Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Electrónica y Telecomunicaciones
con orientación en Telecomunicaciones

Evaluación de sobrecarga por cifrado en redes multisalto para la internet de las cosas médicas.

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:

Adrián Neftalí Sánchez

Ensenada, Baja California, México

2022

Tesis defendida por

Adrián Neftalí Sánchez

y aprobada por el siguiente Comité

Dr. Salvador Villarreal Reyes
Codirector de tesis

Dr. Gabriel Alejandro Galaviz Mosqueda
Codirector de tesis

Dr. Raúl Rivera Rodríguez

Dr. Vitali Kober

Dr. Miguel Morales Sandoval

Dra. María del Carmen Maya Sánchez
Coordinadora del Posgrado en Electrónica y Telecomunicaciones

Dr. Pedro Negrete Regagnon
Director de Estudios de Posgrado

Resumen de la tesis que presenta Adrián Neftalí Sánchez como requisito parcial para la obtención del grado de Maestro en Ciencias en Electrónica y Telecomunicaciones con orientación en Telecomunicaciones.

Evaluación de sobrecarga por cifrado en redes multisalto para la internet de las cosas médicas.

Resumen aprobado por:

Dr. Salvador Villarreal Reyes
Codirector de tesis

Dr. Gabriel Alejandro Galaviz Mosqueda
Codirector de tesis

El Internet de las cosas médicas (IoMT por sus siglas en ingles) permitirá habilitar servicios y aplicaciones enfocados a diagnósticos más precisos y mejorar la cobertura efectiva de los sistemas de salud. Para esto, es necesario la recolección y envío de información de salud de las personas a través de sensores que monitorean continuamente los signos vitales de las personas. El envío a través de redes para el internet de las cosas (IoT por sus siglas en ingles) de información de salud plantea varios retos importantes para los sistemas de comunicaciones. Uno de los más relevantes es el de preservar la confidencialidad y privacidad de la información desde su recolección hasta su tratamiento y almacenamiento. Esto es especialmente complejo considerando que los sensores utilizados suelen tener pocos recursos en términos de memoria y capacidad de procesamiento. Por lo tanto se han propuesto una serie de algoritmos que cifran esta información de una manera eficiente usando los recursos limitados con los que cuentan estos dispositivos. Estos algoritmos han sido evaluados desde la perspectiva de consumo de potencia y rendimiento. Sin embargo, no se ha hecho una evaluación de cómo éstos afectan el comportamiento de una red IoMT en su conjunto. El presente trabajo de investigación busca evaluar el comportamiento de la implementación de una red IoMT cuya pila de tecnologías en su capa de seguridad utilice algoritmos de cifrado ligero para proteger información desde el momento de conformación de la red. Para realizar esta evaluación se consideraron 5 candidatos de algoritmos de cifrado ligero del concurso que está llevando a cabo el instituto nacional de estándares y tecnología de los Estados Unidos (NIST por sussiglas en ingles), con los cuales utilizando métricas de red, se evaluó el comportamiento de la red al variar entre los distintos algoritmos de cifrado ligero además de compararlos con una implementación del cifrado por defecto AES-128 y una implementación sin ningún tipo de cifrado. Para realizar las diferentes implementaciones con las que se realizó esta evaluación se utilizó el sistema operativo IoT Contiki-ng el cual fue implementado tanto en dos distintos simuladores (Cooja y Renode) como en el dispositivo físico Sensortag-CC2650.

Palabras clave: IoT, IoMT, Contiki-NG, WSN, 6LoWPAN, RPL, IPv6, DTLS, TLS, PSK, TinyDTLS, CoAP, CC2650, CC2538, Grain-128AEAD, Xoodyak, ASCON, GIFT-COFB, TinyJambu, AES-128

Abstract of the thesis presented by Adrián Neftalí Sánchez as a partial requirement to obtain the Master of Science degree in Electronics and Telecommunications with orientation in Telecommunications.

Evaluation of encryption overload in multihop networks for the internet of medical things.

