No hay
AUTH	15	Cliente ↔ Servidor	Intercambio de autenticación	No hay

Tabla 2.3: Paquetes de control MQTT v5.0

Para realizar una conexión entre clientes se debe seguir una secuencia específica. El flujo de paquetes se puede observar en la figura 2.6. Primero ambos clientes envían un paquete CONNECT para realizar una petición de conexión con el *broker*, si no hay problemas o fallos el *broker* responde con un paquete CONNACK para confirmar la conexión. En el caso del cliente-publicador se envía un paquete PUBLISH hacia el *broker* con la información que se quiere intercambiar y el tema al que corresponde la información enviada, el *broker* a su vez responde con un paquete PUBACK para confirmar que el paquete se ha recibido de forma correcta. Cuando el *broker* ha recibido la información por medio del paquete PUBLISH, este se encarga de direccionar la información a los clientes-suscriptores que están conectados al mismo tema. Una vez terminada la transmisión de información se envía un paquete DISCONNECT para finalizar la conexión [2].

Para el cliente-suscriptor una vez que se ha establecido la conexión se procede a realizar la suscripción del tema, para ello se envía un paquete SUBSCRIBE con el nombre del tema al que se quiere suscribir, el *broker* contesta con un paquete SUBACK para confirmar que se realizó la suscripción de forma correcta. Una vez suscrito el cliente, este se queda esperando que el *broker* le envíe la información por medio del paquete PUBLISH y cuando esta acción ocurre el cliente responde con un paquete PUBACK para confirmar que se ha recibido la información. Cuando se quiere terminar la suscripción se envía un paquete UNSUBSCRIBE, el *broker* responde con un UNSUBACK y finalmente se envía un paquete DISCONNECT para terminar la conexión [2].

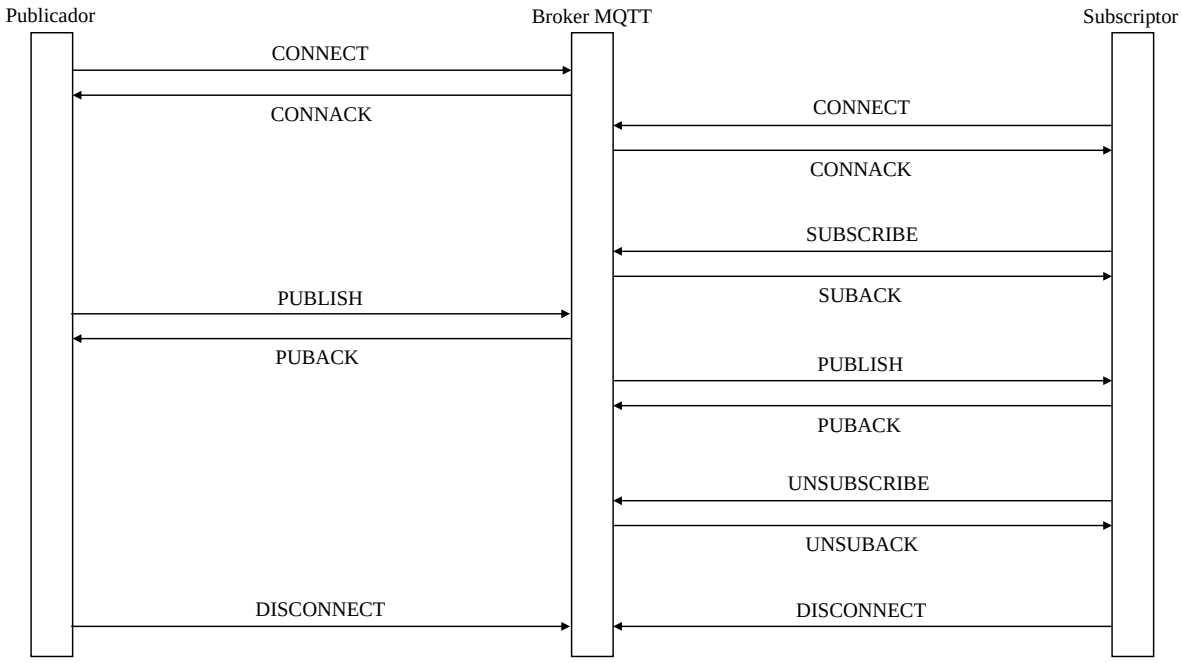


Figura 2.6: Diagrama que muestra el intercambio de paquetes para realizar la conexión entre los clientes y el *broker*

2.5. Esteganografía

La criptología es considerada como la rama de la seguridad informática que se encarga de estudiar el contenido que se encuentra cifrado, oculto e invisible en diferentes portadores, las cuales son consideradas como ciencias por la relación que tienen con otras áreas. En la figura 2.7 se muestra las diversas ramas que componen la criptología, entre las cuales se encuentra la esteganografía y la criptografía, que se analizan a continuación [1].

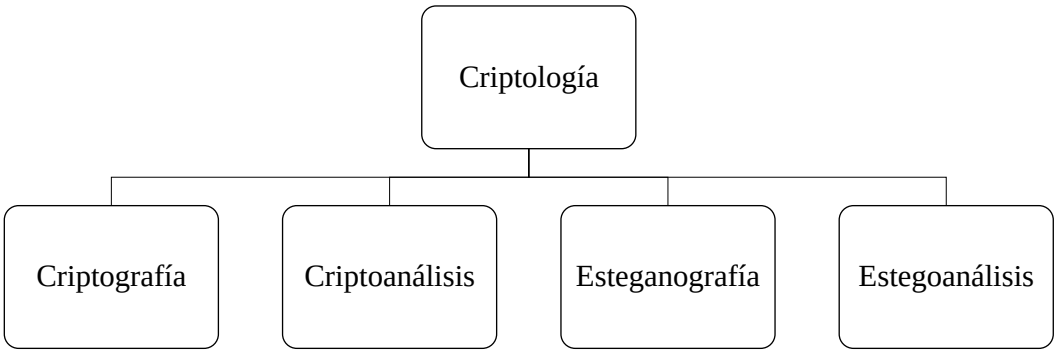


Figura 2.7: División de la criptología [1]

La esteganografía es la ciencia que permite una comunicación de datos ocultos a través de un objeto portador adecuado. El objetivo de la esteganografía es ocultar un mensaje dentro de un objeto portador de tal manera que no es posible identificar si el objeto contiene un mensaje o no, asegurando así que el mensaje solo sea accesible por el destinatario deseado y evitando una posible fuga de información. La esteganografía tiene varios campos de aplicación, como agencias de inteligencia, agencias militares, imágenes médicas, transmisión de televisión, incorporación de suma de chequeo que es una suma de corroboración para verificar la veracidad de los datos, estructuras de datos avanzadas, sistemas de radar y detección remota, en los cuales se necesita un nivel de seguridad alto para un correcto funcionamiento [48].

En el método esteganográfico se requiere tener los siguientes elementos importantes para su funcionamiento: el objeto portador el cual será el medio en donde se insertan los datos ocultos; el mensaje secreto que son los datos que se quieren transportar sin ser detectados; la función estego que es el método que ayuda a ocultar el mensaje secreto en el objeto portador; la función estego inversa que permite obtener el mensaje oculto del objeto portador una vez que se ha aplicado el ocultamiento por medio de la función estego y una clave estego que proporciona mayor seguridad al ocultar el mensaje, la implementación de este elemento es opcional [4, 49]. En la figura 2.8 se puede observar un diagrama general sobre un sistema esteganográfico.

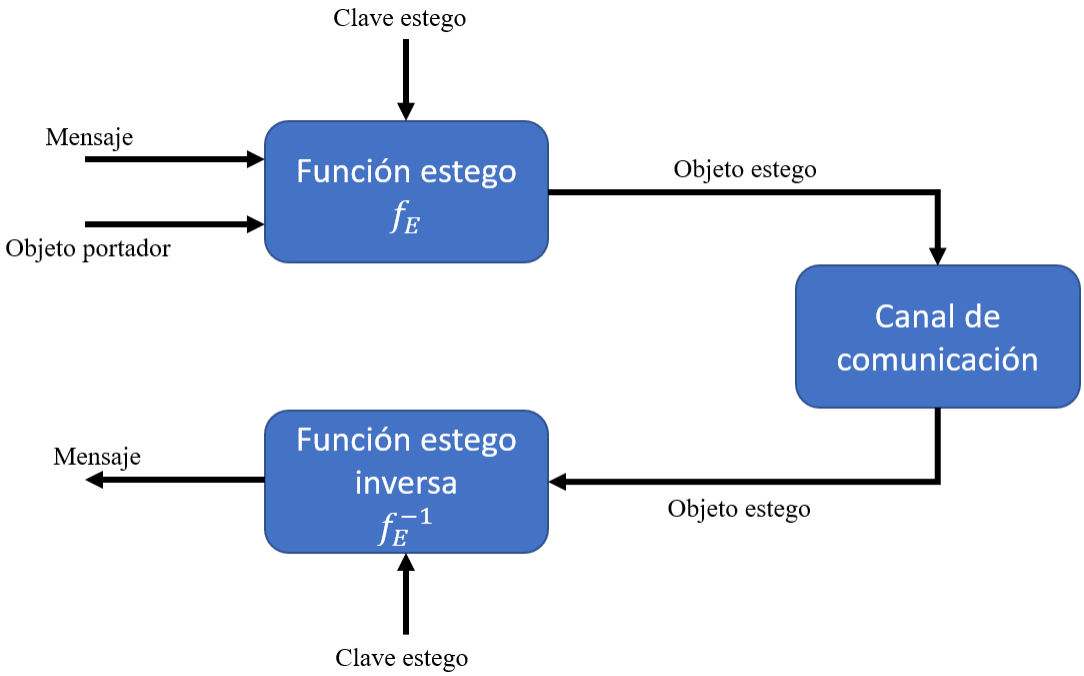


Figura 2.8: Diagrama general de un sistema esteganográfico

Una técnica esteganográfica ideal debe mantener y optimizar tres propiedades esenciales las cuales son: robustez, la propiedad que dificulta la posibilidad de eliminar los datos secretos del objeto estego; indetectabilidad, la propiedad de poder hacer alguna distinción entre el objeto estego y el objeto portador y, por último; capacidad, la cantidad máxima de datos secretos que puede ocultar el objeto portador [4].

Existen diversos métodos esteganográficos, los cuales se agrupan en dos categorías. La primera son los métodos en el dominio espacial, en donde los elementos que integran el mensaje secreto se pueden insertar en el objeto portador sin una modificación previa. La segunda son los métodos en el dominio de la frecuencia, en donde se modifican mediante transformaciones matemáticas los elementos antes de ser insertados [50].

Los objetos que pueden ser utilizados por la esteganografía para incrustar un mensaje secreto pueden ser un archivo de texto, una imagen fija, audio, vídeo o un protocolo de comunicación. A continuación, se mencionan algunas técnicas para los distintos objetos portadores que se utilizan para la esteganografía.

La esteganografía de texto plano consiste en ocultar el mensaje secreto dentro de un archivo de texto; usando como técnica el uso de caracteres seleccionados del objeto portador, en donde el remitente envía una serie de números enteros al destinatario, acordando que el mensaje secreto está oculto dentro de la posición respectiva de las palabras subsiguientes del texto de portada. Otra técnica es el uso de espacios en blanco adicionales en el objeto portador, cuyo funcionamiento es insertar varios espacios en blanco adicionales entre palabras consecutivas del objeto portador, y el número de espacios en blanco corresponde a un índice de una tabla de búsqueda [49].

Las imágenes fijas son el objeto portador más utilizado en esteganografía, esta técnica aprovecha la debilidad del sistema visual humano que no puede detectar la variación en la luminiscencia de los vectores de color en el extremo de frecuencia más alta del espectro visual, y así ocultar el mensaje en la imagen estego y hacerla pasar por la imagen original. Una técnica usada en la esteganografía de imágenes es la modificación del bit menos significativo del objeto portador, se aplica en el domino espacial, consiste en distribuir el mensaje oculto binario entre los bits menos significativos de cada píxel de la imagen. El inconveniente con la modificación del bit menos significativo es que es vulnerable a ataques, tales como la compresión y el formato de imágenes. Las siguientes técnicas se aplican en el dominio de la frecuencia, las cuales ocultan la información en partes significativas de la imagen y por lo tanto son resistentes a los ataques que las técnicas del dominio espacial no lo son. Las transformaciones más comunes que se utilizan son la transformada de coseno discreto y la transformada de ondícula. La técnica de espectro ensanchado trata la imagen como ruido y se intenta agregar ruido pseudoaleatorio al objeto portador. La técnica estadística, modifica las características del objeto portador y las conserva en el proceso de incrustación. La técnica de distorsión requiere que se tenga el objeto portador para el proceso de recuperación del mensaje secreto. La técnica de generación de cobertura se usa para ser una cobertura para la transferencia del mensaje secreto [51].

Usando como objetos portadores audio o video, el mensaje secreto se incrusta en la señal de audio o en los fotogramas del video, en este caso al tratarse de audio se debe tener en consideración que el sistema auditivo es más sensible a detectar las diferencias entre dos archivos de audio en comparación con el sistema visual para detectar la diferencia entre dos archivos de imágenes, por lo que las técnicas deben ser más complejas para lograr engañar al sistema auditivo. Las técnicas más usadas son: la codificación del bit menos significativo, el bit menos significativo de cada muestra de audio se reemplaza con el correspondiente bit del mensaje secreto. La codificación de fase codifica los bits del mensaje secreto en cambios de fase en el espectro de fase, logrando una codificación inaudible en términos de relación señal/ruido. El espectro ensanchado utiliza dos enfoques que son el espectro ensanchado de secuencia directa y el espectro ensanchado por salto de frecuencia, en esta técnica se ocupa más ancho de banda que la señal de información que se está modulando. La ocultación de eco consiste en incrustar la señal de ruido de cobertura como un eco, y se modifican la amplitud, la tasa de caída y el desplazamiento para representar el mensaje secreto [52, 53].

Recientemente se ha implementado un nuevo objeto portador que resultan ser los protocolos de comunicación, consiste en usar los campos que no son prescindibles en el protocolo y ocultar el mensaje en ellos, este método se conoce como esteganografía de red. La esteganografía de red se puede dividir en tres categorías: métodos que modifican el encabezado o la carga útil de los paquetes de red, que trata en que los campos del protocolo serán modificados y, algunos métodos dentro de esta división también modifican la carga útil del paquete, estos métodos tienen una alta capacidad de esteganografía; métodos que modifican la estructura de los flujos de paquetes,

modifican la forma en que son enviados los paquetes de red, algunos ejemplos son la afectación en el orden de envío de los paquetes, el tiempo de retardo en la transferencia entre paquetes y; los esquemas híbridos, estos métodos combinan el método de modificación de la cabecera del protocolo y también el flujo del paquete para así lograr una mejor indetectabilidad del mensaje oculto.

En la esteganografía por red se utilizan como objetos portadores los datagramas de los protocolos de comunicación y se realiza de forma que la modificación no pueda ser detectada por los observadores de red como rastreadores o sistemas de detección de intrusos. Las técnicas que se pueden usar para la esteganografía son las siguientes: protocolo TCP, para este protocolo se utilizan diferentes campos que conforman el encabezado TCP.

- Campo de número de secuencia inicial. La secuencia inicial es definida de manera distinta en cada sistema operativo por lo que resulta un medio perfecto para enviar datos debido a su naturaleza y tamaño.

- Campo bandera. Tiene un tamaño de 6 bits y puede llegar a tener 29 combinaciones que permiten mandar un mensaje de forma oculta sin alterar el funcionamiento del protocolo.

- Campo de puntero urgente. Está constituido por 16 bits y solo es relevante cuando se establece URG, lo que lo hace ideal como medio para un mensaje oculto.

En el protocolo IPv4 se utilizan los siguientes campos.

- Campo identificación del paquete. Es un número aleatorio generado por el remitente, si no se produce fragmentación es un campo ideal para insertar el mensaje a ocultar.

- Campo banderas. Consta de 3 bits en donde el primero es reservado, el segundo nos indica que no hubo fragmentación y el tercero que, si hubo fragmentación, en caso de no presentarse fragmentación el segundo bit es un valor no esencial por lo que sirve para insertar un bit del mensaje.

Y como último en el protocolo IPv6 se pueden aprovechar los siguientes campos

- Campo clase de tráfico. Tiene un tamaño de 8 bits.

- Campo etiqueta de flujo. Cuyo tamaño es de 20 bits.

- Campo dirección de origen. Se pueden usar 16 bytes para datos del mensaje oculto [54].

2.6. Criptografía ligera

La criptografía es la práctica y el estudio de ocultar información. Es la ciencia de rehacer mensajes para hacerlos seguros y resistentes a los ataques. En criptografía, el mensaje original se convierte en otro mensaje en el lado del cifrado y se convierte en el mensaje original en el lado del receptor [55]. Los algoritmos criptográficos fueron diseñados para satisfacer las necesidades de la era de la informática de escritorio. Esta criptografía no es adecuada para dispositivos basados en hardware y software altamente restringidos que necesitan comunicarse de forma inalámbrica.

La criptografía en IdC se utiliza para cumplir con los siguientes objetivos de seguridad fundamentales para el mensaje compartido.

Confidencialidad: el mensaje solo se puede visualizar por elementos, clientes, centros, dispositivos y administraciones autorizados. La información privada, las claves y las calificaciones de seguridad deben estar protegidas de elementos no aprobados.

Integridad: el primer mensaje no se altera.

Autenticación y autorización: la disponibilidad de los dispositivos dificulta el problema de la confirmación debido al control de entrada y la idea de correspondencia remota en los marcos de IdC.

Disponibilidad: El marco continúa llenando su necesidad y permanece ininterrumpidamente accesible para elementos genuinos. Se requiere que los marcos de IdC sean efectivos para permitir que las administraciones lleguen cuando sean necesarias.

Responsabilidad: para mejorar la cordialidad de las administraciones en la condición de IdC, la responsabilidad de los marcos de IdC es fundamental [56].

La criptografía ligera aborda problemas de seguridad para dispositivos altamente restringidos. El cifrado y descifrado ligero se implementan en plataformas, así como en hardware y software. Debido a las estrictas restricciones de costos de estas aplicaciones de gran volumen, las implementaciones económicas de software y hardware de los algoritmos criptográficos son de suma importancia para comprender la visión de la computación generalizada [6].

El objetivo de la criptografía ligera es permitir una amplia gama de aplicaciones, como sistemas de seguridad de vehículos, medidores inteligentes, pacientes inalámbricos, internet de las cosas, un sistema de transporte inteligente y sistemas de monitoreo. Esto se ve agravado por el hecho que estos dispositivos normalmente podrían caracterizar la interacción directa con el mundo físico, mediante los actuadores correspondientes, lo que podría comprometer la seguridad de los usuarios en caso de uso indebido.

En la criptografía ligera el sistema de seguridad es diseñado para dispositivos con recursos restringidos. El enfoque que se da a un algoritmo al momento de hacerlo liviano es en su implementación en hardware. La compuerta lógica requerida para ejecutar cualquier programa se denomina puerta equivalente y cuanto más bajo es el uso de puertas equivalente más ligero es el algoritmo. Para el desarrollo en software se trata de hacer el código más pequeño posible sin afectar la seguridad. El software debe ser compatible con el sistema operativo de los dispositivos pequeños que funcionan

con baterías [57].

Los algoritmos criptográficos se clasifican en dos categorías, cifrado de llave simétrica y llave asimétrica. La llave simétrica utiliza una única clave tanto para el cifrado como para el descifrado de datos, y el cifrado asimétrico utiliza dos llaves diferentes para cifrar y descifrar los datos. La criptografía de llave simétrica es segura y comparativamente rápida, su inconveniencia es que las partes que se comunican deben compartir la llave sin comprometerla. La criptografía asimétrica utiliza dos pares de llaves públicas y privadas. Garantiza la confidencialidad y la integridad al hacer uso de la llave pública del receptor y, además garantiza la autenticación al usar la llave privada del remitente para cifrar los datos. El problema del cifrado asimétrico es su aumento en la complejidad y la ralentización del proceso [58].

Existen dos propiedades fundamentales que necesita un algoritmo criptográfico, la confusión y la difusión. La confusión hace que la relación entre el texto cifrado y la clave sea lo más complejo posible, mientras que la difusión disipa la estructura estadística del texto plano sobre la mayor parte del texto cifrado mediante la permutación [58].

En la criptografía simétrica se encuentran el cifrado por bloques y el cifrado por flujo. En el cifrado por bloques, tanto el cifrado como el descifrado se realiza en un bloque de tamaño fijo (64 bits o más), mientras que el cifrado por flujo procesa continuamente los elementos de entrada bit a bit (o palabra a palabra). El cifrado por flujo usa solo la propiedad de confusión, en cambio el cifrado por bloques usa tanto la confusión como la difusión con un diseño simple [58].

Los algoritmos de criptografía ligera están diseñados manteniendo el *Advanced Encryption Standard* (AES) como algoritmo estándar, debido a su estandarización por el *National Institute of Standards and Tecnology* (NIST), se basa en una red de permutaciones y sustituciones. PRESENT también se basa en la estructura de permutación por sustitución, es eficiente en cuanto al hardware, pero la capa de permutación del cifrado consume grandes ciclos a nivel de software. RECTANGLE es una mejora del algoritmo PRESENT, es un cifrado de bloque ultraligero de segmento de bits adecuado para múltiples plataformas que tienen un área muy baja en hardware y también un rendimiento muy competitivo en software [58].

High security and lightweight (HIGHT) está basado en la estructura de red de Feistel. CLEFIA es otro cifrador basado en una red de Feistel y fue estandarizado por el NIST en 2007, es un cifrador equilibrado en rendimiento y seguridad, tiene un buen rendimiento de hardware en comparación con otros cifradores por bloque. CAMELLIA es un cifrador por bloques de clave simétrica, fue diseñado para su implementación de software y hardware, se utiliza para tarjetas inteligentes de bajo costo para sistemas de red con altas velocidades. TWINE se basa en una red Feistel generalizada de tipo dos con una mezcla de bloques altamente difusiva, cabe en un hardware muy pequeño y proporciona un rendimiento notable en software integrado. SIMON y SPECK se introdujeron en 2013, SIMON es un cifrador que tambien están basados en una red de Feistel, que está optimizado para el rendimiento en implementaciones de hardware, por otro lado, SPECK se ha optimizado para implementaciones de software [58].

2.6.1. Cifrador SIMON

SIMON es una familia de cifrados de bloque liviano diseñados por la Agencia de Seguridad Nacional de EE.UU y publicados en 2013. El cifrado realizado con una palabra de n bits se denomina SIMON2n, donde n requiere tomar los valores 16, 24, 32, 48 o 64. SIMON2n con una palabra de tamaño mn bits se denomina SIMON2n/mn, por ejemplo, SIMON64/128 se refiere a la versión de SIMON que utiliza bloques de texto plano de 64 bits y una llave de 128 bits para cifrar el texto plano.

SIMON2n para el cifrado y descifrado utiliza las siguientes operaciones en palabras de n-bits [6].

- XOR bit a bit, $\oplus$
- AND bit a bit, &
- Desplazamiento circular a la izquierda S^j, donde j es el número de bits.