Abstract approved by:

Dr. Salvador Villarreal Reyes
Thesis Co-Director

Dr. Gabriel Alejandro Galaviz Mosqueda
Thesis Co-Director

The Internet of medical things (IoMT) will enable services and applications focused on more accurate diagnoses and improve the effective coverage of health systems. For this, it is necessary to collect and send people's health information through sensors that continuously monitor people's vital signs. Delivering health information over internet of things networks (IoT) poses several significant challenges for communications systems. One of the most relevant is to preserve the confidentiality and privacy of the information from its collection to its treatment and storage. This is especially complex considering that the sensors used tend to have few resources in terms of memory and processing capacity. Therefore, a series of algorithms have been proposed that encrypt this information efficiently using the limited resources available to these devices. These algorithms have been evaluated from the perspective of power consumption and performance. However, no assessment has been made of how these affect the behavior of an IoMT network as a whole. This research work seeks to evaluate the behavior of an implementation of an IoMT network whose stack of technologies in its security layer uses lightweight encryption algorithms to protect information from the moment the network is formed. To carry out this evaluation, 5 candidates for lightweight encryption algorithms from the contest being carried out by National Institute of Standards and Technology (NIST) were considered, with which, using network metrics, the behavior of the network was evaluated for different lightweight encryption algorithms in addition to comparing them. with an implementation of the default encryption AES-128 and an implementation without any encryption. To carry out the different implementations with which this evaluation was carried out, the IoT Contiki-ng operating system was used, which was implemented both in two different simulators (Cooja and Renode) and in the physical device Sensortag-CC2650.

Keywords: IoT, IoMT, Contiki-NG, WSN, 6LoWPAN, RPL, IPv6, DTLS, TLS, PSK, TinyDTLS, CoAP, CC2650, CC2538, Grain-128AEAD, Xoodyak, ASCON, GIFT-COFB, TinyJambu, AES-128

Dedicatoria

A mi querido hermano que descanse en paz.

Agradecimientos

A mi madre Luisa Balbina Sanchez porque sin su apoyo incondicional y su gran fortaleza yo no seria la persona que soy.

A Marisela por ser mi compañera en este gran viaje, por todo su apoyo y ayuda incondicional. Sin ti la maestría hubiese sido menos divertida y mas difícil.

Al Dr. Salvador Villareal por ser un gran mentor, enseñarme y apoyarme en los momentos mas difíciles.

Al Dr. Alejandro Galaviz Mosqueda por todo su tiempo y conocimientos otorgados tanto en lo académico como en la vida.

Al Dr. Aldo Perez Ramos por ser el primero que me mostró este camino, por su apoyo, por su tiempo y por sus conocimientos.

A Enrique, Shiro, Christian y Edwin por su amistad, sus retroalimentaciones y sus enseñanzas tanto en este trabajo como en lo personal.

A los miembros de mi comité de tesis: Dr. Raúl Rivera Rodríguez, Dr. Vitali Kober, Dr. Miguel Morales Sandoval por su valioso tiempo entregado para este trabajo, por sus acertados comentarios, y por sus valiosas aportaciones que ayudaron a enriquecer el trabajo de investigación.

Al Dr. David Covarrubias y al Dr. Jaime Sánchez por todo los conocimientos brindados sin los cuales este trabajo no hubiese sido lo mismo. También al Mtro. Moisés Castro por sus aportaciones a mi gramática.

A Rodolfo, Eduardo, German, Antonio, Fernando, y Rene por ser personas extraordinarias y hacer mas divertido el año de materias.

Al Centro de Investigación Científica y de Educación Superior de Ensenada por proporcionarme maestros extraordinarios y permitirme usar sus instalaciones para descubrir nuevos problemas y proponer algunas soluciones.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría. No. de becario: 989674

Tabla de contenido

Página

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de figuras (continuación)

Figura Página

Lista de tablas

Tabla Página

Capítulo 1. Introducción

1.1. Antecedentes

El avance de las nuevas tecnologías de comunicación inalámbrica y procesadores de bajo consumo de potencia en los últimos años ha derivado en una nueva rama de investigación llamada Internet de las cosas. El Internet de las cosas (IoT por sus siglas en ingles) es el concepto comúnmente utilizado para referirse al conjunto de tecnologías que hacen posible que dispositivos simples se puedan conectar al Internet e intercambiar información entre ellos y hacia la nube sin intervención de un usuario (Bansal y Kumar, 2020). Un ejemplo de esto es un refrigerador que se conecta al Internet para pedir un carrito de la compra a domicilio cuando detecta que la despensa escasea.