El cifrado SIMON se puede describir de forma matemática de la siguiente manera. Para la función de ronda de SIMON2n que utiliza la llave $k \in GF(2)^n$, se realiza una red de Feistel de dos etapas $R_k : GF(2)^n \times GF(2)^n \to GF(2)^n \times GF(2)^n$ que esta definida por la siguiente ecuación:

$$R_k(x, y) = (y \oplus f(x) \oplus k, x), \tag{2.1}$$

en donde $f(x) = (Sx \& S^8 x) \oplus S^2 x$ y k es la llave de ronda. La función de ronda inversa que se utiliza para el descifrado se puede observar en la siguiente ecuación [6]:

$$R_k^{-1}(x, y) = (y, x \oplus f(y) \oplus k). \tag{2.2}$$

La llave de ronda es obtenida de generar una secuencia de T palabras de llaves $k_0, \ldots, k_{T-1}$, donde T es el número de rondas. El mapa de cifrado se compone de la secuencia $R_{k_{T-1}}, \ldots, R_{k_1}$. En la figura 2.9 se observa el diagrama de la red de Feistel que se utiliza en el cifrador SIMON. Se ve como la información se separa en dos bloques x_{i+1} del lado izquierdo y x_i del lado derecho. Al bloque x_{i+1} se le aplican dos desplazamientos circulares a la izquierda uno de un bit y otro de ocho bits, a estas rotaciones se les aplica una operación AND. Posteriormente al bloque x_{i+1} se le aplica otro desplazamiento circular de dos bits y se aplica una operación XOR entre la señal obtenida por la operación AND, la señal desplazada dos bits y la señal de entrada x_i. Con la señal obtenida de la operación XOR se aplica otra operación XOR con la llave de ronda k_i y la señal obtenida será la señal de salida x_{i+2}. La señal x_{i+1} se cruza para quedar en el bloque derecho y la señal x_{i+2} queda del lado izquierdo [6].

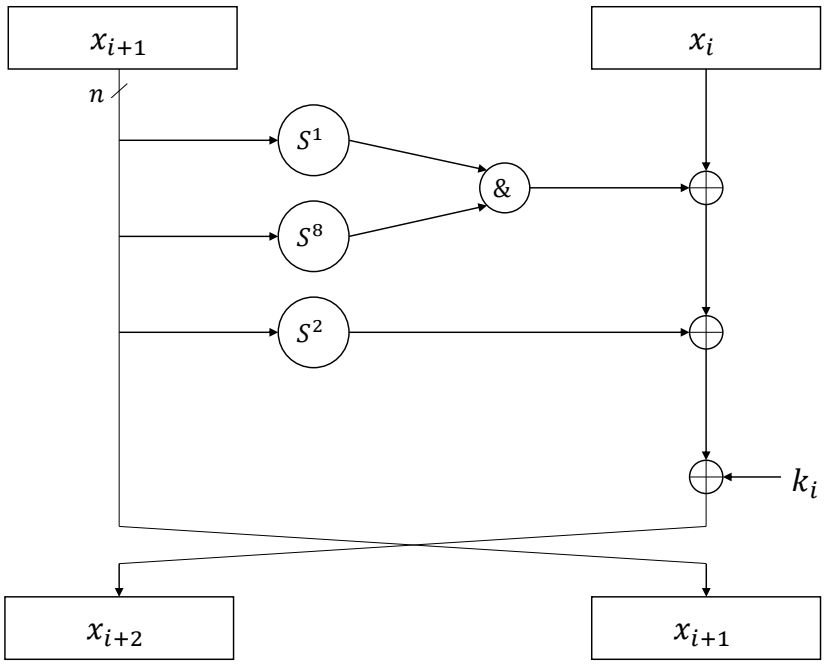


Figura 2.9: Red de Feistel para la función de ronda del cifrador SIMON

Generación de las llaves de ronda

Las operaciones del cifrado SIMON son las mismas y son perfectamente simétricas en todas las rondas, respecto al mapa de bits del desplazamiento circular en palabras de n bits. La llave de ronda emplea una secuencia de constantes de un bit para eliminar las propiedades del desplazamiento y la simetría. Se utilizan las constantes z_0, z_1, z_2, z_3 y z_4. Estas constantes se definen de la siguiente manera [6].

$$z_0 = 00011001110000110101001000101111101100111000011010100100010111111$$

$$z_1 = 00010110100001100100111110111000101011010000110010011111101110001$$

$$z_2 = 00110011011010011111100010000101000110010010110000001110111101 01$$

$$z_3 = 00111100001011001110010100010010000001111010011000110101110110 11$$

$$z_4 = 00111101110010010100110000111010000001000110110101100111100010 11$$

Sea $c = 2^n - 4 = 0xff\ldots fc$. Para SIMON2$n$ con m palabras de llave $(k_{m-1}, \ldots k_1, k_0)$ y la secuencia de ronda z_j, las llave de ronda se generan de la siguiente manera:

$$k_{i+m} = \begin{cases} c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+1}, & \text{si } m = 2 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})S^{-3}k_{i+2}, & \text{si } m = 3 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}), & \text{si } m = 4 \end{cases} \tag{2.3}$$

para $0 < i < T$. Para conocer la constante z_j que se utiliza en el cifrado se consulta la tabla 2.4, en esta se observan también los parámetros que se necesitan saber

para cada configuración del cifrador SIMON. El tamaño del bloque es la cantidad de información de entrada, medida en bits, que se quiere cifrar. El tamaño de llave es el tamaño de la llave con la que se va a cifrar la información. El tamaño de palabra son el tamaño de los bloques que dividen la llave de cifrado. Las palabras llave son el número de bloques que se generan al dividir la llave de cifrado. La constante de ronda son las constantes $z_0 \dots z_4$ que se utilizan en cada configuración. El número de rondas son las rondas totales en la que se aplica la red de Feistel a la información que se va a cifrar. Las palabras de llave k_0 a k_{m-1} son usadas como las primeras m llaves de ronda. Estas palabras son desplazadas en donde k_0 se ubica a la derecha y k_{m-1} a la izquierda [6].

Tamaño de bloque $2n$	Tamaño de llave mn	Tamaño de palabra n	Palabras llave m	Constante de ronda	Número de rondas T
32	64	16	4	z_0	32
48	72	24	3	z_0	36
	96		4	z_1	36
64	96	32	3	z_2	42
	128		4	z_3	44
96	96	48	2	z_2	52
	144		3	z_3	54
128	128	64	2	z_2	68
	192		3	z_3	69
	256		4	z_4	72

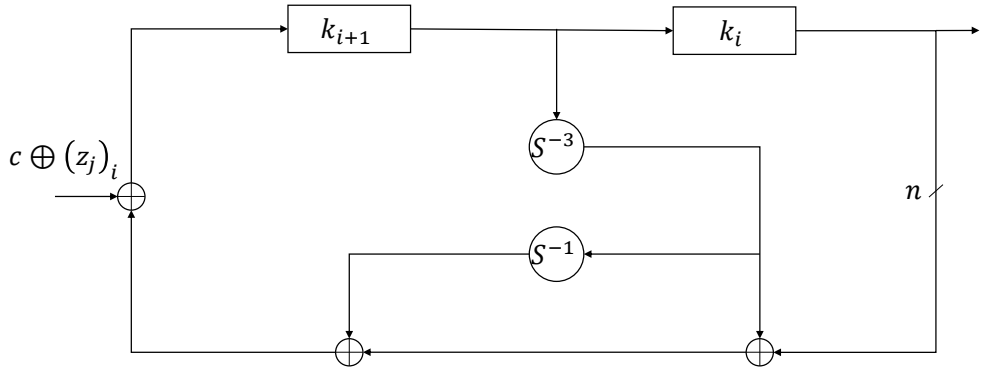
Tabla 2.4: Parámetros SIMON

Los diagramas para la generación de las llaves de ronda se observan en la figura 2.10. En la figura 2.10a se observa el diagrama cuando el número de palabras de llave es dos, se observa como al bloque k_{i+1} se le hace un desplazamiento circular de tres bits hacia la derecha y se obtiene otra señal con el bloque k_{i+1} desplazado circularmente cuatro bits a la derecha, a estas señales se les aplica un XOR con el bloque de llave k_i. A la señal obtenida se le aplica una operación XOR con la constante de ronda y el valor de c y así obtener la siguiente llave de ronda [6].

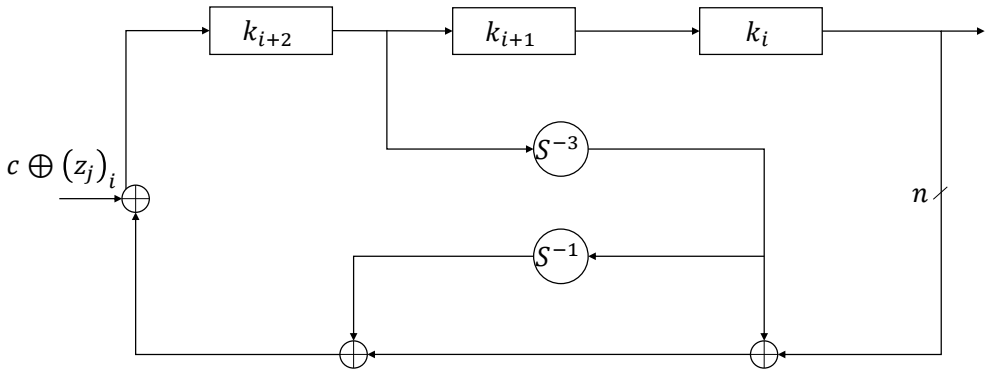
En la figura 2.10b, se muestra el diagrama cuando el número de palabras de llave es igual a tres. Se toma el bloque de llave k_{i+2} y se aplican los desplazamientos circulares hacia la derecha de tres y cuatro bits, posteriormente se aplica la operación XOR entre estas señales y la señal del bloque k_i. A esta señal obtenida se aplica una operación XOR con la constante de ronda y el valor de c para obtener la siguiente llave de ronda [6].

Se observa en la figura 2.10c el diagrama cuando la llave se separa en cuatro palabras de llave. En este caso se toma el bloque k_{i+3} para aplicar el desplazamiento circular a la derecha de tres y cuatro bits. A diferencia de los diagramas anteriores, en este diagrama se aplica la operación XOR a la señal desplazada tres bits a la derecha con el bloque k_{i+1}, se realiza una operación XOR entre la señal obtenida

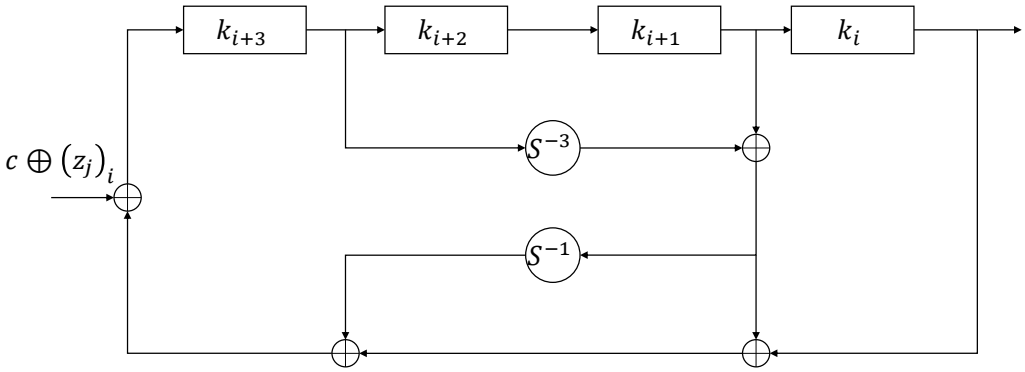
anteriormente junto con la señal desplazada cuatro bits a la derecha y el bloque k_i. Finalmente, para obtener la llave de ronda siguiente se aplica una operación XOR entre la señal resultante anterior, la constante de ronda y el valor de c [6].



(a) $m = 2$ palabras de llave.



(b) $m = 3$ palabras de llave.



(c) $m = 4$ palabras de llave.

Figura 2.10: Generación de llave de ronda para el cifrador SIMON

2.6.2. Cifrador SPECK

SPECK se ha diseñado para tener un rendimiento excelente tanto en hardware como en software. Sin embargo, ha sido optimizado para trabajar en microcontrola-

dores. La notación que se utiliza para describir la configuración del cifrador SPECK es simular a la del cifrador SIMON. Si se especifica una configuración SPECK96/144 se refiere a un cifrado con un tamaño de bloque de 96 bits y una llave de 144 bits.

El cifrador SPECK utiliza las siguientes operaciones con palabras de n bits [6].

- XOR bit a bit $\oplus$,
- Adición módulo 2^n $+$,
- Desplazamiento a la izquierda y derecha, S^j y S^{-j} respectivamente, donde j es el número de bits.

Para $k \in GF(2)^n$, la ecuación que define la función de ronda del cifrador SPECK es la siguiente:

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k), \tag{2.4}$$

donde los valores de las rotaciones son $\alpha = 7$ y $\beta = 2$ en caso de que $n = 16$ (tamaño de bloque $= 32$), y $\alpha = 8$ y $\beta = 2$ para otros tamaños de bloque.

La ecuación que se utiliza para la función de ronda inversa, que es necesaria para el descifrado, usa la substracción modular en lugar de la adición modular y se define en la ecuación (2.5) [6].

$$R_k^{-1}(x, y) = (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)) \tag{2.5}$$

La llave de ronda se obtiene de una secuencia que es generada mediante T palabras de llave $(k_0, \ldots, k_{T-1})$, donde T es el número de rondas.

En la figura 2.11 se observa el diagrama para el cifrador SPECK. El bloque de información de entrada se divide en dos bloques x_{2i} a la derecha y x_{2i+1} a la izquierda. Al bloque x_{2i+1} se le aplica un desplazamiento circular a la derecha de α bits. Esta señal se suma con el bloque x_{2i}, y se aplica una operación XOR con la llave de ronda, esta señal es el bloque de salida a la izquierda x_{2i+3} . A la señal del bloque x_{2i} se le aplica un desplazamiento a la izquierda de β bits, y se aplica una operación XOR con la señal del bloque de salida x_{2i+3}, la señal resultante es el bloque de salida a la izquierda x_{2i+2} [6].

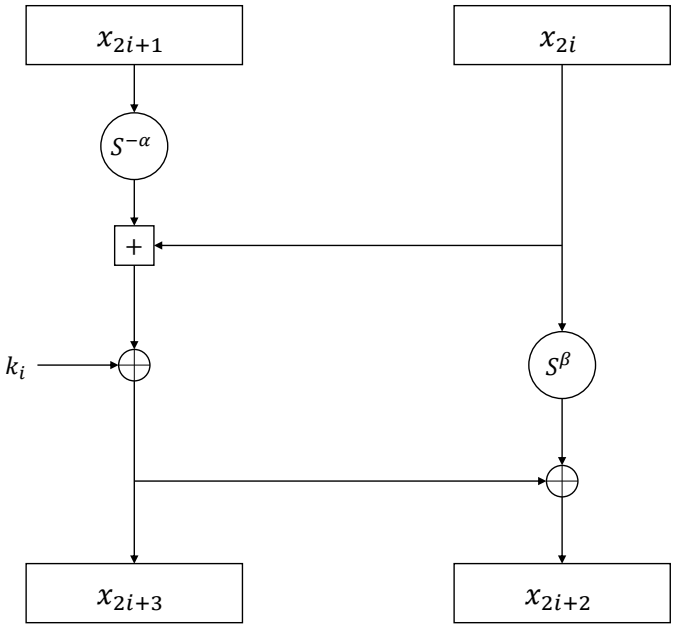


Figura 2.11: Red de Feistel para la función de ronda del cifrado SPECK

Generación de llave de ronda

La generación de llave de ronda k_i se realiza mientras se ejecuta la función de ronda. Sea K una llave para el cifrador de bloques SPECK2n, se puede escribir como $K = (\ell_{m-2}, \ldots, \ell_0, k_0)$, en donde $\ell_i, k_0 \in GF(2)^n$, para un valor de m en $\{2, 3, 4\}$. Los valores de las secuencias k_i y ℓ_i estan definidas por las siguientes ecuaciones:

$$\ell_{i+m-1} = (k_i + S^{-\alpha}\ell_i) \oplus i \tag{2.6}$$

$$k_{i+1} = S^{\beta}k_i \oplus \ell_{i+m-1} \tag{2.7}$$

donde k_i es la i-ésima llave de ronda para $0 \leq i < T$. En la figura 2.12 se puede observar el diagrama para la generación de las llaves de ronda del cifrado SPECK. La llave de ronda k_i es la llave que se va generando mientras que los bloques de llave $\ell_{2i+1} \ldots \ell_i$ se van desplazando y funcionan como entrada de la siguiente ronda. R_i representa la red de Feistel que se utiliza para el cifrado SPECK, el bloque de salida del lado izquierdo se utiliza para el desplazamiento de los bloques de llave y el bloque de salida de la derecha es la llave de ronda siguiente [6].

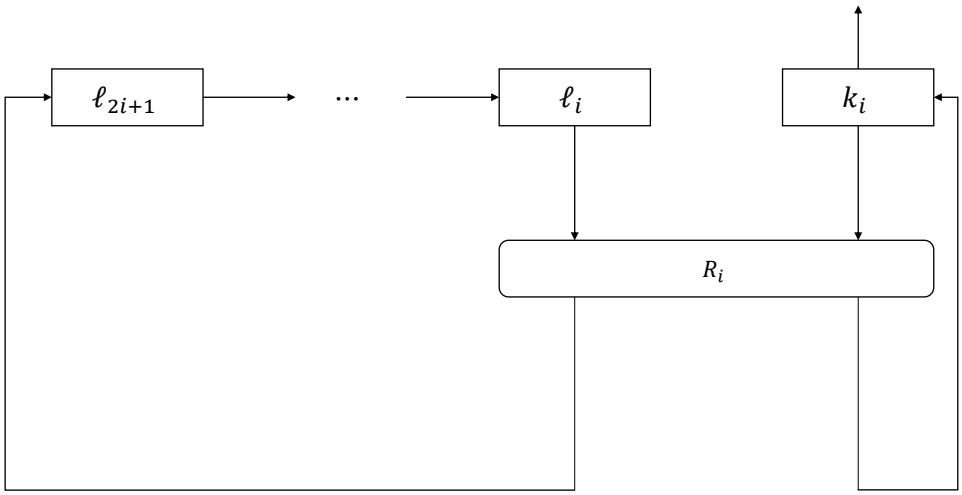


Figura 2.12: Generación de la llave de ronda para el cifrador SPECK

Capítulo 3

Estado del arte

La tecnología IdC ha ido en aumento en los últimos años y ha sido un gran tema de interés en la comunidad, sin embargo, con el aumento de su uso también aumentan las vulnerabilidades y se descubren fallas que no estaban contempladas. En una tecnología que permite el intercambio de información se debe garantizar al usuario que su información permanecerá segura y no será de conocimiento para un tercero no autorizado.

En esta sección se presentan los trabajos relacionados a esta tesis, para ello se muestran cinco temas que tienen relación con el tema que se trabaja en la tesis, estos son trabajos de seguridad en IdC con un enfoque general, esteganografía con un enfoque general, esteganografía usando como objeto portador un protocolo de comunicación, seguridad en IdC que usa esteganografía en general y para finalizar seguridad en IdC usando esteganografía como objeto portador el protocolo de comunicación.

3.1. Seguridad en IdC con un enfoque general

Dorri, et. al [59], proponen un método de cadena de bloques ligero escalable (LSB), que proporciona seguridad de extremo a extremo optimizado para los requisitos de IdC. El LSB fue desarrollado con un algoritmo de gestión de rendimiento distribuido que garantiza que el rendimiento de la cadena de bloques no se desvié de la carga acumulada de transacciones de la red. Los resultados que obtuvieron fueron que el LSB es altamente seguro, ya que si los nodos clave fallan se puede observar una degradación excelente. Sus simulaciones mostraron que su arquitectura reduce el ancho de banda y el tiempo de procesamiento en comparación con los métodos clásicos de cadena de bloques.

Sarker, et. al [60], mencionan que el uso de inteligencia artificial, en especial soluciones de aprendizaje automático y profundo, son de gran importancia para desarrollar un sistema de seguridad para el sistema IdC. Se presenta una investigación sobre la inteligencia de seguridad de IdC, que se basa en tecnologías de aprendizaje automático y profundo para proteger los datos de ataques cibernéticos. Concluyeron que el sistema antes de tomar una decisión inteligente debe desarrollar un algoritmo

de aprendizaje eficaz con el conocimiento de seguridad adquirido con la aplicación de destino, además que las soluciones enfocadas en aprendizaje profundo son un camino prometedor para implementaciones futuras de seguridad en IdC.

Kudithi y Sakthivel [61], utilizan criptografía de curva elíptica ya que proporciona mejores estándares de seguridad en comparación con otros algoritmos criptográficos convencionales. Se enfocaron en mejorar la velocidad de cómputo y el área requerida para la implementación de hardware, y así lograr una manera eficiente de usar los recursos de hardware compartidos y los mecanismos de programación para curvas elípticas en coordenadas afines. Desarrollaron una arquitectura de hardware para realizar una multiplicación escalar y así reducir el área y el número de ciclos utilizados en comparación con otros diseños que utilizan las mismas coordenadas afines.

Otro método de seguridad es el trabajo realizado por Al-Refai y Alawneh [62], ellos proponen y desarrollan un marco de seguridad IdC mejorando la autenticación y autorización. Utilizaron un método de autenticación de un identificador, mejorándolo en términos de expandir el identificador estándar al agregar información de verificación de identidad y un nivel de permiso de autorización para los datos de la carga útil del identificador. Posteriormente el servicio cifra la carga útil para evitar que los datos sean comprometidos en caso de un robo de datos. La autenticación se realiza mediante una técnica inteligente basada en una marca de tiempo que refleja el tiempo real de la solicitud enviada. Además de utilizar una huella dactilar biométrica para aumentar el nivel de seguridad de autenticación y evitar ataques de fuerza bruta. Concluyeron con los resultados de sus pruebas, que su sistema protege las redes y los protocolos de IdC de diferentes ataques y la carga que se agrega al dispositivo IdC es muy baja.

3.2. Esteganografía con un enfoque general

Tomando en cuenta los trabajos realizados utilizando esteganografía con un enfoque general para ocultar información se encontraron los siguientes. AlWatyan, et. al [4], realizaron un método automatizado para proteger un mensaje utilizando dos niveles de seguridad. El primer nivel cifra los datos utilizando el método de cifrado "*Character Bit Shuffle*". En el segundo nivel los datos cifrados se ocultan dentro de la imagen insertando dos bits en las dos posiciones menos significativas de los píxeles de 32 bits de la imagen, llamando a esta técnica LSB 1-1-0. Concluyeron que su método LSB 1-1-0 produce un PSNR promedio de 54.16 dB, esto demuestra que las imágenes generadas por esta técnica tienen una fuerte imperceptibilidad, además se reduce el área en un 80 % en comparación con otros trabajos.

Kait y Chauhan [63] trabajaron con la esteganografía de segmentación de complejidad de plano de bits (BPCS) que da una mejor imperceptibilidad visual. Esta técnica utiliza regiones que son similares al ruido en los planos de bits de la imagen portadora. Para incrustar la información dentro de las imágenes se requieren de cálculos intensivos, por lo tanto, la técnica se implementa en una FPGA para aumentar la velocidad de procesamiento. Al final encontraron que la arquitectura de hardware

FPGA de la técnica BPCS muestra una mejora significativa en comparación con una implementación LSB.

En [64] Martínez, et. al presentaron la arquitectura y la implementación en FPGA de un sistema de ocultamiento de señales de voz ocultándolo en un ruido que proviene de las propias componentes Wavelet de la señal. La conclusión a la que llegaron fue que su ventaja es aprovechar los componentes de la señal y no utilizar otra señal como lo hacen otros métodos esteganográficos que utilizan circuitos adicionales para almacenar o procesar la señal extra. El sistema completo está compuesto por el codificador y el decodificador que tienen la capacidad de trabajar en tiempo real.

3.3. Esteganografía usando un protocolo de comunicación

El concepto de esteganografía de red fue introducido en 2003 [65], donde se utiliza un protocolo de comunicación como objeto portador, donde Szczypiorski analizó el datagrama del protocolo TCP/IP para descubrir los posibles campos que se pueden utilizar para ocultar información. En esta misma línea, Kundur y Ahsan [66], investigan dos enfoques para ocular información, el primero manipulando el encabezado de los paquetes y el segundo utilizando el campo de número de secuencia inicial, además encontraron que se pueden utilizar los campos del protocolo que no son utilizados. Los trabajos relacionados con esteganografía de red que se encontraron son los siguientes.

Melo, et. al [67] usaron el campo número de secuencia inicial del protocolo TCP, con hipótesis de que es más difícil detectar debido al valor arbitrario que puede tener. Proponen aumentar la indetectabilidad agregando identificadores dinámicos. Con las diferentes técnicas de clasificación que utilizaron observaron que el esquema que propusieron tiene una menor tasa de detectabilidad y es menos detectable en comparación con otras técnicas.

En su trabajo Bobade y Goudar [54] utilizan el protocolo IPv6 para ocultar información cifrada mediante el algoritmo RSA, en el campo de etiqueta de flujo con un tamaño de 20 bits. Ellos concluyeron que no es posible la detección de la información debido a las transformaciones y cálculos aleatorios aplicados en el algoritmo de codificación.

Como parte de su investigación Xue, et. al [68] desarrollan un sistema esteganográfico de dos niveles, en el nivel superior se utiliza para transmitir el texto cifrado. En el nivel inferior se utiliza para entregar la llave de cifrado de forma encubierta. Se utilizan los paquetes como TCP y UDP que representan un 0 y 1 respectivamente que se utiliza para codificar la clave como método del nivel inferior. Los resultados que obtuvieron mostraron que se logra un alto ancho de banda y una fuerte indetectabilidad.

Aprovechando el mecanismo de retransmisión del protocolo TCP Brodzki y Bieniasz [69] sobrescriben la carga útil de los segmentos TCP sin calcular las sumas de verificación, este segmento se puede considerar como incorrecto y se pueden ex-

traer datos ocultos. A pesar de considerarse incorrecto el segmento se retransmite de acuerdo con las especificaciones del protocolo y la comunicación no se ve afectada. Realizaron un análisis estadístico que mostró la alta resiliencia de la técnica a la detección.

3.4. Seguridad en IdC que usa esteganografía en general

En los siguientes trabajos se abordan temas como esteganografía en IdC pero usando como objeto portador un elemento distinto a un protocolo de comunicación. En el trabajo Geethanjali, et. al [70] presenta una técnica que combina criptografía y esteganografía como método de seguridad para la transferencia de información. Para la parte criptográfica utilizaron curva elíptica y en la parte esteganográfica utilizaron el bit menos significativo. Concluyeron que su técnica logra un mayor grado de seguridad en los datos en una red IdC. Evaluaron sus resultados con los parámetros MSE, PSNR, eficiencia de incorporación y complejidad de tiempo y los compararon con técnicas existentes como FMO, XOR y OMME.

En el artículo [7] Khari, et. realizan una combinación de técnicas criptográficas y esteganográficas para la seguridad en una red IdC. Usando criptografía elíptica de Galois para cifrar los datos y Matriz XOR para ocultar los datos cifrados, además de utilizar *Adaptive Firefly* [1] para optimizar la selección de bloques de cobertura dentro de la imagen. Con lo realizado concluyeron que se mejoró la seguridad debido a la capacidad avanzada de ocultación de datos, simulando el sistema en MATLAB encontraron que se logra una eficiencia de aproximadamente 86 % en la incorporación de esteganografía.

3.5. Seguridad IdC usando criptografía o esteganografía con un protocolo de comunicación como objeto portador

Los trabajos que incorporan esteganografía con un protocolo de comunicación como objeto portador son los siguientes. Trujillo, et al. [71], proponen integrar un método de criptografía basado en el caos como seguridad adicional para proteger datos confidenciales, en este caso se utilizan imágenes RGB, de extremo a extremo. Ellos introducen un algoritmo simple, seguro y eficiente que mejora la aleatoriedad de los mapas caóticos 1D que ayuda en el cifrado de imágenes en tiempo real. Este algoritmo es factible de implementar en dispositivos de telecomunicaciones que emplean multiprocesadores, o cualquier dispositivo IdC con capacidades de procesamiento de imágenes. El sistema se verificó utilizando enlaces M2M a través del protocolo MQTT

[1] Algoritmo metaheurístico inspirado en el comportamiento del centelleo de las luciérnagas

en internet. El análisis mostró que el algoritmo de cifrado propuesto que utiliza la función mod1023 ofrece robustez y alta seguridad contra varios ataques. La implementación fue realizada en una computadora personal con un reloj de 2.9 GHz y usando las secuencias mejoradas con el mapa Logistic 1D alcanza un rendimiento de hasta 47.44 Mb/s, y la implementación en Raspberry Pi 4 alcanza los 10.53 Mb/s.

Almohammedi y Shepelev [72] analizaron el rendimiento de un sistema basado en el modelo de cadena de Markov 2-D en relación con el estándar IEEE 802.11p para redes IdC. Encontraron que los valores del rendimiento del sistema de canal esteganográfico para tramas de datos y control son bajos a medida que aumenta la tasa de tráfico, el número de vehículos, el tamaño del paquete y el valor de BER, además el valor del rendimiento del sistema del canal esteganográfico basado en tramas de datos y control disminuye cuando la capacidad del canal es pequeña y el tamaño de la red es grande.

En su trabajo Velinov, et. al [46], analizaron el protocolo MQTT v3.1.1 para encontrar una manera viable de ocultar información, encontrando siete canales encubiertos directos y seis indirectos que evaluaron y categorizaron con base en un enfoque de patrones de ocultación de información de red. Realizando la implementación de dos canales encubiertos indirectos demostraron la viabilidad de la ocultación de información en el protocolo MQTT v3.1.1.

En el artículo de Koziak, et. al [73], modificaron un sistema de detección de intrusos para detectar ciertos tipos de esteganografía, esto mediante el software Zeek. Realizaron pruebas en protocolos, tales como, ICMP, TCP, IP, MQTT y SIP. Demostraron que con la implementación de su sistema lograron detectar la mayoría de los casos de pruebas esteganográficas.

Una investigación acerca de canales encubiertos en el protocolo MQTT v5.0 fue realizada por Mileva, et. al [74]. Esta investigación se encuentra relacionada con el trabajo de Velinov, et. al [46] y este trabajo es una actualización con respecto a la nueva versión del protocolo, dado que incluye nuevas características y nuevos campos. Encontraron que hay nuevos canales encubiertos que no eran viables para la versión anterior de MQTT, añadieron un nuevo patrón de ocultación que utiliza reconexiones. Su implementación la realizan con dos canales indirectos y evalúan su ancho de banda, la robustez, y su indetectabilidad.

En la tabla 3.1 se puede observar un resumen de los trabajos revisados y como se relacionan con los temas que se quiere abordar en el presente trabajo.

Título	Autor(es)	IdC	FPGA	Criptografía o esteganografía	Protocolo de comunicación
A lightweight scalable blockchain	Dorri, Kanhere, et al. 2019 [59]	✓			

Internet of things (IoT) security intelligence: a comprehensive overview, machine learning solutions and research directions	Sarker, Khan, et al. 2022 [60]	✓			
High-performance ECC processor architecture design for IoT security applications	Kudithi y Sakthivel, 2019 [61]	✓		✓	
User authentication and authorization framework in IoT protocols	Al-Refai y Alawneh, 2022 [62]	✓			
Security approach for LSB steganography based FPGA implementation	AlWatyan, et al. 2017 [4]		✓	✓	
BPCS Steganography for Data Security Using FPGA Implementation	Kait, Chauhan. 2015 [63]		✓	✓	
Message Concealment System of Voice Signals Implemented on FPGA	Martínez et al. 2016 [64]		✓	✓	
Enhanced TCP Sequence Number Steganography using Dynamic Identifier	Melo, Sison, Medina 2019 [67]			✓	✓
Secure data communication using protocol steganography in IPv6	Bobade 2015 [54]			✓	✓
The Solution of Key Transmission in Multi-Level Network Steganography	Xue et al. 2017 [68]			✓	✓
Yet Another Network Steganography Technique Based on TCP Retransmissions	Brodzki, Bieniasz 2019 [69]			✓	✓
Enhanced Data Encryption in IoT using ECC Cryptography and LSB Steganography	Geethanjali et al. 2021 [70]	✓		✓	

Securing Data in Internet of Things (IoT) Using Cryptography and Steganography Techniques	Khari et al. 2019 [7]	✓		✓	
Saturation Throughput Analysis of Steganography in the IEEE 802.11p Protocol in the Presence of Non-Ideal Transmission Channel	Almohammedi, Shepelev 2021 [72]	✓		✓	
Real-time RGB image encryption for IoT applications using enhanced sequences from chaotic maps	Trujillo-Toledo, et al. 2021 [71]	✓		✓	✓
Covert Channels in the MQTT-Based Internet of Things	Veilinov, et al. 2019 [46]	✓		✓	✓
How to make an intrusion detection system aware of steganographic transmission	Kosiak, et al. 2021 [73]	✓		✓	✓
Comprehensive analysis of MQTT 5.0 susceptibility to network covert channels	Mileva, et al. 2021 [74]	✓		✓	✓

Tabla 3.1: Trabajos relacionados

Capítulo 4

Descripción del sistema

En este capítulo se aborda una descripción del sistema propuesto en esta tesis. Mencionando primero cómo se utiliza la red del protocolo MQTT y cómo se debe hacer la conexión y las peticiones pertinentes para el funcionamiento de la red. Posteriormente se hace una descripción del sistema, en donde se menciona el objeto portador que se utiliza, y qué configuración del objeto se utiliza. Se describen dos esquemas como parte de la propuesta de este trabajo, el primero para ocultar la información a compartir, y el segundo para ocultar la llave de cifrado. Y finalmente se presenta la descripción de las funciones estego que se utilizaron para ocultar la información.

4.1. Red del sistema

El sistema se basa en la arquitectura publicador-suscriptor que utiliza el protocolo MQTT, en la figura 2.5 se puede observar como la red está compuesta por tres elementos, un cliente-publicador, un cliente-suscriptor y un dispositivo centralizado llamado *broker*. Se necesita tener un cliente que publique el mensaje y otro que lo reciba, el *broker* se encarga de direccionar los mensajes mediante el nombre del tema, así varios dispositivos pueden conectarse, pero solo aquellos que se suscriban al mismo tema podrán recibir el mensaje.

En la figura 2.6 se visualiza la forma en que el protocolo MQTT realiza la comunicación, primero el cliente-publicador envía el paquete de control CONNECT para establecer la conexión y una vez establecida se pueden enviar paquetes PUBLISH que contendrán la información que se desea compartir. Finalmente se envía un paquete DISCONNECT para indicar que se termina la conexión con el *broker*.

Es necesario tener un cliente-suscriptor, de manera similar que el cliente-publicador, primero se envía un paquete CONNECT para establecer la conexión con el *broker* y el cliente-suscriptor, después se envía un paquete SUBSCRIBE con el nombre del tema al que se quiere suscribir, esto indica al *broker* que se debe enviar la información que esté relacionada a ese tema, al dispositivo que se acaba de conectar. El cliente se queda esperando hasta que reciba un paquete PUBLISH por parte del *broker* para así recibir la información. Si es necesario terminar con la suscripción se envía un

paquete UNSUBSCRIBE y posteriormente un paquete DISCONNECT para indicar el término de la conexión.

Usando la arquitectura de red del protocolo MQTT, se diseña un esquema que pueda ocultar información cifrada en el protocolo MQTT. El enfoque que se pensó es ocultar información sensible en un campo del protocolo MQTT que lo permita, y así desarrollar funciones esteganográficas que ayuden a esta tarea. El sistema otorgará confidencialidad y ocultamiento a la información que se desea compartir a través del protocolo MQTT.

4.2. Esquema del sistema

La esteganografía consiste en ocultar información dentro de un objeto portador que puede ser texto, imágenes, audio, entre otros. Recientemente se utiliza como objeto portador los protocolos de comunicación y es de interés poder utilizar campos del protocolo que no se utilicen y que no afecten su funcionamiento. A continuación, se presenta un análisis del protocolo MQTT, específicamente del paquete PUBLISH, para encontrar un campo que permita el ocultamiento de información.

4.2.1. Paquete de control PUBLISH

El protocolo MQTT resulta interesante para trabajar, debido a que es el más utilizado entre los protocolos de comunicación enfocados en IdC como se vio en el capítulo 2. En ese mismo capítulo se observa que el protocolo MQTT v5.0 se conforma por 15 paquetes de control destinados para una tarea específica, que va desde iniciar una conexión, enviar o recibir mensajes, hasta la finalización de la conexión. De los paquetes de control el más utilizado es el paquete PUBLISH. Los demás paquetes en su mayoría se envían una vez en todo el proceso de comunicación. Por ejemplo, el paquete CONNECT se envía solo cuando se quiere iniciar la conexión con el *broker*, y el paquete CONNACK solo se envía cuando la conexión ha sido realizada de manera correcta, además este es enviado por el *broker*, y así con la mayoría de los paquetes. El paquete PINGREQ es enviado más veces para corroborar que el cliente sigue conectado, pero se envía cada intervalo de tiempo determinado y no es eficiente en la velocidad de envío de la información. El paquete PUBLISH se envía cada que se quiere compartir información y es normal tener un alto tráfico de este paquete, por lo que no es de extrañar si muchos paquetes son enviados en poco tiempo [2].

El paquete PUBLISH, al igual que los demás paquetes está conformado por un encabezado fijo, un encabezado variable y la carga útil. En la figura 4.1 se muestra como está conformado el encabezado fijo. Este encabezado tiene una longitud de 2 bytes. En el primer byte se encuentran los siguientes campos: en el primer bit se encuentra el campo RETAIN que indica si el mensaje va a ser retenido para que quede guardado o no en el *broker*, de los bits 1 al 2 está el campo nivel de QoS qué índica con que nivel de calidad será enviado el mensaje, el campo indicador DUP que establece si el paquete PUBLISH ya a sido enviado y de ser así se intenta volver a

entregar dicho paquete en caso de un fallo en el envío, y en los bits 4 al 7 el campo tipo de paquete de control, que toma un valor de 0011 en número binario para indicar que es un paquete de tipo PUBLISH. El segundo byte indica la longitud restante del paquete [2].

Bit	7	6	5	4	3	2	1	0
Byte 1	Tipo de control de paquete MQTT(3)				Indicador DUP	Nivel de QoS		RETAIN
	0	0	1	1	X	X	X	X
Byte 2	Longitud restante							

Figura 4.1: Encabezado fijo del paquete PUBLISH

El siguiente elemento es el encabezado variable. Para cada paquete el encabezado variable contiene diferentes campos. El encabezado variable del paquete PUBLISH se muestra en la figura 4.2, en ella se pueden observar como primer elemento la longitud del nombre del tema, el tipo de dato que maneja es un entero sin signo de 16 bits, que se encuentra separado en dos bytes, el primero contiene el byte más significativo (MSB) y el segundo el byte menos significativo (LSB). Posteriormente se encuentra el campo de nombre del tema, que almacena como su nombre lo indica el nombre del tema y el tipo de dato que maneja es una cadena codificada en UTF-8 y puede tener una longitud de 0 a 65 535 bytes. El siguiente campo es el identificador de paquete, en este campo se deja un identificador que en caso de que se utilice una calidad de servicio QoS 1 o QoS 2 este identificador sirve para formar un único conjunto por separado entre el cliente y el *broker*, este paquete se conforma de dos bytes y al igual que la longitud del nombre del tema utiliza un tipo entero de 16 bits sin signo y se divide en MSB y LSB. Como últimos dos campos se encuentran la longitud de propiedades que es un entero de byte variable y las propiedades, estas pueden estar o no presentes dependiendo de las propiedades que se utilicen en el paquete [2].

Los campos anteriormente descritos tienen una función que no puede ser cambiada, por tal motivo no pueden ser utilizadas para ocultar información, ya que un cambio en su valor altera por completo la funcionalidad del paquete. Es necesario evaluar los campos presentes en las propiedades y seleccionar uno que pueda ser utilizado para ocultar información.

En las propiedades del paquete PUBLISH, mostrado en la figura 4.3, se encuentra los siguientes campos. Indicador de formato de la carga útil, en este campo se utiliza un byte para indicar si la carga útil tiene bytes no especificados (0x00) o si la carga útil tiene datos en formato UTF-8. Intervalo de caducidad del mensaje, este campo está conformado por cuatro bytes y utiliza un tipo entero de 32 bits sin signo, indica que si ha pasado el tiempo establecido y el servidor no ha logrado iniciar la entrega del paquete a un suscriptor coincidente, se debe eliminar el mensaje para ese suscriptor. Alias del tema, este campo tiene un tamaño de dos bytes y utiliza el tipo entero de 16 bits sin signo, se utiliza para aligerar el tamaño del paquete utilizán-

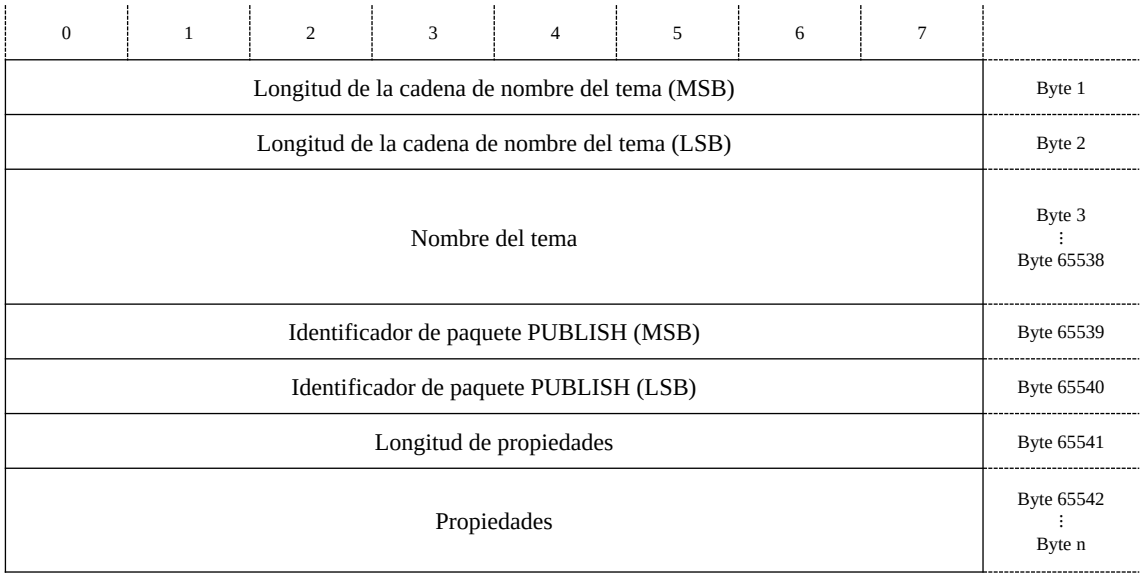


Figura 4.2: Encabezado variable del paquete PUBLISH

dolo en lugar del nombre del tema. Tema de respuesta, el tipo de dato que utiliza es una cadena de caracteres codificada en UTF-8, este campo se utiliza para enviar un mensaje de respuesta a todos los suscriptores que reciben el mensaje. Datos de correlación, el tamaño utilizado es un byte y se usan los bits que lo componen, se utiliza para identificar para que solicitud es el mensaje de respuesta cuando se recibe. Propiedad del usuario, utiliza una cadena de caracteres codificados en UTF-8, puede aparecer varias veces para representar varios pares de nombre y valor cuyo significado e interpretación solo conocen los programas de aplicación responsables de enviarlos y recibirlos. Identificador de suscripción utiliza un formato de entero variable, utilizado por MQTT, este campo puede tener un valor de 1 a 268 435 55 y como su nombre lo indica representa el identificador de la suscripción. Tipo de contenido, usa una cadena de caracteres codificada en UTF-8, en este campo se puede describir el contenido del mensaje que se está enviando [2].

Evaluando los campos descritos anteriormente, se puede hacer un análisis de cuál es el mejor campo para ocultar información. La información que regularmente se envía por MQTT es texto y por tanto los campos que utilizan caracteres pueden ser muy predecibles en cuanto a ocultar la información, debido a que, si se hace de forma directa la inserción, se puede observar el contenido del campo y no se lograría el objetivo de ocultar información. Quedan los campos que utilizan un tamaño determinado de bits estos utilizan un tipo numérico y pueden ser ideales si se desarrolla de forma correcta una función estego que permita ocultar información, el único campo disponible que no tenga un uso específico es el campo intervalo de caducidad del mensaje, debido a que no afecta si se cambia el valor que contiene y no afecta el funcionamiento del paquete.

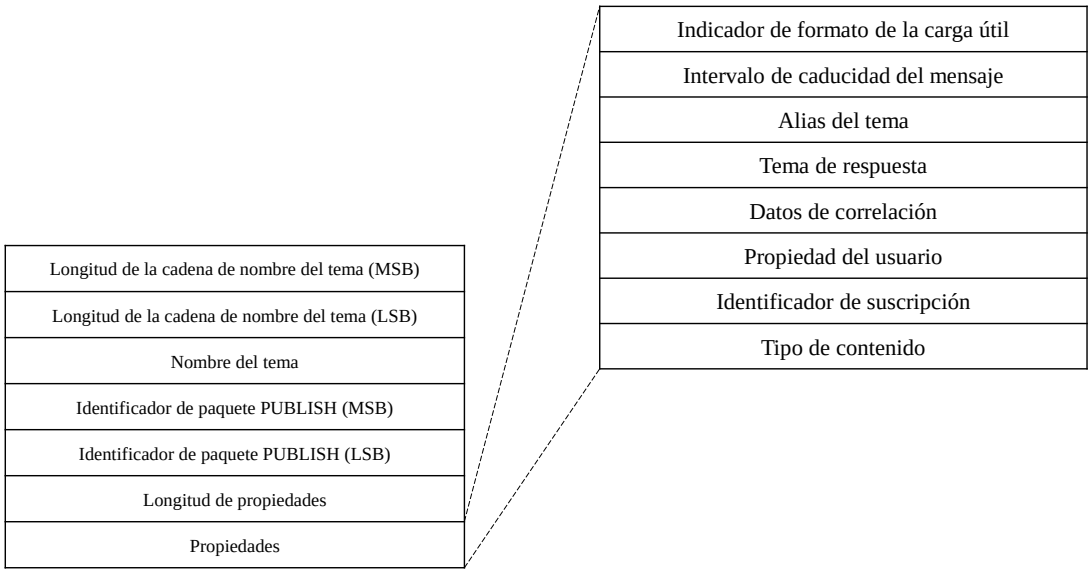


Figura 4.3: Propiedades del paquete PUBLISH

4.3. Funciones estego

La función estego inserta la información a ocultar dentro del objeto portador, esta función debe brindar indetectabilidad, robustez y capacidad para asegurar un correcto ocultamiento de la información. Con base a lo descrito anteriormente se propone diseñar e implementar una función estego utilizando como objeto portador el datagrama del protocolo MQTT v5.0 en el campo intervalo de caducidad del mensaje, tomando en cuenta todas las características del objeto portador y de la función estego para una correcta implementación. A continuación, se muestran dos funciones desarrolladas para este trabajo.

4.3.1. Función estego bit menos significativo

Para el desarrollo de esta función estego se toma en cuenta que no debe ser complicada en términos de operaciones, debido al tipo de dispositivos con los que se quieren trabajar. Por lo tanto, basándose en una técnica utilizada para la esteganografía de imágenes, en donde se oculta la información bit a bit en el bit menos significativo de los píxeles que conforman la imagen, esto se hace para no deformar de manera evidente la imagen original. En el campo intervalo de caducidad del mensaje del paquete PUBLISH, se inserta la información en el bit menos significativo de los 32 bits de este campo. Así se descompone la información en bits y cada bit se inserta en un paquete PUBLISH para ser enviado.

En la figura 4.4 se observa de forma gráfica el funcionamiento de esta función. La cadena “Hola a todos, esta es una prueba”, se descompone en bits, para ejemplificar esto se toma el caracter ‘H’ y se descompone en bits que equivale al número en binario “01001000”, estos bits se insertan en el bit menos significativo del campo y en cada

paquete se envía un bit de la información.

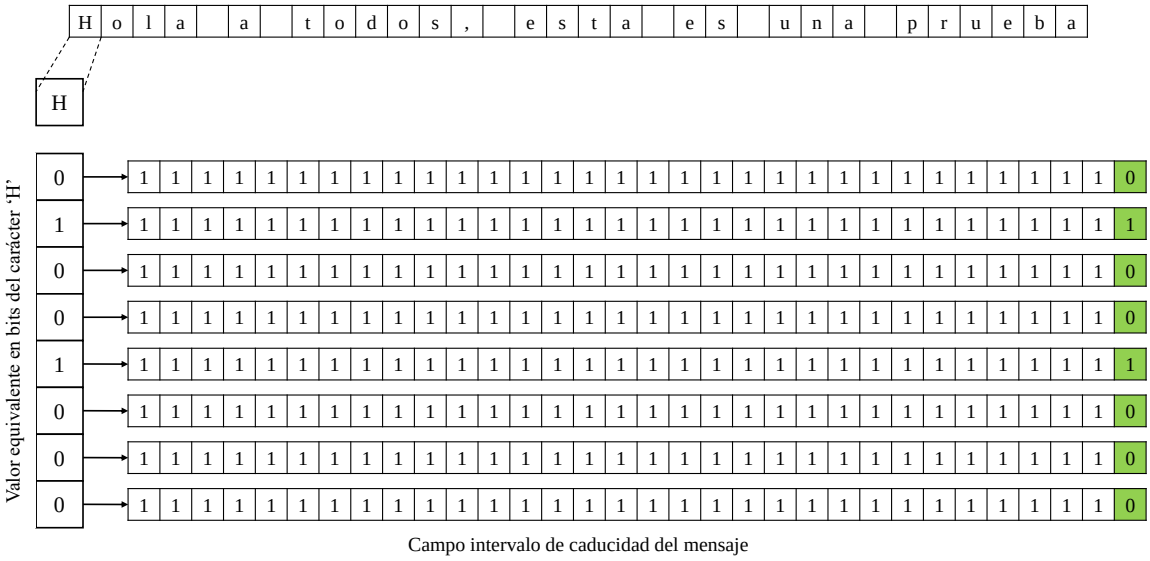


Figura 4.4: Diagrama de la función estego bit menos significativo

4.3.2. Función estego bit menos significativo *nibble*

Esta función toma como base la función anterior. Para este caso se dividen los 32 bits en bloques de 4 bits (*nibble*) y en cada bit menos significativo de cada *nibble* se inserta un bit de información a ocultar. Con esto se busca usar menos paquetes para enviar la información y así aumentar su capacidad. En la figura 4.5 se muestra cómo se realiza esta función. Usando el texto "Hola a todos, esta es una prueba" se separa la información en bits, para el ejemplo se toma el caracter 'H' y se descompone en bits que corresponde al número "01001000", y en cada bit menos significativo de los *nibbles* se inserta un bit de información.

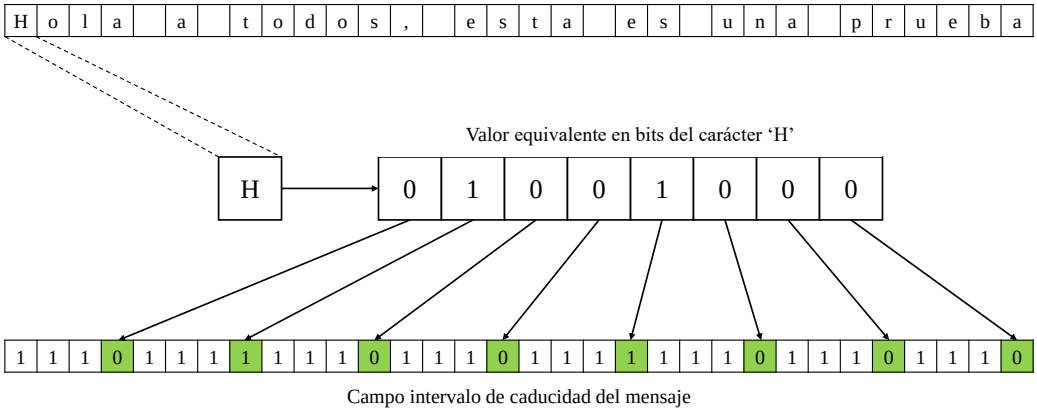


Figura 4.5: Diagrama de la función estego bit menos significativo *nibble*

4.4. Esquemas del sistema

El sistema se diseña para utilizar un cifrador, una función estego, el objeto portador, una función estego inversa y un descifrador. Con estos elementos se diseñaron dos esquemas. El primero se enfoca en ocultar la información sensible que se quiere compartir. El segundo en ocultar la llave que se usa para cifrar la información. La llave de cifrado posee una dificultad al ser compartida, debido a que si algún tercero no autorizado logra conocer la llave podrá acceder a la información que se está compartiendo, por eso es necesario lograr compartir la llave por un medio seguro.

4.4.1. Esquema de ocultamiento de información

El esquema mostrado en la figura 4.6, se representa en bloques los elementos que constituyen el sistema propuesto. Con este sistema se busca ingresar la información, cifrarla y ocultarla. El primer paso es ingresar la información en el cifrador, para el caso de este trabajo es el cifrador SIMON o SPECK dependiendo del dispositivo. Estos cifradores requieren una llave para realizar el cifrado y por tanto se define una que será previamente compartida para que ambos clientes tengan el conocimiento de la llave. Después mediante la función estego la información que fue previamente cifrada se inserta en el objeto portador, es este caso el datagrama del protocolo MQTT, y así se podrá enviar la información de forma oculta. El paquete PUBLISH de MQTT es enviado al *broker* que a su vez lo envía al cliente-suscriptor que recibe el paquete. Una vez que se ha recibido el paquete se procede a utilizar la función estego inversa para recuperar la información que fue ocultada en el datagrama del paquete PUBLISH de MQTT. Esta información se va almacenando hasta que se termina de recibir la información completa. Cuando la información se ha terminado de recibir se procede a separarla en bloques y así proceder con el descifrado de la información usando la llave previamente compartida. Una vez que se termina con el descifrado se obtiene la información original.

4.4.2. Esquema de ocultamiento de llave de cifrado

Para el cifrado simétrico se necesita compartir la llave para utilizarla en el descifrado. Es necesario que al compartir la llave se haga con sumo cuidado, y así evitar que terceros no autorizados logren tener conocimiento de la llave, ya que si se logra conocer se puede acceder a la información y eso es algo no deseado. Por tal motivo se diseñó un esquema que permita compartir la llave de forma oculta usando esteganografía. En la figura 4.7 se observa el diagrama del esquema que realiza el ocultamiento de la llave. Primero la información pasa por el cifrador y usando la llave se cifra la información y se coloca en la carga útil del paquete PUBLISH. La llave que se utiliza se oculta dentro del paquete PUBLISH con la ayuda de la función estego y se envían los paquetes necesarios para compartir todo el tamaño de la llave. Una vez que se ha terminado de enviar toda la información de la llave de cifrado se procede a obtener

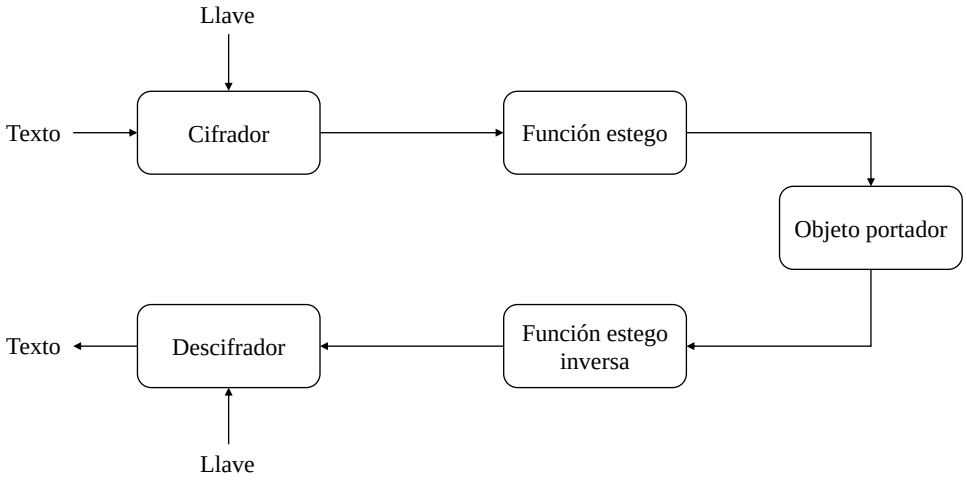


Figura 4.6: Diagrama del esquema para ocultar la información a compartir

la llave que se ocultó con el uso de la función estego inversa. Cuando se tiene la llave y la información, se descifra la información para así obtener el mensaje original.

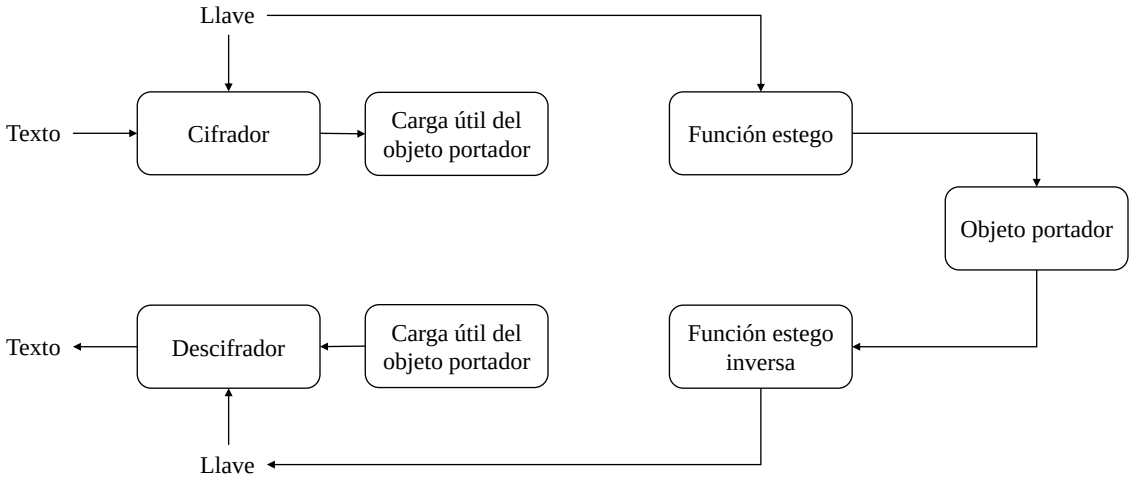


Figura 4.7: Diagrama del esquema para ocultar la llave de cifrado

La gran diferencia entre ambos esquemas es la información que se oculta. Ambos esquemas buscan compartir de forma segura un tipo de información y así evitar una posible fuga de información. Ambos esquemas comparten el objeto portador que es el protocolo MQTT y la función estego que se utiliza es la misma. En el primer esquema se busca obtener un sistema de seguridad que nos permita intercambiar información de forma segura aplicando confidencialidad y ocultamiento a la información que se quiere intercambiar. En el segundo esquema se busca aplica seguridad a dos elementos de información, por un lado, aplicar confidencialidad a la información que se quiere compartir y ocultación a la llave de cifrado.

Capítulo 5

Implementación

La implementación del cliente-publicador en hardware se hace en una tarjeta PYNQ Z2 que posee un procesador Cortex-A9 Dual Core ARM y una FPGA de 1.3 M de puertas reconfigurables. En la implementación en software, se utiliza únicamente el procesador para la ejecución del programa, este se encargará de realizar el cifrador de la información, crear el paquete del protocolo MQTT, agregar la información en el campo correspondiente y enviar el paquete al destinatario deseado.

El procesador necesita un sistema para operar. El sistema que Xilinx ofrece es petalinux, esta herramienta brinda un sistema operativo basado en Yocto. Este sistema operativo ofrece un entorno en Linux. Al ser una versión simplificada hay que seleccionar los elementos que se quieren tener instalados desde la configuración del sistema.

Para la implementación completa en software se utiliza una Raspberry Pi 4 que contedrá el cliente-publicador y se encargará de realizar el cifrado, la función estego, la creación del paquete MQTT y el envío de dicho paquete. El sistema instalado es Raspberry Pi OS que está basado en Linux y la conexión será mediante el puerto Ethernet.

El *broker* se utilizará de dos maneras. Primero será alojado junto con el cliente-suscriptor en una computadora de escritorio donde se recibirán todos los paquetes que se enviarán de los dispositivos mencionados anteriormente. La segunda forma será utilizar un *broker* público para asegurar que se puede enviar información a la nube y recuperar sin problemas.

5.1. Petalinux

Petalinux es un kit de desarrollo de software (SDK) de Linux integrado dirigido a diseños de sistema en un chip (System on Chip) basados en FPGA. Permite personalizar, construir e implementar soluciones de Linux embebido en los sistemas de procesamiento de Xilinx. Diseñada a medida para acelerar la productividad del diseño, la solución funciona con las herramientas de diseño de hardware de Xilinx para facilitar el desarrollo de sistemas Linux para Versal, Zynq UltraScale MPSoC, SoC

Zynq-7000 y MicroBlaze.

Para el desarrollo del sistema primero se debe desarrollar el esquema completo que involucra el procesador y el FPGA para que se realicen las conexiones necesarias que sirvan como puente de enlace entre el procesador y el FPGA. El enlace se realiza por medio del software Vivado. Vivado se encarga de realizar las conexiones internas necesarias para habilitar la comunicación de información entre el procesador y el FPGA.

En Vivado se crea un nuevo diseño y se agrega el bloque del procesador ZYNQ. Posteriormente se crea un nuevo bloque IP que contendrá la configuración de hardware para la implementación de la función estego. También se crea un nuevo bloque IP que tendrá la configuración de hardware del cifrador SIMON. Se agregan los dos bloques creados al diseño y se selecciona la conexión automática de los bloques agregados.

Una vez que los bloques han sido creados y se han conectados, se procede a realizar una síntesis del diseño para corroborar que no hay fallos en la configuración y conexión del diseño. Una vez que la síntesis se realizó sin fallos se procede a exportar el diseño del hardware para poder crear la configuración de petalinux necesaria.

Ahora que se tiene la configuración de hardware se procede a configurar y construir el proyecto de de petalinux que se utilizará para el sistema. Esto se puede observar en el apéndice B.

5.2. Clientes MQTT

El sistema operativo del procesador se encuentra en funcionamiento, ahora se desarrollarán los clientes publicador y suscriptor para realizar la comunicación. Para la creación de los clientes es necesaria una biblioteca que brinde los recursos necesarios para crear la conexión, además se busca que pueda trabajar con la versión 5.0 de MQTT, dado que los campos seleccionados para ocultar información solo existen en esta versión. El software seleccionado es Mosquitto que nos brinda el *broker*, y las bibliotecas para crear clientes que utilicen la versión 5.0 de MQTT.

Mosquitto

Mosquitto proporciona implementaciones de servidor y cliente que cumplen con los estándares del protocolo MQTT. Mosquitto está diseñado para usarse en todas las situaciones en las que se necesita una mensajería ligera, particularmente en dispositivos restringidos con recursos limitados. Mosquitto es un proyecto de Eclipse Foundation. Mosquitto consta de tres partes; el servidor principal de Mosquitto o *broker*, los clientes mosquitto_pub y mosquitto_sub que sirven para comunicarse con un servidor MQTT y por último una biblioteca para clientes MQTT escrita en C [75].

5.2.1. Cliente-publicador

Para desarrollar el cliente-publicador se siguen los siguientes pasos. Primero se debe ingresar la información que se va a cifrar, luego se deben iniciar las variables que almacenará el texto que se ira dividiendo en bloques. Posteriormente se inicializa la biblioteca de Mosquitto y se configura para que los paquetes que se utilizan sean de la versión 5.0 de MQTT. Para realizar la conexión entre el cliente y el *broker* lo primero es enviar un paquete CONNECT, donde se especifica la dirección IP del *broker* y el puerto que se utiliza. Cuando el *broker* envía un paquete CONNACK para informar que se ha establecido conexión, se ejecuta una función que nos informa de dicho evento en donde se programa un mensaje para informar de la conexión exitosa.

Ahora que la conexión se ha establecido, se procede a separar por bloques la información, esto depende de la configuración del cifrador, para esta implementación se utilizó la configuración 32/64 para ambos cifradores SIMON y SPECK, esto debido a que la información que se desea transmitir es pequeña, se quieren compartir datos muy específicos como nombres o edades. La información del tamaño del bloque se cifra y se utiliza el modo *Cipher block chaining* (CBC) para el cifrador por bloques. En la figura 5.1, se muestra el cifrado mediante el modo de operación CBC. Este consiste en ingresar un bloque del texto que se quiere cifrar y realizar una operación XOR con un vector de inicialización. El vector de inicialización (IV) se conforma de una parte generada de manera aleatoria y usando el modo contador para cada bloque del texto. El resultado de la operación XOR pasa al cifrador para finalmente obtener el bloque cifrado. Para los siguientes bloques se utiliza el bloque cifrado anterior en lugar del bloque IV para generar los nuevos bloques cifrados.

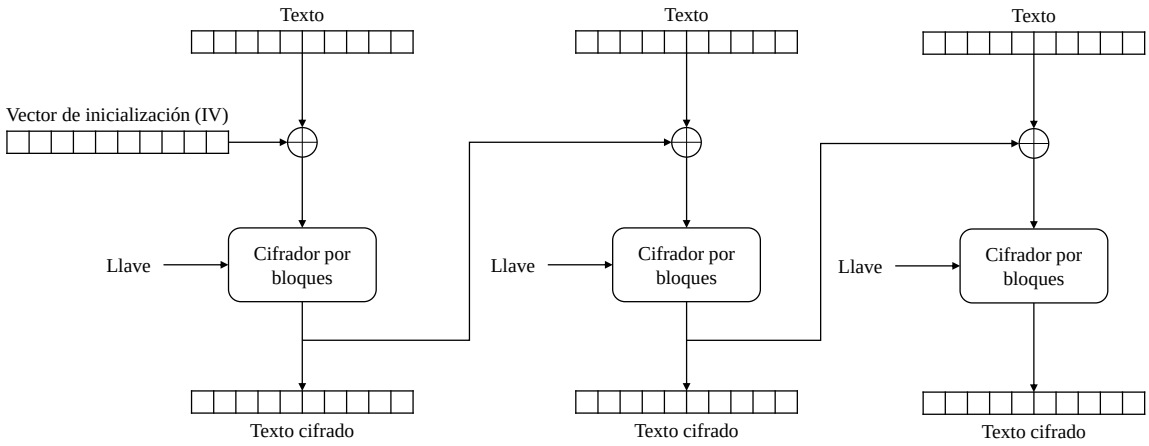


Figura 5.1: Diagrama del modo de operación CBC

Cada bloque de la información cifrada pasa a la función estego Bit menos significativo *nibble*, en donde se calcula un valor aleatorio entero de 32 bits para el campo intervalo de caducidad del mensaje y se utiliza la función estego Bit menos significativo *nibble* que divide un dato de 32 bits en 8 *nibbles* y almacena un bit de información en cada bit menos significativo de los *nibbles*.

Con la información cifrada se agrega el valor entero a una variable que tiene el formato para las propiedades variables del protocolo MQTT. Posteriormente se publica el mensaje, estableciendo el nombre del tema al que se va a publicar, el mensaje que se quiere publicar y la lista que contiene las propiedades variables, se ejecuta la función que nos permite saber que se ha enviado y se ha recibido o si hubo un error en el mensaje. Así llega el mensaje al destinatario que se ha suscrito al mismo tema y se procede a recibir y visualizar el mensaje que se cifró y ocultó. En el algoritmo 1 se observa el pseudocódigo del cliente-publicador.

Algoritmo 1 Pseudocódigo cliente-publicador

Entrada: Información a compartir, llave para cifrado, configuración del cifrador
Salida: Información cifrada, paquetes PUBLISH

1: Ingresar información a ocultar
2: Iniciar biblioteca Mosquitto
3: Crear paquete MQTT
4: Enviar paquete CONNECT
5: Separar información en bloques de tamaño n
6: Cifrar la información
7: Generar número aleatorio de 32 bits
8: Aplicar función stego
9: Enviar paquete PUBLISH con la información oculta
10: Enviar paquete que indique el termino del envío de la información
11: Enviar paquete DISCONNECT

5.2.2. Cliente-suscriptor

El cliente-suscriptor debe seguir el siguiente orden. Primero se inicializa la biblioteca de Mosquitto y posteriormente se indica que se trabaja con la versión 5.0 de MQTT. Se establece la conexión mediante el envío del paquete CONNECT con la dirección IP del *broker* y el puerto que se utilizará para la comunicación, y si se realizó correctamente se ejecuta la función que muestra un mensaje. Una vez que la conexión se ha establecido se manda el paquete SUBSCRIBE para establecer la suscripción al tema que se desea. El tema debe ser el mismo al que el cliente-publicador envía la información, así todos los clientes que estén conectados reciben el mensaje en la carga útil, pero solo el cliente que tenga el algoritmo para extraer la información oculta y el descifrador puede visualizar el texto que es envió de forma oculta.

El cliente-suscriptor se queda en espera de recibir la información y cuando se detecta el envío de un paquete PUBLISH se ejecuta una función para realizar una acción. Para el sistema se envía información y el cliente-suscriptor recibe dicha información, en cada paquete se extraen los bits correspondientes a la información que se envía de forma oculta y se van almacenando, el cliente se detiene cuando en el campo intervalo del mensaje recibe un valor específico, en este caso el valor 169 que

corresponde al carácter ®en código ASCII. Cuando se recibe este valor el cliente da por hecho que se ha compartido toda la información y procede a realizar el descifrado de la información. Cuando la información ha terminado de ser descifrada se puede visualizar la información original que se compartió.

Algoritmo 2 Pseudocódigo cliente-suscriptor

Entrada: Paquetes PUBLISH, llave de cifrado, configuración del cifrador
Salida: Información original

```
1: Iniciar biblioteca Mosquitto
2: Crear paquete MQTT
3: Enviar paquete CONNECT
4: Enviar paquete SUBSCRIBE
5: while No se recibe paquete PUBLISH do
6:     Queda en espera
7: end while
8: if Se recibe un paquete PUBLISH que no indica el termino de la información
   then
9:     Se extrae el dato usando la función estego inversa
10:    Se añade lo obtenido a la información previamente obtenida
11: else
12:    Separar la información recibida en bloques de tamaño n
13:    Se descifra la información
14:    Visualización de la información original
15: end if
```

5.3. Función estego software

La función estego que se utiliza para la implementación es bit menos significativo *nibble*. En software esta función se implementa primero calculando un valor numérico aleatorio de 32 bits para el campo intervalo de caducidad del mensaje. Posteriormente se aplica una operación AND con el número en hexadecimal "EEEEEEEE" y dejar con un valor de '0' binario el bit menos significativo de cada *nibble* que componen el campo de 32 bits. Finalmente se insertan los bits de la información en el valor calculado para ocultar la información.

Algoritmo 3 Pseudocódigo función bit menos significativo *nibble*

Entrada: Información de entrada
Salida: Intervalo aleatorio con información insertada

```
1: Generar aleatoriamente un valor de 32 bits
2: Operación AND entre el valor aleatorio y el número en hexadecimal "EEEEEEEE"
3: Insertar la información bit por bit en el bit menos significativo de cada nibble
```

5.4. Función estego hardware

En hardware la función bit menos significativo *nibble* se implementa de la siguiente forma. Primero se declaran cinco bloques de 32 bits para la entrada, uno para la información que se va a ocultar y los otros cuatro bloques son cuatro valores aleatorios diferentes en donde se insertará la información. En FPGA el manejo de bits es más directo y se puede ocultar más información en 4 cuatro paquetes al mismo tiempo dada la ejecución concurrente de la FPGA. El valor aleatorio se calcula cuatro veces en software y a continuación la inserción en bits se hace en hardware. En la figura 5.2 se observa el diagrama de la función estego desarrollada en hardware.

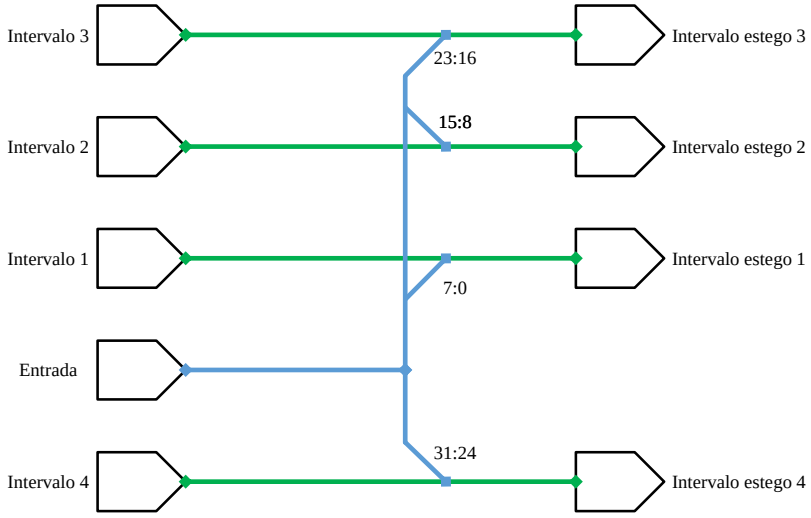


Figura 5.2: Diagrama de la función bit menos significativo en hardware.

5.5. Cifrador SIMON software

En software la implementación del cifrador SIMON se realizó en forma de biblioteca, en este caso para la configuración SIMON64/32. La función recibe como parámetros la configuración como un valor entero, el apuntador del bloque de información del tamaño de la configuración seleccionada y el apuntador al bloque que contiene la llave para cifrar la información.

Siguiendo el diagrama de la red de Feistel el bloque de información de entrada se separa en dos. Se inicializa los bloques que almacenaran las llaves de ronda con la llave que se ingresa. Se calculan las operaciones que se observan en la red de Feistel para obtener la información cifrada. La información se devuelve con un apuntador que contiene la información cifrada. Ahora que la información ha sido cifrada se puede enviar sin ningún problema a través del protocolo MQTT.

5.6. Descifrador SIMON

Para el descifrar la información mediante el algoritmo SIMON se debe tener en cuenta que al generar llaves de ronda se necesita la última generada para lograr obtener la información original. Para descifrar la información se necesita generar la última llave para ir en retroceso e ir obteniendo en orden inverso las llaves de ronda. Para ello el diagrama para generar la llave de ronda se modifica para ir recuperando las llaves en la ronda correspondiente.

El diagrama que se muestra en la figura 5.3, se observa el flujo del desplazamiento de los registros va en dirección contraria y que se quiere calcular el valor de la llave k_i. Las ecuaciones que describen este diagrama es la siguiente.

$$k_i = \begin{cases} c \oplus (z_j)_i \oplus k_{i+2} \oplus (I \oplus S^{-1})S^{-3}k_{i+1}, & \text{si } m = 2 \\ c \oplus (z_j)_i \oplus k_{i+3} \oplus (I \oplus S^{-1})S^{-3}k_{i+2}, & \text{si } m = 3 \\ c \oplus (z_j)_i \oplus k_{i+4} \oplus (I \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}), & \text{si } m = 4 \end{cases} \tag{5.1}$$

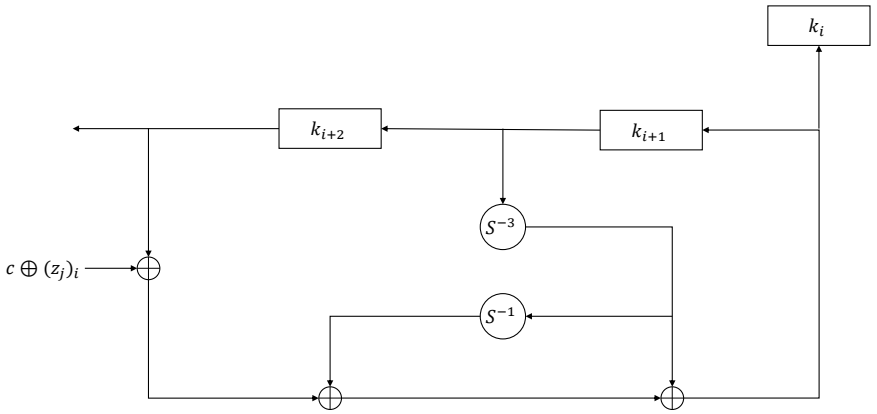
Para representar con mayor claridad el diagrama se agregan los bloques con etiqueta k_i para señalar la llave que será la próxima a calcularse.En las ecuaciones anteriores k_{i+4}, k_{i+3} y k_{i+2} representan las últimas llaves de ronda dependiendo del número de bloques es que se separan las llaves para el cifrado. La llave que se necesita calcular es k_i ya que es una llave anterior a k_{i+1} y no la siguiente a k_{i+m}. El proceso es simular el cifrado, pero cambiando el sentido del desplazamiento de los registros.

5.7. Cifrador SIMON hardware

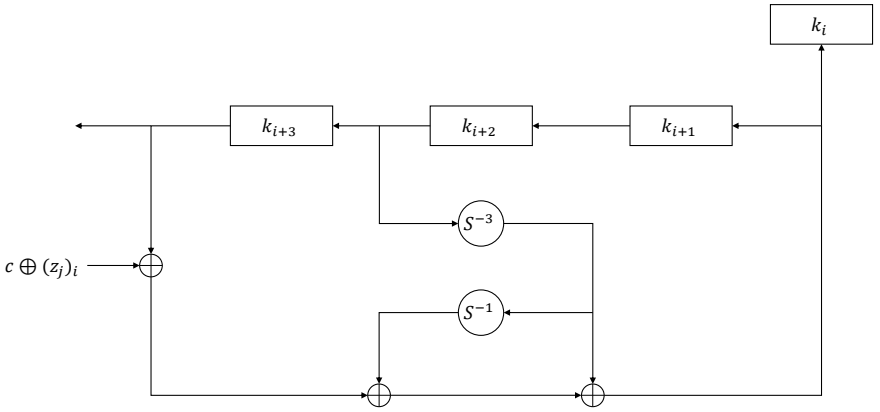
El cifrador SIMON implementado en hardware se basa en el diagrama de estados mostrado en la figura 5.4. El primer estado es el RESET, este estado se presenta cada vez que el sistema es reiniciado. El siguiente estado es IDLE que es el estado inactivo del sistema y cada que se acaba un estado se regresa a él. Si se está en el estado IDLE y la configuración del cifrador es "11" se inicia el cifrado de la información en el estado CIPHER_START. CIPHER_RUN es el estado siguiente y se encarga de verificar que se cumpla el número de rondas necesarias para la configuración, una vez que se cumplen el número de rondas se procede a ir al siguiente estado. CIPHER_FINISH_1 y CIPHER_FINISH_2 son los estados siguientes y se utilizan para realizar los últimos movimientos del cifrado y así obtener el valor cifrado de la información. CIPHER_LATCH es el último estado y se encarga de guardar el cifrado de la información en los registros de salida del cifrador.

En la figura 5.5 se muestra el diagrama implementado en FPGA de la red de Feistel en donde se utilizan compuertas AND y XOR así como desplazamientos a la derecha y los tamaños de datos manejados son de 16 bits y el bloque k_i se calcula con el generador de llaves de ronda.

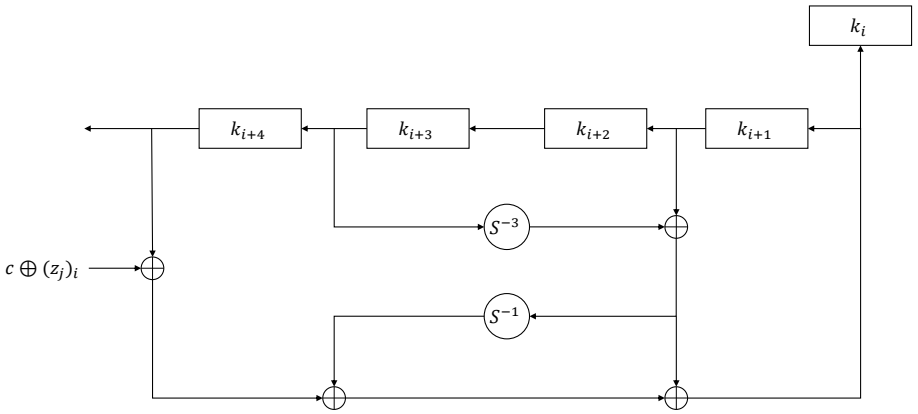
La generación de llaves de ronda se implementa en FPGA y la figura 5.6 se observa el diagrama de su implementación. Se implementa con operaciones XOR, desplazamientos a la derecha de tres y un bit y con constantes para cada ronda.



(a) $m = 2$ palabras de llave



(b) $m = 3$ palabras de llave



(c) $m = 4$ palabras de llave

Figura 5.3: Diagrama para la generación de la llave de ronda en la función de ronda para el descifrado SIMON

El cifrador funciona un determinado número de rondas. El cifrador habilita la generación de llaves para cada ronda y este a su vez genera la llave de ronda para que el cifrador utilice esta llave para realizar las operaciones pertinentes. El número de

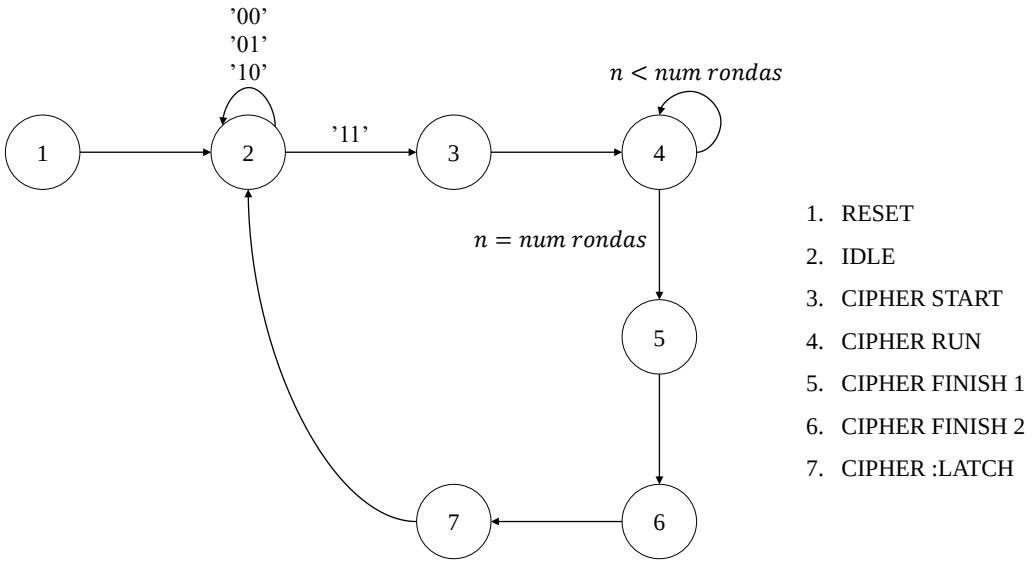


Figura 5.4: Diagrama de estados del cifrador SIMON en hardware

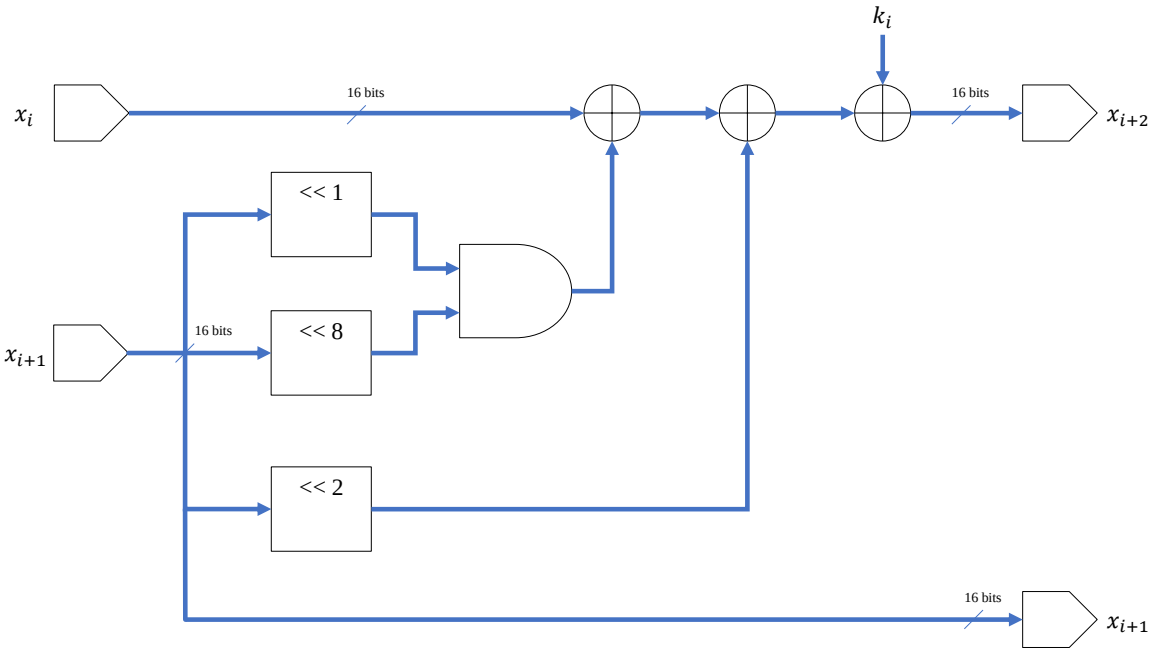


Figura 5.5: Red de Feistel implementado en FPGA.

rondas totales de operación debe ser igual al número de rondas - 2, que se muestra en la tabla 2.4 que depende de la configuración del cifrador, y así terminar con el proceso de cifrado. En la figura 5.7 se muestra el diagrama de lo descrito anteriormente.

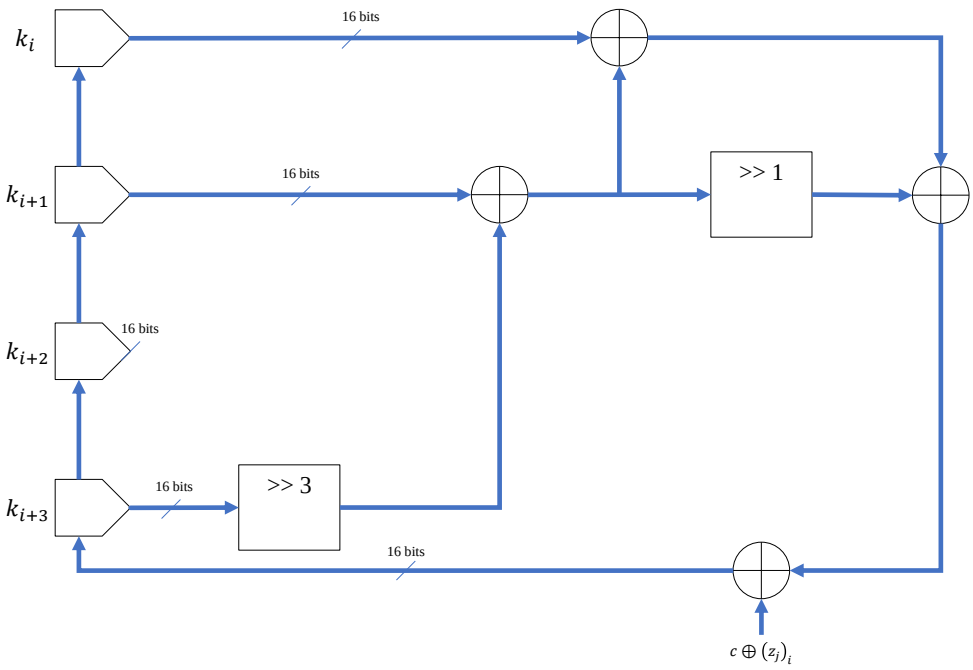


Figura 5.6: Generación de llaves de ronda en FPGA.

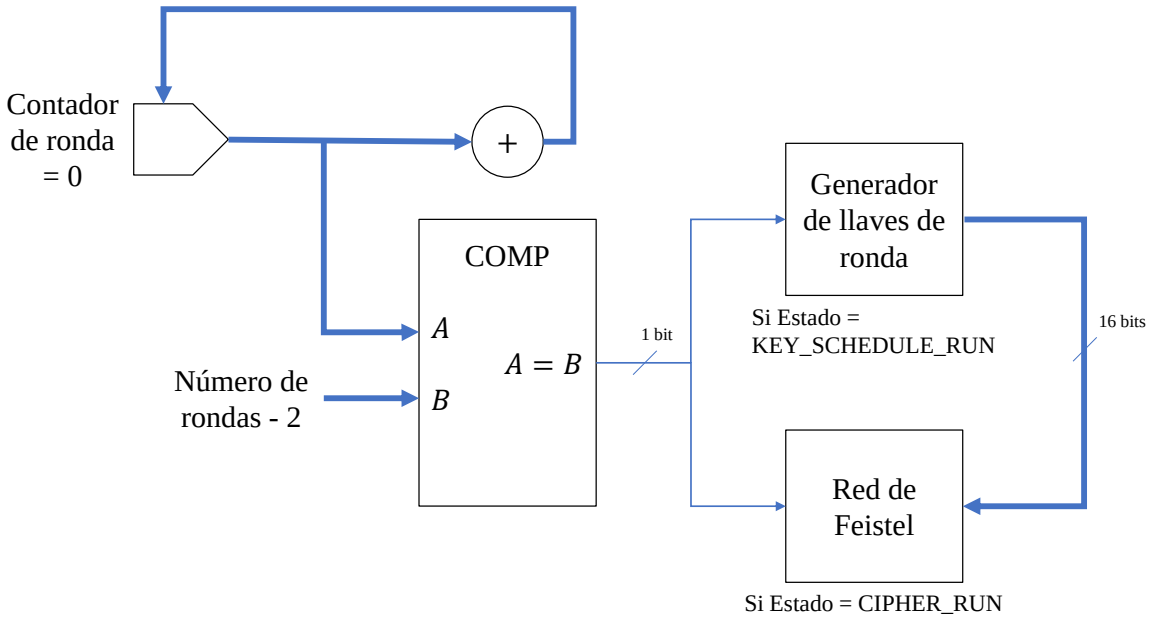


Figura 5.7: Funcionamiento del cifrador SIMON en FPGA.

5.8. Cifrador SPECK software

El cifrador SPECK se programó, al igual que el cifrador SIMON, como una biblioteca. La función utiliza tres parámetros, el primero es la configuración con la que trabajará

el cifrador, el segundo un apuntador a la información que se desea cifrar y por último un apuntador a la llave que se necesita para el cifrado. A continuación, la información se separa en dos registros de entrada a la red de Feistel y la llave se agrega a los registros para su separación dependiendo de la configuración utilizada. Posteriormente se aplica la red de Feistel que se muestra en la figura 2.11 y aplicar el diagrama de la figura 2.12 para calcular la llave de ronda de cada ronda. Finalmente se regresa el valor cifrado para ese bloque de información y se procede a realizar el cifrado del siguiente bloque de información o continuar con lo siguiente si no hay información.

5.9. Descifrador SPECK software

Para el descifrado SPECK se sigue el flujo contrario de la red de Feistel para el cifrado. En la figura 5.8 se puede observar el diagrama que describe la red de Feistel para el descifrado. Las rotaciones de bits se realizan de manera inversa, es decir si en el cifrado se realizó una rotación a la derecha, para el descifrado se debe realizar una rotación a la izquierda en la misma cantidad de bits. En lugar de la adición módulo 2^n, se realiza una sustracción módulo 2^n y las operaciones XOR se conservan solo cambiando los datos con los que se realiza la operación.

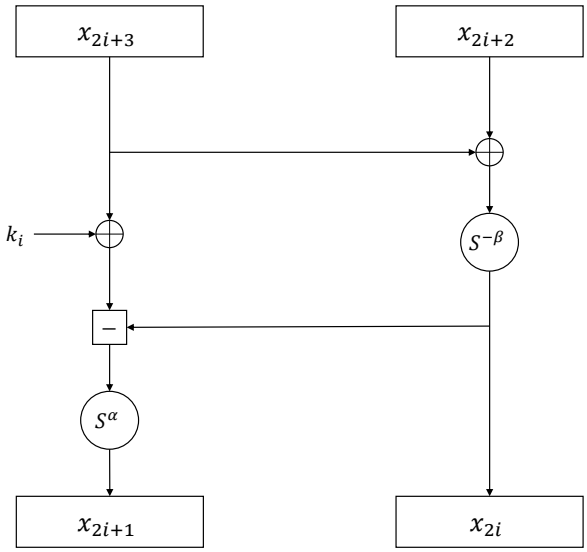


Figura 5.8: Red de Feistel para el descifrado de SPECK.

En el caso de la generación de llave de ronda, al utilizar la red de Feistel, diagrama mostrado en la figura 2.12 se denota como R_i, se utiliza también el diagrama de la figura 2.11 para realizar la generación de llaves de ronda de forma inversa. Una vez que se han calculado las llaves de ronda se procede a seguir con el algoritmo para descifrar y así obtener la información original.

5.10. Sistema embebido

En la implementación por hardware se utiliza un sistema embebido. Este sistema utiliza un procesador que tiene conexión con una FPGA. Para el sistema propuesto el algoritmo de cifrado SIMON y la función estego se implementan en la parte de la FPGA, mientras la parte del cliente-publicador se implementa en el procesador. Los datos son transferidos a través de bloques de 32 bits para el ingreso a la FPGA y su salida. Este diagrama se muestra en la figura 5.9.

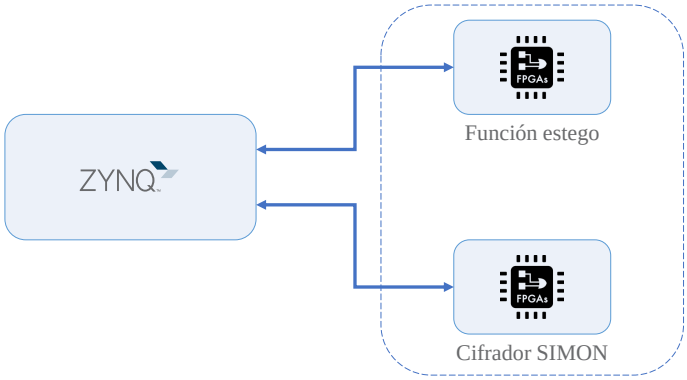


Figura 5.9: Diagrama del sistema embebido para el sistema.

5.11. *Broker* público

El *broker* se implementa en una computadora de escritorio para poder registrar a los paquetes recibidos y realizar un análisis. En la actualidad los servicios en la nube son de gran importancia dado a la gran cantidad de información que se comparte, por tal motivo es necesario lograr una conexión con un *broker* público. Un *broker* público permite realizar una conexión de varios usuarios a través de un servidor en la nube, de esta manera cualquiera puede tener acceso a este servidor.

Existe una gran variedad de *brokers* públicos, HiveMQ y Mosquitto por mencionar algunos. Para esta implementación se utiliza el servidor proporcionado por Mosquitto que se utiliza como un entorno de prueba. El servidor utilizado es "test.mosquitto.org" en el puerto 1883 que no necesita seguridad TLS. Al ser un *broker* público no hay necesidad de utilizar autenticación ya que cualquier usuario puede conectarse. Los clientes se modifican para que se puedan conectar al servidor en la nube en el puerto seleccionado.

En la figura 5.10 se observa los elementos que conforman el sistema para un servidor público. La información de entrada para el cliente-publicador son el mensaje en texto que se quiere compartir y la llave que se usa para cifrar dicha información. La información se cifra y se oculta mediante la función estego, esto da como resultado un paquete PUBLISH que contiene de manera oculta información que se quiere compartir. El paquete se envía a un *broker* público, indicando la dirección host y el

puerto utilizado. El *broker* se encarga de direccionar el mensaje publicado al cliente suscrito al tema en que se publica el mensaje, el tema debe ser único ya que cualquier dispositivo conectado a dicho tema recibirá el mensaje. El mensaje es recibido por el cliente-suscriptor, se extrae la información con la ayuda de la función estego inversa, se descifra la información con la llave de cifrado previamente compartida y se obtiene el mensaje original.

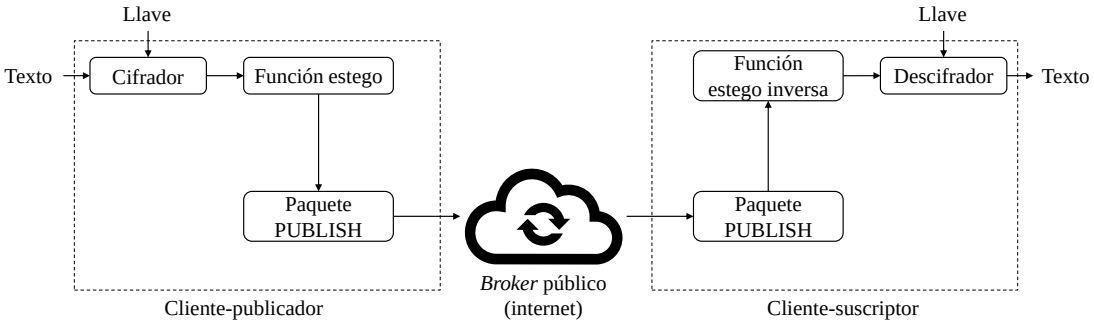


Figura 5.10: Diagrama del sistema usando un *broker* público.

Capítulo 6

Resultados

La implementación del sistema se realiza de dos maneras. El primer método se realiza en software con la ayuda de una Raspberry Pi 4 con una memoria microSD de 32 GB para el almacenamiento del sistema operativo y archivos, y una computadora de escritorio con sistema operativo Linux funcionando como *broker*. El dispositivo Raspberry funciona como cliente-publicador y se encarga de realizar la recepción de la información, el cifrado de la información, realizar la función estego para insertar la información y el envío del paquete PUBLISH. La computadora se encarga de alojar el *broker* que redirecciona la información, y el cliente que recibe dicha información, la descifra y muestra la información original.

Lo primero a analizar son las tres propiedades necesarias para la esteganografía, robustez, indetectabilidad y capacidad. La robustez es la propiedad que dificulta eliminar la información del objeto portador. Hay que revisar cuanta robustez es brindada por el protocolo MQTT, esta puede verse afectada si algún otro cliente-publicador publica en el mismo tema, y se pueden perder el orden de los paquetes y afectar la información que se recibe. Para evitar este tipo de problemas se puede utilizar una funcionalidad que permita del lado del *broker* para especificar que cliente puede publicar en un tema en específico.

El protocolo MQTT permite la reconexión mediante el identificador de cliente (ClientID), por tal motivo si ocurre una desconexión el dispositivo podrá conectarse de nuevo y seguir enviando la información sin problemas. Con estos dos mecanismos se asegura tener una robustez en la intermitencia de los datos o alguna perdida de desconexión.

Para la capacidad se evalúa para cada función estego. En la función bit menos significativo en cada paquete se puede ocultar un bit de información, es decir si la información tiene un tamaño de n bits, se necesitarán n paquetes para que la información sea enviada. La función bit menos significativo *nibble* oculta un bit en cada *nibble* que conforma el campo de 32 bits, se pueden formar ocho *nibbles* en el campo por tal motivo se pueden ocultar ocho bits (un byte) de información en cada paquete. En la ecuación 6.1 se representa como calcular el número de paquetes que se deben enviar para compartir la información.

$$\text{Num. de paquetes} = \frac{\text{Tamaño de la información}(n)}{8} \tag{6.1}$$

La indetectabilidad es la propiedad que no permite detectar diferencia entre el objeto portador y el objeto portador con información insertada, es decir es difícil detectar la información que se ocultó. El mensaje puede ser detectado con facilidad si el mensaje se inserta de forma directa en el campo. Para realizar la prueba de indetectabilidad de usa el software Wireshark. Con la ayuda del software se puede observar la información que contiene el paquete PUBLISH que se ha enviado, la cadena que se quiere ocultar es "Hola a todos, esta es una prueba", el nombre del tema "prueba" y "Hola" como carga útil.

En la figura 6.1 se muestra los paquetes capturados por Wireshark, se filtran los paquetes para solo encontrar paquetes relacionados al paquete MQTT. En el cuadro rojo se puede observar los paquetes que usa el cliente-publicador, ya que estos son los mensajes que se quieren analizar, y se visualiza un paquete CONNECT como su paquete de respuesta CONNACK para posteriormente tener los paquetes PUBLISH que contienen la información. En el cuadro azul se muestra cómo se conforma la información que se recopilo, nos da la información del protocolo, el sistema operativo del sistema, el puerto de salida y, de entrada, la longitud, entre otra información. Y en el cuadro verde se observa la información del paquete que se recopilo, se observa dicha información en formato hexadecimal y ASCII.

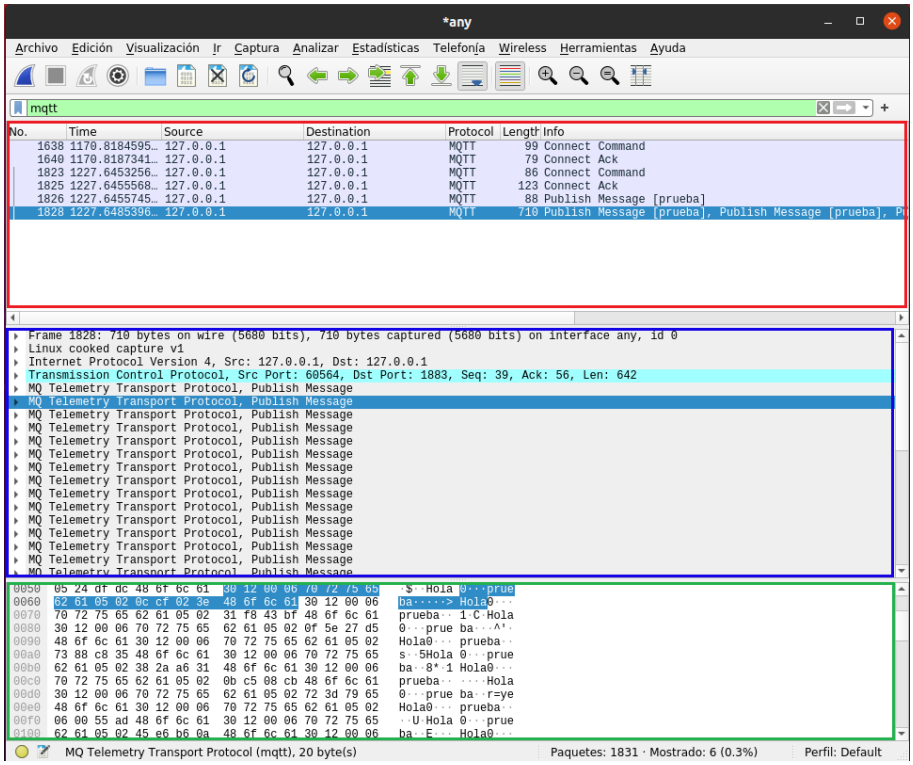


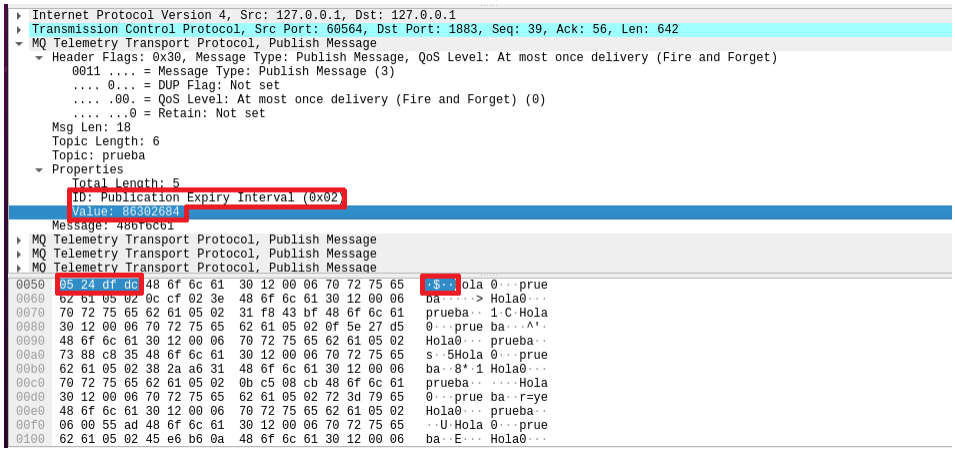
Figura 6.1: Captura de paquetes del esquema ocultación de información

La figura 6.2a muestra el contenido que conforma el paquete PUBLISH. Se analiza el campo intervalo de caducidad de mensaje, en donde se puede observar que tiene un valor numérico de 86302684 y un valor hexadecimal 0524DFDC y caracteres ·$··. Para la figura 6.2b se muestra el paquete consecutivo al anterior cuyo valor numérico es 214893118, 0CCF023E en hexadecimal y ···> como caracteres. Y en la figura 6.2c se muestra otro paquete que sigue con los anteriores, el valor numérico es 838353855, el hexadecimal 31F843BF y los caracteres 1·C·.

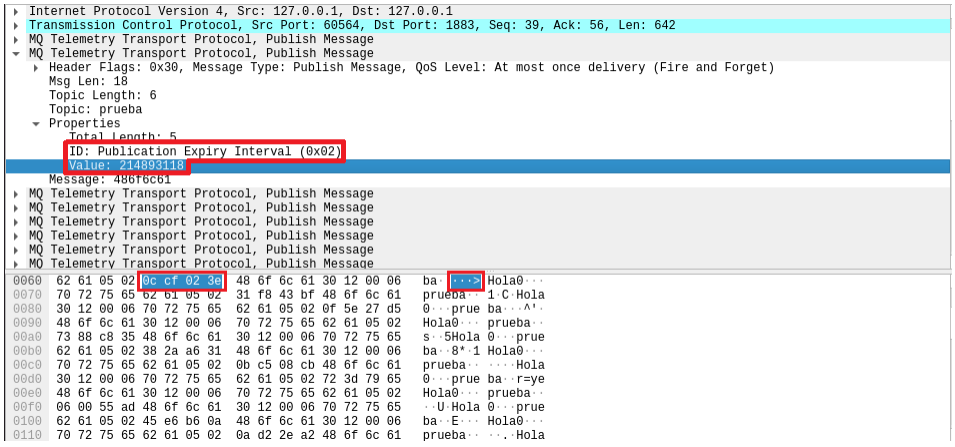
Observando los valores anteriores, se puede apreciar que entre ellos no hay relación alguna en ninguno de los valores, además no se guarda relación alguna con la cadena original. Por tal motivo si varios paquetes son interceptados y analizados es poco probable que se encuentre una relación entre ellos y la información oculta. Con lo anterior se asegura que es difícil detectar la información que se oculta lo que brinda una alta indetectabilidad a la función estego.

A continuación, se realiza un análisis en el tiempo de cifrado y estego junto con el tiempo de ejecución para el esquema de ocultación de la información. El tiempo del cifrado se une con el tiempo de la realización de la función estego dado que es el tiempo del sistema. El tiempo del sistema se debe conocer y observar cómo se desempeña en ambas plataformas. Para ello se realizan 100 muestras y se grafican los tiempos del sistema y de ejecución. Se utiliza un mensaje con tamaño de 128 bits.

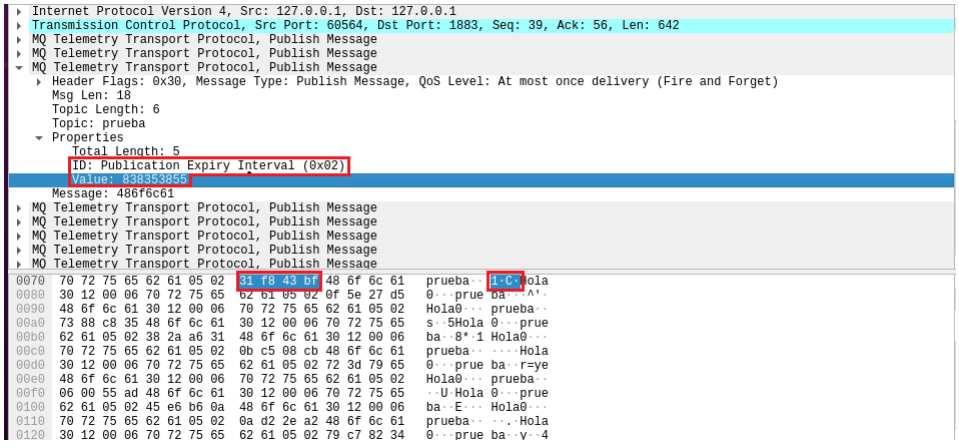
En la figura 6.3a se observa el tiempo que tarda el sistema en realizar el cifrado y la función estego en la tarjeta con el procesador Cortex-A9 y la FPGA y en la figura 6.3b el tiempo realizado por la Raspberry pi 4. Como se puede observar en la ejecución en FPGA el tiempo no es estable y en ocasiones realiza tiempos abruptos, van desde 20 ns hasta 0.07 s. En la Raspberry se observa un tiempo promedio de 0.8 ms. Los tiempos de la Raspberry no son menores que 0.5 ms. En términos de tiempo el sistema desarrollado en la FPGA puede tener tiempos menores y lo hace mejor computacionalmente frente al implementado en Raspberry pi 4.



(a) Primer paquete

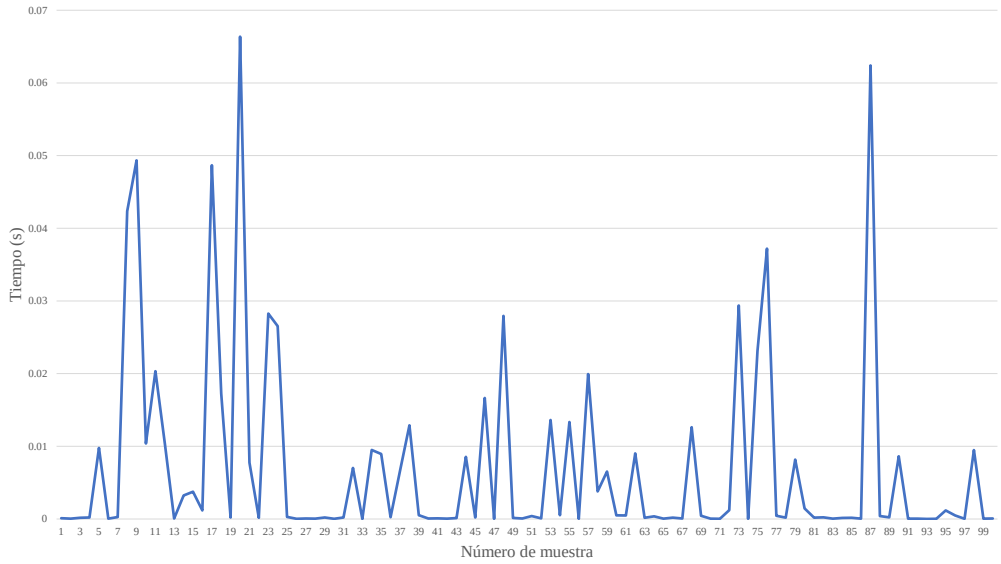


(b) Segundo paquete

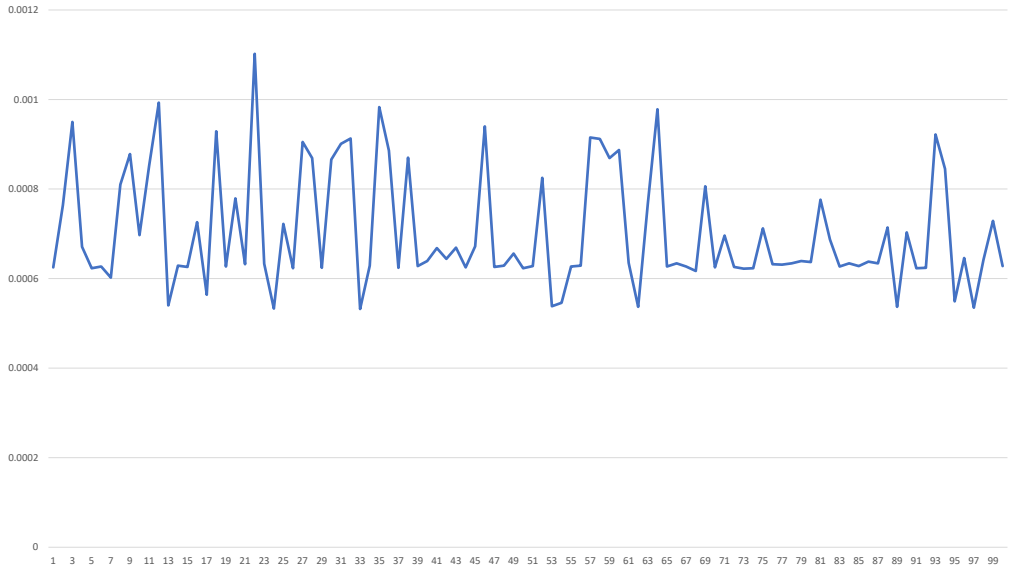


(c) Tercer paquete

Figura 6.2: Información de paquetes PUBLISH



(a) FPGA



(b) Raspberry pi 4

Figura 6.3: Tiempo en realizarse el cifrado y función estego del esquema ocultación de información

En el esquema de ocultación de la llave de cifrado se analizan los paquetes que se capturan en Wireshark. En la figura 6.4 se muestra el contenido de dos paquetes PUBLISH. El contenido marcado en azul muestra el penúltimo paquete que se envió

en donde se puede observar como el campo intervalo del mensaje contiene el número 42C59843 en hexadecimal, que contiene los ocho últimos bits de la información que se ocultó es decir la llave de cifrado, y la carga útil se tiene el mensaje "hola". El siguiente paquete marcado en color rojo marca el fin del envío de información mediante el valor A9, en hexadecimal, en el campo, en este paquete se envía la información cifrada en la carga útil. Los paquetes no tienen relación entre sí y en el paquete final se observa la información cifrada, pero la llave de cifrado no se observa dado que se ocultó por la función estego.

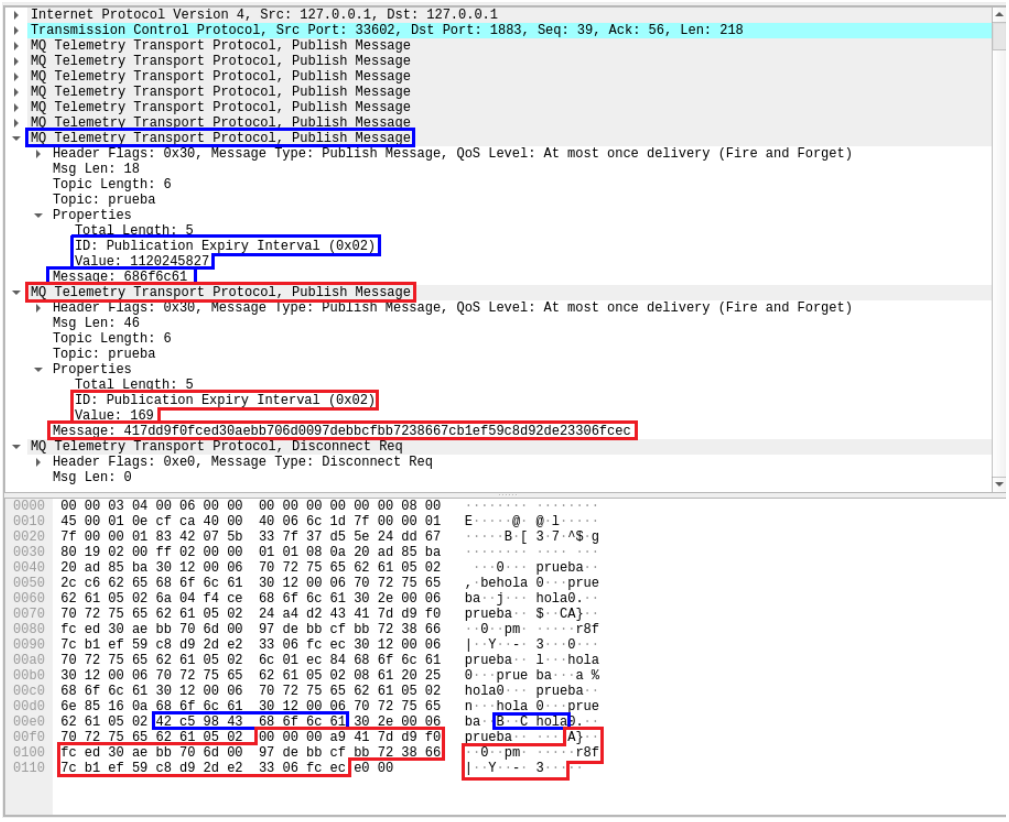
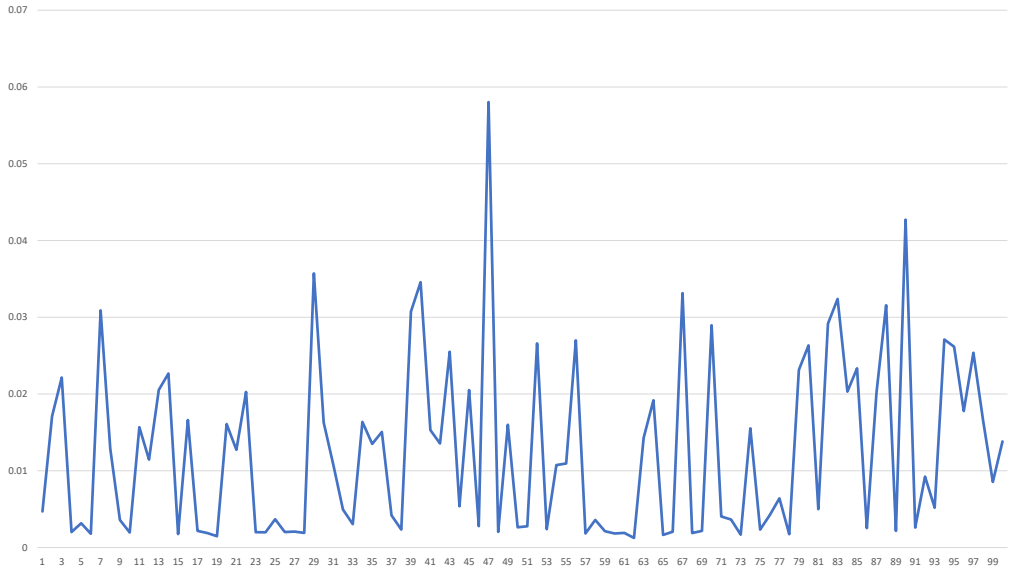
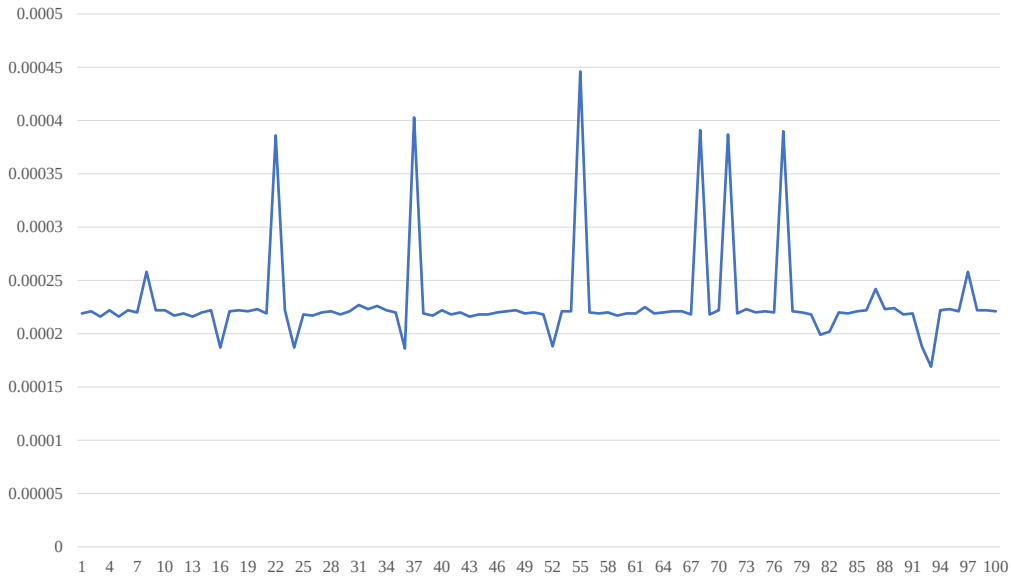


Figura 6.4: Captura de paquetes del esquema ocultación de llave

En la figura 6.5a se observa una gráfica con el tiempo en segundos que tarda el cifrado y la función estego usando la FPGA, y la figura 6.5b se muestra la gráfica de tiempo igual en segundos usando una Raspberry Pi 4. El tiempo que emplea la FPGA es mayor a la Raspberry por tal motivo, en este esquema es preferible usar el software en lugar de hardware para ocultar la llave de cifrado. La forma de ocultar la llave es correcta, la información cifrada se puede conocer dado que la carga útil es visible, pero la llave se puede compartir y es difícil poder encontrar la relación que hay entre los paquetes involucrados.



(a) FPGA

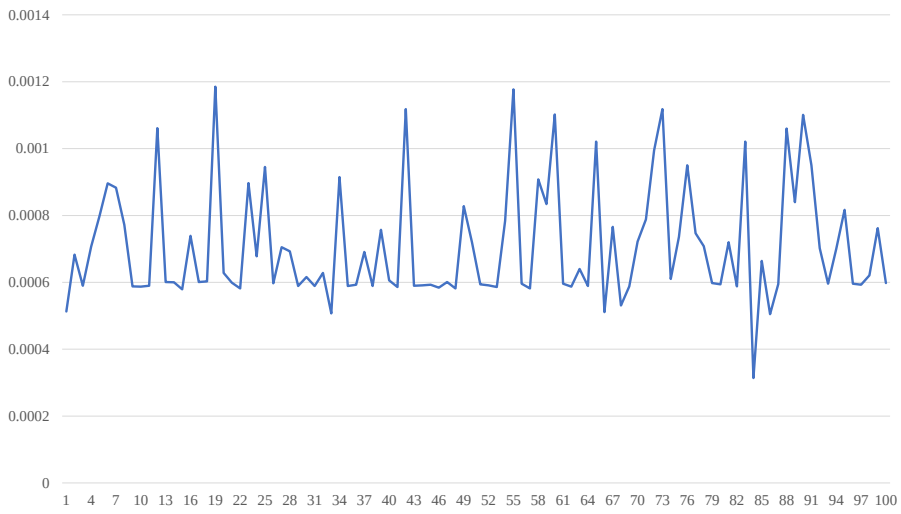


(b) Raspberry pi 4

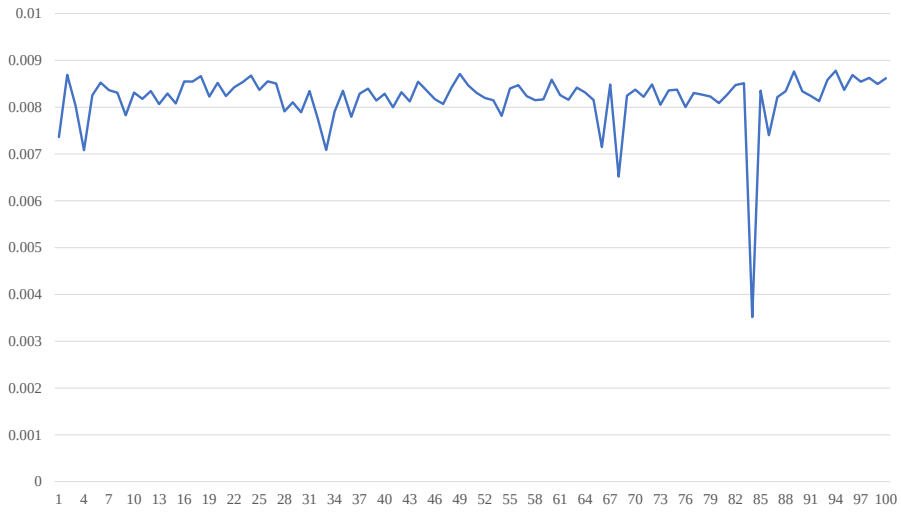
Figura 6.5: Tiempo en realizarse el cifrado y función estego del esquema ocultación de llave de cifrado

El sistema utiliza, además del cifrador SIMON, el cifrador SPECK. Este cifrador es diseñado para su implementación en software. Este sistema solo fue implementado en una Raspberry Pi 4. En las figuras 6.6a y 6.6b se muestra el tiempo de ejecución

del cifrado y de la función estego, y el tiempo de ejecución del programa completo. El tiempo del cifrado y de la función estego se encuentra en un rango entre 0.0006 y 0.001 s, el tiempo es muy intermitente y no presenta un tiempo constante, esto puede deberse a la forma es que se cifra la información y el manejo de bits en software para la función estego. El tiempo de ejecución del sistema presenta un tiempo más constante entre 0.008 y 0.009, presenta tiempos menores en algunas pruebas, pero en general el tiempo es consistente.



(a) Cifrado y función estego



(b) Sistema completo

Figura 6.6: Tiempo de ejecución del sistema con el cifrador Speck

La implementación usando un *broker* público se logró con éxito. Ambos clientes, publicador como suscriptor, se conectaron y el intercambio de información se logró satisfactoriamente. El cliente-publicador realiza las operaciones de cifrado y la ocultación de la información, los paquetes generados son enviados al *broker* público y este a su vez envía los paquetes al cliente-suscriptor. La información oculta se recupera de forma correcta cuando se reciben los paquetes del lado del suscriptor, se extrae la información oculta y se descifra.

Capítulo 7

Conclusiones y trabajo a futuro

Este trabajo presenta un sistema que oculta y brinda confidencialidad a la información usando el protocolo de comunicación MQTT en dispositivos de IdC. El protocolo MQTT v5.0 mostró ser un buen objeto portador. Este protocolo es poco complejo y liviano lo que lo hace ideal para trabajar con dispositivos con recursos limitados. La versión 5.0 contiene más campos que sirven para ocultar información a diferencia de la versión 3.1.1, lo que lo vuelve un excelente candidato para usar como objeto portador. El paquete PUBLISH es el más idóneo para utilizar debido a que es usado para intercambiar información entre los clientes y el *broker* y no es extraño que una gran cantidad de estos paquetes sean intercambiados a comparación de otros paquetes como CONNECT o SUBSCRIBE que solo se envía un par de veces. Los paquetes se conforman de un encabezado fijo, un encabezado variable y la carga útil, el encabezado fijo no puede ser modificado o el paquete no funcionará de manera correcta, la carga útil contiene la información que se quiere compartir y no es preferible modificarla porque es fácil de visualizar, el encabezado variable contiene varios campos que se pueden utilizar para ocultar información unos más viables que otros que dependen de la función estego que se desarrolle para la ocultación.

La información se cifra mediante dos algoritmos criptográficos ligeros, SIMON y SPECK. El primero se enfoca en hardware y el segundo en software. SIMON se implementó en hardware y software para realizar un análisis sobre el tiempo de ejecución. En las figuras 6.3a y 6.3b se muestran los tiempos del cifrado y la función estego de la ejecución en FPGA y Raspberry respectivamente, en la FPGA los tiempos son algo variables pero logra tener tiempos menores en comparación con el tiempo en la Raspberry. El cifrado que se realiza no afecta la función estego por tal motivo si se logra proveer confidencialidad con un algoritmo que se adecua a dispositivos de recursos limitados.

Se desarrollaron cinco funciones esteganográficas (las primeras tres funciones se detallan en el Apéndice C) que están enfocadas para cuatro campos, tema de respuesta, tipo de contenido, nombre del tema e intervalo de caducidad del mensaje. Los tres primeros campos utilizan el tipo de dato UTF-8 y el último utiliza un entero sin signo de 32 bits. La primera función estego llama uno a uno se utilizan dos campos, tema de respuesta y tipo de contenido. Se inserta la información de tal manera que los

caracteres en las posiciones pares se insertan en las posiciones pares del campo tema de respuesta y los caracteres en las posiciones impares se insertan en las posiciones impares del campo tipo de contenido y las posiciones vacías se llenan con caracteres obtenidos de forma aleatoria. La segunda función estego llamada intercalado parte la cadena de caracteres original en dos y la primera mitad los inserta en las posiciones pares del campo tema de respuesta y la segunda mitad en las posiciones impares. Ambas funciones trabajan con caracteres y al no transformar la información se logra apreciar una relación entre el mensaje original y la cadena obtenida al aplicar la función estego, otro problema que se encuentra es que solo se pueden usar caracteres alfanuméricos y si se aplica un cifrado no es seguro que la información solo contenga estos caracteres lo cual hace inviable estas dos funciones.

La tercera función estego llamada fin de cadena, utiliza el principio de delimitar una cadena. Cuando se quiere expresar el fin de una cadena se utiliza el caracter '\0' y así el sistema detecta el tamaño de una cadena. Para esta función se agrega este carácter y seguido se añade la información que se quiere ocultar, así usando la forma en que el bróker detecta el nombre del tema se averigua si es factible el uso de esta función. Al realizar los experimentos y al usar el software Wireshark se observó una alerta. Dicha alerta mostraba que se detectaba un error en la creación del paquete, esto no es deseado ya que indica a alguien que analiza el paquete que hay una modificación. Esta alerta se dejó pasar y se observó el comportamiento del *broker*, la información no llegaba al cliente suscrito al tema por lo que esta función es inviable.

Las últimas dos funciones se describen en este trabajo de tesis y se pudo observar que logran ocultar información. Al analizarse con Wireshark no se encontró relación alguna entre los paquetes que se envían y con la información original que se ocultó. Logrando así dos funciones que permiten ocultar información que cumplen con la robustez, capacidad e indetectabilidad propiedades necesarias en una función estego.

La implementación en FPGA requería un sistema operativo para el procesador Cortex-A9, con la que la placa trabaja. La primera opción FreeRTOS posee una biblioteca para el uso del protocolo MQTT, sin embargo, solo funciona para la versión 3.1.1 y al trabajar con la versión 5.0 no es compatible con el sistema. Se trato de añadir la versión nueva a FreeRTOS para así utilizar el sistema operativo, no obstante, no se logró el objetivo y por falta de tiempo se tuvo que dejar el desarrollo de esta implementación.

El siguiente sistema operativo es Petalinux, un sistema basado en Yocto que nos permite recrear un entorno de Linux enfocado en sistemas embebidos de Xilinx. Los paquetes necesarios están disponibles desde la creación del proyecto petalinux, la versión disponible 1.6.12 cuenta con soporte para la versión 5.0 que permite el desarrollo del sistema planteado en el presente documento. Por lo que es posible el desarrollo del sistema usando una FPGA para el proceso de cifrado y de la función estego.

Se desarrollo el sistema en una Raspberry Pi 4 para así comparar el comportamiento del sistema en ambos dispositivos. En el primer esquema la FPGA en el cifrado y la función estego en ocasiones el tiempo es mucho menor comparado con el tiempo

en la Raspberry y en otras el tiempo es diez veces mayor, esto puede ser producido por el tiempo que se tarda en enviar la información o por la misma naturaleza de la FPGA. Para el segundo esquema se puede apreciar que el desarrollo en software es más eficiente en términos de tiempo.

Dado que se logró implementar ambos esquemas en los dispositivos seleccionados, se puede concluir que este sistema es capaz de ser utilizado en dispositivos con recursos limitados, ya que no se necesita un sistema tan complejo para realizar las funciones esteganográficas desarrolladas y los cifradores SIMON y SPECK están diseñados para trabajar en un entorno con recursos limitados.

Un *broker* público permite una conexión a cualquier usuario. No es necesario usar credenciales de autenticación. Se usa el *broker* de Mosquitto con dirección host "test.mosquitto.org" y en el puerto 1883. En este puerto no hay seguridad, y no es necesaria debido a que ya se otorga seguridad con el sistema implementado. Aunque los paquetes sean interceptados si no se logra saber que dentro del campo intervalo de caducidad se ocultó información, y además se debe conocer la función estego, el algoritmo de cifrado y la llave de cifrado para conocer la información que se compartió.

En las pruebas usando un *broker* público se logró enviar y recibir de manera correcta. Se puede concluir que no se necesita un *broker* en específico, ya que las modificaciones se realizan en los clientes. El *bróker* queda sin modificar y solo se necesitan los clientes para lograr añadir seguridad a la información. No se modifica el *bróker* y esto permite que cualquier *bróker* pueda ser utilizado para este sistema.

Hay tres trabajos relacionados con el trabajo realizado. El primer trabajo realizado por Kosiak, et al. [73] plantea utilizar un sistema de detección de intrusos para detectar ciertos tipos de esteganografía mediante el software Zeek. En este trabajo se analiza el protocolo MQTT v3.1.1 y lograron detectar la mayoría de los casos. En comparación con el trabajo presentado, se utiliza la versión 5.0 que es la más actual en el momento de realizar el trabajo, además no presenta una función estego que inserte la información en el paquete del protocolo.

En el trabajo realizado por Velinov, et al. [46] utiliza el protocolo MQTT v3.1.1 para encontrar canales encubiertos en dicho protocolo. Desarrollaron dos tipos de canales, siete directos y seis indirectos. Los primeros solo necesitan que el emisor y el receptor permanezcan activos para la transmisión de información y los segundos no necesariamente necesitan que ambos dispositivos estén activos, pero necesitan un intermediario para la comunicación. Los canales directos utilizan un campo del protocolo para ocultar información, sin embargo, no utilizan una función estego para insertar la información, en su lugar insertan directamente la información. Las pruebas que realizaron solo fueron enfocadas en los canales indirectos que se basan en indicar un '1' binario si un campo en específico es utilizado y un '0' si el campo no es utilizado.

Mileva, et al. [74] sigue el trabajo realizado por Velinov, et al. [46], en este trabajo se analiza el protocolo MQTT v5.0 para encontrar nuevos canales directos e indirectos, dado que la nueva versión añade campos nuevos. Encontraron nuevos canales además de los encontrados en el trabajo anterior, sin embargo, su implementación sigue enfocada en canales indirectos que presentan un valor '1' binario si un campo

está presente y un '0' si no lo está.

Estos trabajos se asemejan a el trabajo realizado, no obstante, hay técnicas que no se aplican en los trabajos anteriores mencionados. Estos son utilizar una función estego para la inserción de la información al objeto portador además de agregar un cifrador y utilizar la versión más actual de este protocolo.

Como trabajo a futuro se propone sustituir el sistema operativo Petalinux, por uno más liviano como FreeRTOS ya que en este momento no hay biblioteca para el protocolo MQTT v5.0. Implementar más funciones esteganográficas para tener más variedad en la forma de ocultar la información. Este trabajo se enfoca en el protocolo MQTT, pero se puede ampliar más el panorama al utilizar más protocolos tales como CoAP o XMPP como medios para ocultar la información. Desarrollar a futuro la implementación del cifrador y la función estego como un circuito integrado de un único propósito.

Bibliografía

[1] Maria Guadalupe Parra. Sistema computacional para la detección de información oculta en archivos de audio digital utilizando la transformada rápida de fourier: Detesteg audio 1.0, Aug 2016.

[2] OASIS Standard. MQTT version 5.0. *Retrieved June*, 22:2020, 2019.

[3] Syaiful Andy, Budi Rahardjo, and Bagus Hanindhito. Attack scenarios and security analysis of mqtt communication protocol in iot system. In *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 1–6. IEEE, 2017.

[4] Abdullah AlWatyan, Wesam Mater, Omar Almutairi, Mohammed Almutairi, Aisha Al-Noori, et al. Security approach for LSB steganography based FPGA implementation. In *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, pages 1–5. IEEE, 2017.

[5] National Institute of Standars and Technology. Lightweight cryptography. https://csrc.nist.gov/Projects/Lightweight-Cryptography, 2017. [Accedido el 22 de Noviembre de 2021].

[6] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *cryptology eprint archive*, 2013.

[7] Manju Khari, Aditya Kumar Garg, Amir H Gandomi, Rashmi Gupta, Rizwan Patan, and Balamurugan Balusamy. Securing data in internet of things (IoT) using cryptography and steganography techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(1):73–80, 2019.

[8] Anshuman Kalla, Pawani Prombage, and Madhusanka Liyanage. Introduction to iot. *IoT Security: Advances in Authentication*, pages 1–25, 2020.

[9] Omkar Badve, BB Gupta, and Shashank Gupta. Reviewing the security features in contemporary security policies and models for multiple platforms. In *Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security*, pages 479–504. IGI Global, 2016.

[10] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[11] Brij B Gupta. *Computer and cyber security: principles, algorithm, applications, and perspectives*. CRC Press, 2018.

[12] Omnia Abu Waraga, Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. Design and implementation of automated iot security testbed. *Computers & Security*, 88:101648, 2020.

[13] James A Jerkins. Motivating a market or regulatory solution to iot insecurity with the mirai botnet code. In *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)*, pages 1–5. IEEE, 2017.

[14] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[15] Zhen Ling, Kaizheng Liu, Yiling Xu, Yier Jin, and Xinwen Fu. An end-to-end view of iot security and privacy. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[16] Orlando Arias, Jacob Wurm, Khoa Hoang, and Yier Jin. Privacy and security in internet of things and wearable devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):99–109, 2015.

[17] Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. Smart nest thermostat: A smart spy in your home. *Black Hat USA*, (2015), 2014.

[18] Bryan Parno, Jonathan M McCune, and Adrian Perrig. *Bootstrapping trust in modern computers*. Springer Science & Business Media, 2011.

[19] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A {Large-Scale} analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, 2014.

[20] Mathieu Stephan. Hacking the d-link dsp-w215 smart plug, May 2014.

[21] Critical security flaw: Glibc stack-based buffer overflow in getaddrinfo() (cve-2015-7547), Feb 2016.

[22] Ms Smith. Security holes in the 3 most popular smart home hubs and honeywell tuxedo touch. *Network World*, 2015.

[23] Ishtiaq Rouf, Hossen Mustafa, Miao Xu, Wenyuan Xu, Rob Miller, and Marco Gruteser. Neighborhood watch: Security and privacy analysis of automatic meter reading systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 462–473, 2012.

[24] Chaoshun Zuo, Wubing Wang, Zhiqiang Lin, and Rui Wang. Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services. In *NDSS*, 2016.

[25] Xuanxia Yao, Xiaoguang Han, Xiaojiang Du, and Xianwei Zhou. A lightweight multicast authentication mechanism for small scale iot applications. *IEEE Sensors Journal*, 13(10):3693–3701, 2013.

[26] Xiaojiang Du and Hsiao-Hwa Chen. Security in wireless sensor networks. *IEEE Wireless Communications*, 15(4):60–66, 2008.

[27] Jesus Molina. Learn how to control every room at a luxury hotel remotely: The dangers of insecure home automation deployment. *Black Hat USA*, 2014, 2014.

[28] Mahmudur Rahman, Bogdan Carbunar, and Madhusudan Banik. Fit and vulnerable: Attacks and defenses for a health monitoring device. *arXiv preprint arXiv:1304.5672*, 2013.

[29] Mengmei Ye, Nan Jiang, Hao Yang, and Qiben Yan. Security analysis of internet-of-things: A case study of august smart lock. In *2017 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 499–504. IEEE, 2017.

[30] Kelvin Ly and Yier Jin. Security studies on wearable fitness trackers. In *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE*, 2016.

[31] Tiffany Hyun-Jin Kim, Lujo Bauer, James Newsome, Adrian Perrig, and Jesse Walker. Challenges in access right assignment for secure home networks. In *5th USENIX Workshop on Hot Topics in Security (HotSec 10)*, 2010.

[32] Sergei Chistiakov et al. Secure storage and transfer of data in a smart lock system. 2017.

[33] Yossef Oren and Angelos D Keromytis. From the aether to the {Ethernet—Attacking} the internet using broadcast digital television. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 353–368, 2014.

[34] Tamara Denning and Tadayoshi Kohno. Empowering consumer electronic security and privacy choices: Navigating the modern home. In *Symposium on Usable Privacy and Security (SOUPS)*. Citeseer, 2013.

[35] Tamara Denning, Tadayoshi Kohno, and Henry M Levy. Computer security and the modern home. *Communications of the ACM*, 56(1):94–103, 2013.

[36] Blase Ur, Jaeyeon Jung, and Stuart Schechter. Intruders versus intrusiveness: teens' and parents' perspectives on home-entryway surveillance. In *Proceedings*

of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pages 129–139, 2014.

[37] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 636–654. IEEE, 2016.

[38] Nathaniel Gyory and M Chuah. Iotone: Integrated platform for heterogeneous iot devices. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 783–787. IEEE, 2017.

[39] Eduardo Fernandez, Juan Pelaez, and Maria Larrondo-Petrie. Attack patterns: A new forensic and design tool. In *IFIP International Conference on Digital Forensics*, pages 345–357. Springer, 2007.

[40] Thamer A Alghamdi, Aboubaker Lasebae, and Mahdi Aiash. Security analysis of the constrained application protocol in the internet of things. In *Second international conference on future generation communication technologies (FGCT 2013)*, pages 163–168. IEEE, 2013.

[41] Britt Cyr, Webb Horn, Daniela Miao, and Michael Specter. Security analysis of wearable fitness devices (fitbit). *Massachusetts Institute of Technology*, 1, 2014.

[42] Cheena Sharma and Naveen Kumar Gondhi. Communication protocol stack for constrained iot systems. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. IEEE, 2018.

[43] Eclipse Foundation. IoT Developer Survey Results, Apr 2018.

[44] Eclipse Foundation. 2020 IoT Developer Survey Key Findings, Oct 2020.

[45] OASIS Standard. Mqtt version 3.1. 1. *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, 1:29, 2014.

[46] Aleksandar Velinov, Aleksandra Mileva, Steffen Wendzel, and Wojciech Mazurczyk. Covert channels in the MQTT-Based internet of things. *IEEE Access*, 7:161899–161915, 2019.

[47] Biswajeeban Mishra and Attila Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020.

[48] C Vanmathi and S Prabu. A survey of state of the art techniques of steganography. *International Journal of Engineering and Technology*, 5(1):376–379, 2013.

[49] Soumyendu Das, Subhendu Das, Bijoy Bandyopadhyay, and Sugata Sanyal. Steganography and steganalysis: different approaches. *arXiv preprint arXiv:1111.3758*, 2011.

[50] Sanghamitra Debnath, Manashee Kalita, and Swanirbhar Majumder. A review on hardware implementation of steganography. In *2017 Devices for Integrated Circuit (DevIC)*, pages 149–152. IEEE, 2017.

[51] Ken Kabeen and Peter Gent. Image compression and discrete cosine transform. *College of Redwoods*.

[52] Mark Noto. MP3Stego: Hiding text in MP3 files. *Sans Institute*, page 5, 2001.

[53] Udit Budhia and Deepa Kundur. Digital video steganalysis exploiting collusion sensitivity. In *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III*, volume 5403, pages 210–221. International Society for Optics and Photonics, 2004.

[54] Sandip Bobade and Rajeshawari Goudar. Secure data communication using protocol steganography in IPv6. In *2015 International Conference on Computing Communication Control and Automation*, pages 275–279, 2015.

[55] Sattar B Sadkhan and Akbal O Salman. A survey on lightweight-cryptography status and future challenges. In *2018 International Conference on Advance of Sustainable Engineering and its Application (ICASEA)*, pages 105–108. IEEE, 2018.

[56] Hamidreza Damghani, Heliasadat Hosseinian, and Leila Damghani. Cryptography review in IoT. In *2019 4th Conference on Technology In Electrical and Computer Engineering (ETECH2019)*, 2019.

[57] Deena Nath Gupta and Rajendra Kumar. Lightweight cryptography: an IoT perspective. *Trivium*, 80(1):2580, 2019.

[58] Isha Bhardwaj, Ajay Kumar, and Manu Bansal. A review on lightweight cryptography algorithms for data security and authentication in IoTs. In *2017 4th International Conference on Signal Processing, Computing and Control (ISPCC)*, pages 504–509. IEEE, 2017.

[59] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Lsb: A lightweight scalable blockchain for iot security and anonymity. *Journal of Parallel and Distributed Computing*, 134:180–197, 2019.

[60] Iqbal H Sarker, Asif Irshad Khan, Yoosef B Abushark, and Fawaz Alsolami. Internet of things (IoT) security intelligence: a comprehensive overview, machine learning solutions and research directions. *Mobile Networks and Applications*, pages 1–17, 2022.

[61] Thirumalesu Kudithi and R Sakthivel. High-performance ECC processor architecture design for IoT security applications. *The Journal of Supercomputing*, 75(1):447–474, 2019.

[62] Ammar Mohammad, Hasan Al-Refai, and Ali Ahmad Alawneh. User authentication and authorization framework in iot protocols. *Computers*, 11(10), 2022.

[63] Vikas S Kait and Bina Chauhan. BPCS steganography for data security using FPGA implementation. In *2015 International Conference on Communications and Signal Processing (ICCSP)*, pages 1887–1891. IEEE, 2015.

[64] Alejandro Martinez, Alberto Ramos, Isaac Compean, and Raquel Avila. Message concealment system of voice signals implemented on FPGA. *IEEE Latin America Transactions*, 14(8):3554–3559, 2016.

[65] Krzysztof Szczypiorski. Steganography in TCP/IP networks. In *State of the Art and a Proposal of a New System–HICCUPS, Institute of Telecommunications' seminar, Warsaw University of Technology, Poland*. Citeseer, 2003.

[66] Deepa Kundur and Kamran Ahsan. Practical internet steganography: data hiding in IP. *Proc. Texas wksp. security of information systems*, 2003.

[67] Princess Marie B Melo, Ariel M Sison, and Ruji P Medina. Enhanced TCP sequence number steganography using dynamic identifier. In *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 482–485. IEEE, 2019.

[68] Pengfei Xue, Jingsong Hu, Ronggui Hu, and Yourui Wang. The solution of key transmission in multi-level network steganography. In *2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC)*, pages 1391–1394, 2017.

[69] Artur M. Brodzki and Jędrzej Bieniasz. Yet another network steganography technique basedon TCP retransmissions. In *2019 5th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 35–39, 2019.

[70] Geethanjali G, C Ashwin, Bharath V P, Avinash A, and Anurag Hiremath. Enhanced data encryption in IoT using ECC cryptography and LSB steganography. In *2021 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, pages 173–177, 2021.

[71] DA Trujillo-Toledo, OR López-Bonilla, EE García-Guerrero, E Tlelo-Cuautle, D López-Mancilla, O Guillén-Fernández, and E Inzunza-González. Real-time RGB image encryption for IoT applications using enhanced sequences from chaotic maps. *Chaos, Solitons & Fractals*, 153:111506, 2021.

[72] Akram A. Almohammedi and Vladimir Shepelev. Saturation throughput analysis of steganography in the IEEE 802.11p protocol in the presence of non-ideal transmission channel. *IEEE Access*, 9:14459–14469, 2021.

[73] Tomasz Koziak, Katarzyna Wasielewska, and Artur Janicki. How to make an intrusion detection system aware of steganographic transmission. In *European interdisciplinary cybersecurity conference*, pages 77–82, 2021.

[74] Aleksandra Mileva, Aleksandar Velinov, Laura Hartmann, Steffen Wendzel, and Wojciech Mazurczyk. Comprehensive analysis of MQTT 5.0 susceptibility to network covert channels. *computers & security*, 104:102207, 2021.

[75] Roger A. Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265, 2017.

Apéndice A

Instalación Petalinux

Petalinux es el sistema operativo que se instala en la tarjeta PYNQZ-Z2 para el procesador Cortex-A9. La versión que se utiliza para la implementación de este trabajo es la versión 2021.2 que se puede descargar del siguiente enlace https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2021-2.html.

Para la instalación de Petalinux v2021.2 es necesario instalar los siguientes paquetes para el sistema operativo Ubuntu, que es donde se realizó la instalación de Petalinux, para su correcto funcionamiento. El archivo xlsx que contiene la lista de paquetes que se deben instalar, dependiendo del sistema operativo Linux que se utiliza, se puede descargar del siguiente enlace https://support.xilinx.com/s/article/000032521?language=en_US. La línea de comandos para instalar los paquetes necesarios es la siguiente.

```
$ sudo apt-get install iproute2 gawk python3 python build-essential gcc git make
    net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1
    gnupg wget git-core diffstat chrpath socat xterm autoconf libtool tar unzip
    texinfo zlib1g-dev gcc-multilib automake zlib1g:i386 screen pax gzip cpio
    python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git
    python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3
```

Petalinux necesita que el sistema host /bin/sh sea “bash”. Si el sistema operativo es Ubuntu el sistema host determinado es “dash”, por lo tanto es necesario cambiar el sistema host mediante el siguiente comando.

```
$ sudo dpkg-reconfigure dash
```

Cuando el archivo de instalación se ha descargado se procede a otorgarle permisos de escritura para instalar los archivos necesarios para el uso de Petalinux. El comando necesario es el siguiente.

```
$ chmod 755 ./petalinux-v2021.2-final-installer.run
```

Se puede instalar Petalinux en la carpeta donde se descargo, el manual de instala-

ción recomienda que no se instale Petalinux como usuario root, esto debido a que los archivos se pueden sobrescribir archivos y corromper el sistema. En caso que se quiera especificar la dirección del archivo de registros, la dirección en donde se quiere instalar Petalinux o más opciones. El comando completo queda de la siguiente manera.

```
$ ./petalinux-v2021.2-final-installer.run [--log <LOGFILE>] [-d|--dir
    <INSTALL_DIR>] [options]
```

Para este trabajo se creo una carpeta en la ubicación home y posteriormente se instalo Petalinux en la ubicación que se acaba de crear. Esto se realiza mediante los siguientes comandos.

```
$ mkdir -p /home/<usuario>/petalinux/2021.2
$ $./petalinux-v2021.2-final-installer.run --dir /home/<usuario>/petalinux/2021.2
```

Para habilitar el entorno del trabajo de Petalinux se utiliza el siguiente comando.

```
$ source <directorio-instalacion-Petalinux>/settings.sh
```

Se puede verificar que se habilita correctamente el entorno con el siguiente comando

```
$ echo $PETALINUX
```

Se obtiene la dirección en donde se instaló Petalinux y así se puede verificar que Petalinux se ha instalado de manera correcta y poder empezar a configurar el proyecto de Petalinux.

Apéndice B

Creación de proyecto Petalinux

Para crear el proyecto de Petalinux lo primero que se debe hacer es habilitar el entorno de trabajo para poder utilizar los comandos propios de Petalinux, esto se muestra en el Apéndice A. Una vez que se ha habilitado el entorno de trabajo se procede a desarrollar el proyecto.

Lo primero es crear un proyecto a partir de una configuración de hardware construida previamente mediante un archivo creado en el software Vivado que permite realizar la configuración de hardware para un uso deseado, con los componentes y programación de hardware deseados. Para la creación del proyecto se utiliza el siguiente comando.

```
$ petalinux-create --type project --template <plataforma> --name
    <nombre_del_proyecto>
```

El comando para crear el proyecto es `petalinux-create` y se especifica que será un proyecto mediante el argumento `--type`, con el argumento `--template` se especifica la plataforma en que se desarrolla el proyecto, estas pueden ser `versal`, para los dispositivos Versal ACAP, `zynqMP` para los dispositivos Zynq UltraScale + MPSoC, `zynq` para dispositivos Zynq-7000 y `microblaze` para dispositivos con procesadores MicroBlaze. Esto crea el proyecto en la dirección en la que se ejecutó este comando.

Cuando se crea el proyecto se crea una carpeta con el nombre que se asignó que contendrá todos los archivos que se utilizaran para la creación del sistema Petalinux. Es necesario ingresar a la carpeta con el siguiente comando.

```
$ cd <nombre-del-proyecto>
```

Se debe configurar el proyecto con la configuración de hardware exportada en el archivo XSA que se ha creado previamente, por lo tanto se utiliza el siguiente comando.

```
$ petalinux-config --get-hw-description <directorio-archivo-XSA>
```

Cuando se ejecuta este comando se abre el menú de configuración del sistema, se pueden cambiar cosas como el nombre de la máquina, configuraciones ethernet, entre

otros, para una mayor profundidad se puede consultar la guia de referencia.

Si se desea configurar el U-boot que es el gestor de arranque de la segunda etapa de carga del sistema operativo Linux. Si se desea realizar una configuración en especifico del u-boot se ejecuta el siguiente comando que despliega el menú de configuración.

```
$ petalinux-config -c u-boot
```

Si se quiere configurar el kernel del sistema se utiliza el siguiente comando para abrir el menú de configuración.

```
$ petalinux-config -c kernel
```

Para la creación del proyecto en este trabajo de tesis se modificó la configuración en la opción *Library routines* → *Default contiguous memory area size*, de manera predeterminada el valor es de 16, y se cambio el valor a 6144.

El siguiente paso es configurar los paquetes que se quieren tener en el sistema Petalinux, se debe abrir el menú para seleccionar los paquetes que se quieren añadir. En esta tesis se necesitan instalados los paquetes de mosquitto-dev, que contiene las bibliotecas para crear los clientes de mosquitto, mosquitto-clients, que instala los clientes de prueba para mosquitto, mosquitto, que instala el broker MQTT y por último se necesita el paquete esencial que contiene el compilador gcc para compilar el cliente que se crea.

El paquete esencial se encuentra en el menú de configuración en la ruta *Package* → *misc*, los paquetes de mosquitto no se encuentran habilitados en el menú. Por tanto es necesario verificar si el paquete se encuentra en el sistema Yocto, que el sistema en el que esta basado Petalinux, esto se puede realizar buscado mosquitto en la siguiente página web https://layers.openembedded.org/layerindex/branch/master/recipes/. Mosquitto se encuentra en el sistema y se debe habilitar para que se pueda añadir el paquete al sistema Petalinux. Esto se realiza de la siguiente manera. Primero se deben añadir las siguientes líneas en *<carpeta-proyecto>/project-spec/meta-user/conf/user-rootfsconfig*.

```
$ CONFIG_mosquitto
$ CONFIG_mosquitto-clients
$ CONFIG_mosquitto-dev
```

Con los comandos anteriores se habilitan los paquete en *user-package*, donde se muestra una lista de todos los paquetes que se ha agregado, en este caso se seleccionan los paquetes de mosquitto para que se puedan utilizar en el sistema.

Para finalizar se construirá el proyecto con base en las características y paquetes seleccionados, esto se realiza utilizando el siguiente comando.

```
$ petalinux-build
```

Esto compila y construye el proyecto Petalinux, si hay algún error se muestra en la consola o en un archivo log que describe el error por el cual no se construyó el sistema,

en ocasiones se genera error en la construcción pero basta con volver a ejecutar el comando para generar de forma correcta la construcción. Una vez el proyecto haya sido construido, se procede a generar la imagen que inicia el sistema. La imagen se genera con el siguiente comando.

```
$ petalinux-package --boot --force --fsbl images/linux/zynq_fsbl.elf --fpga
    images/linux/system.bit --u-boot
```

En la carpeta <carpeta-proyecto>/images/linux se crearan los archivos necesarios para la imagen del sistema. Para almacenar estos archivos se necesita una memoria microSD. La memoria debe tener dos particiones, una en formato FAT32 y otra en formato EXT4. La partición FAT32 debe tener un tamaño mínimo de 500 MB. En la carpeta donde se generaron los archivos para la imagen, se seleccionan y copian los archivos BOOT.bin, boot.scr, image.ub en la partición FAT32. Los archivos del sistema operativo se debe colocar en la partición EXT4. Con el archivo rootfs.tar.gz se utiliza el siguiente comando para que se extraigan los archivos en dicha partición. Es necesario que este comando se ejecute como usuario root.

```
$ sudo tar -xzvf rootfs.tar.gz -C /media/<nombre-memoriaSD>
```

Una vez copiado los archivos, se procede a colocar la memoria microSD en la placa y cambiar el arranque a SD. Se enciende la placa y de manera serial se obtiene la información de arranque del sistema, la conexión serial se realizó con la ayuda del software Vitis de Xilinx. Si todo esta correcto aparecerá en la ventana serial lo siguiente.

```
$ root@<nombre-del-pryecto>:
```

Esto nos indica que el sistema ha arrancado de manera correcta y se puede trabajar en él.

Apéndice C

Funciones esteganográficas

En este Apéndice se muestran las funciones esteganográficas que se desarrollaron y se descartaron por no cumplir con los criterios necesarios para ocultar la información de manera correcta.

C.1. Función Uno a uno

Esta función separa la información en caracteres de 8 bits, y utiliza los campos Tema de respuesta y Tipo de contenido. El algoritmo por seguir es colocar los caracteres con índices pares en el campo Tema de respuesta y los caracteres con índices impares en el campo Tipo de contenido. Se rellenan los espacios vacíos con caracteres generados de manera aleatoria. En la figura C.1 se muestra de manera gráfica la inserción de los caracteres en los campos mencionados.

C.2. Función Intercalado

Se separa la información en caracteres de 8 bits y se utiliza el campo Tema de respuesta para ocultar la información.Esta función se aplica de manera que el caracter en la posición cero de la información a ocultar se inserta en la posición cero de campo, posteriormente el carácter en la posición n se coloca en la posición uno del campo, el caracter en la posición uno de la cadena se coloca en la posición dos del campo, el caracter en la posición n – 1 dela cadena se coloca en la posición tres del campo y así sucesivamente. En la figura C.2 se puede apreciar el diagrama que muestra gráficamente la función.

C.3. Función Fin de cadena

Los datos se separan en caracteres de 8 bits y se utiliza el campo Nombre del tema. El funcionamiento es el siguiente, se agrega primero el nombre del tema, se

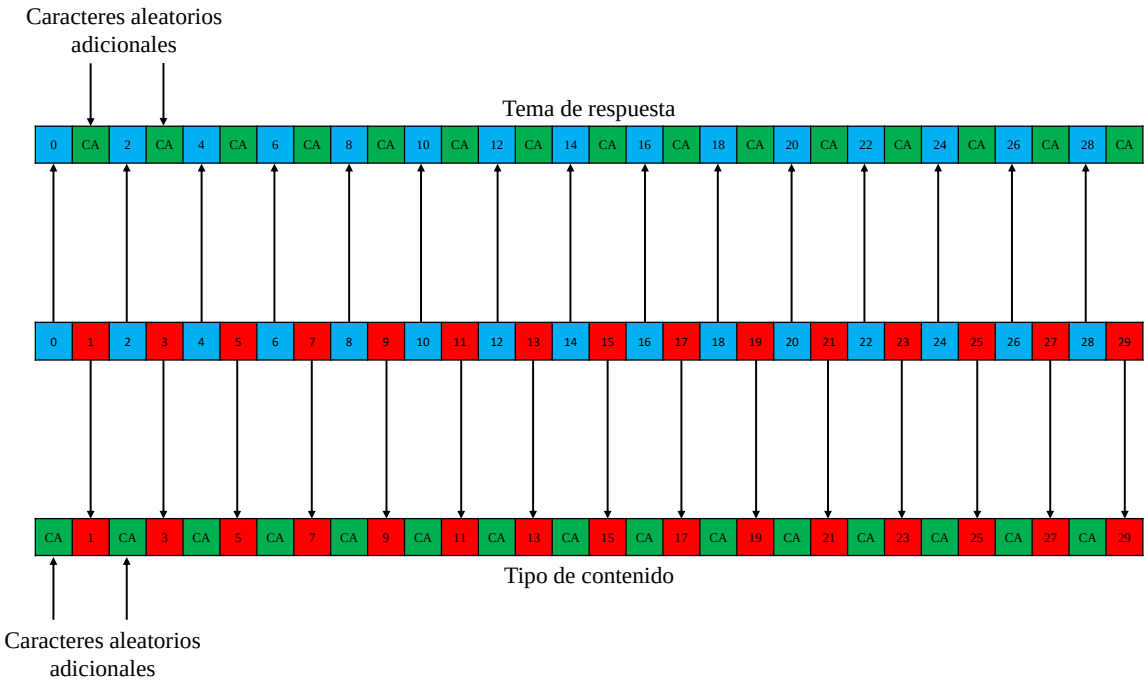


Figura C.1: Diagrama de la función estego Uno a uno

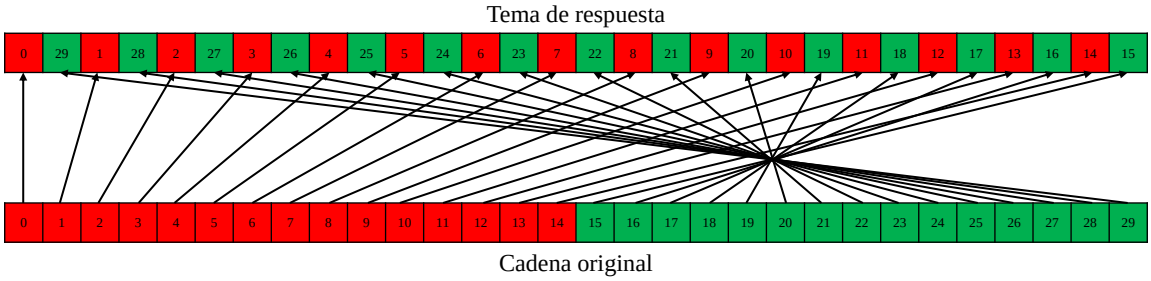


Figura C.2: Diagrama de la función estego Intercalado

agrega el caracter fin de cadena '\0' y se inserta el mensaje a ocultar. En la figura C.3, se muestra de manera gráfica el método que se utiliza para esta función.

Figura C.3: Diagrama de la función estego Fin de cadena