El paradigma introducido por la IoT también se puede aplicar en el área de la medicina, utilizando dispositivos capaces de monitorizar diferentes variables fisiológicas del cuerpo humano. A esto se conoce como “Internet de las Cosas Médicas” (IoMT) (Bansal y Kumar, 2020) (Sethi y Sarangi, 2017) (Vishnu *et al.*, 2020). Un ejemplo de este tipo de dispositivos puede ser un medidor de signos vitales, el cual envía alertas en tiempo real ante algún acontecimiento que pueda poner en riesgo el bienestar del paciente. Debido a la relevancia de la información transmitida y el inherente derecho a la privacidad de los datos del paciente, un fallo en la seguridad de estos dispositivos podría poner en riesgo la vida del usuario o proporcionar información sensible a terceros con fines maliciosos. Con la tendencia a tener dispositivos más pequeños, portátiles, con batería y capacidades de procesamiento restringidas, en la actualidad la seguridad en IoT y en especial en IoMT sigue siendo un área de investigación abierta. Por lo que para desplegar las aplicaciones previstas, es de vital importancia realizar aportaciones en este campo (Alsubaei *et al.*, 2017).

Dentro del contexto de la IoMT, el monitoreo remoto de pacientes se está convirtiendo en una opción viable gracias a la utilización de redes inalámbricas de sensores médicos (WMSN). Éstas redes pueden informar varias condiciones médicas para ser rastreadas cuando el paciente se encuentra fuera del hospital (Hayajneh *et al.*, 2016). Para este tipo de aplicaciones, es de notar la alta importancia que tiene la protección de los datos transmitidos, puesto que en caso de ser comprometidos podría haber

consecuencias catastróficas e incluso mortales. Debido a esto es muy importante que los datos sean cifrados y permanezcan así en todo su proceso de comunicación, descifrandolos únicamente cuando el usuario final los requiera.

Debido a las restricciones en recursos de cómputo, batería y tasa de datos que típicamente poseen los dispositivos IoMT(Mohd *et al.*, 2015), los cifradores que se utilizan actualmente en otras áreas de la computación no pueden ser desplegados directamente en estos dispositivos de manera efectiva. Por esta razón, comenzaron a surgir una nueva gama de cifradores, enfocados en poder trabajar en estos dispositivos adecuadamente, los cuales se denominaron cifradores ligeros (Mohd *et al.*, 2015).

En los últimos años se han diseñado o adaptado nuevas técnicas de cifrado ligero orientadas a establecer enlaces seguros entre dispositivos IoMT. Actualmente, estos cifradores están bajo evaluación por parte del “National Institute of Standards and Technology” (NIST) como parte de un concurso para determinar los algoritmos que se van a utilizar en el próximo estándar de cifrado ligero (Turan y Bassham, 2019). En este concurso, 57 algoritmos de cifrado ligero se han sometido a diversas evaluaciones para determinar cuál es el más apto. A la fecha de la escritura del presente trabajo de tesis, el concurso se encuentra en su fase final en donde de los 57 algoritmos solo quedan 10. En estos algoritmos se hicieron evaluaciones en (Calik *et al.*, 2020) y (Bovy, 2020) enfocadas al consumo de recursos de cómputo y rendimiento general sobre dispositivos de bajos recursos.

Sin embargo, a la fecha de escritura de la tesis, no se encontró ningún trabajo que estudie cómo afecta la inclusión de estos algoritmos en el desempeño de red de un despliegue IoMT típico con nodos con recursos limitados.

1.2. Planteamiento del problema

Los cifradores ligeros presentan una serie de opciones que pueden tener gran potencial para cubrir varias de las vulnerabilidades que se encuentran actualmente en las redes IoMT.

La implementación de cifradores en redes IoT se ha realizado introduciendo una capa de seguridad entre las capas de aplicación y la de transporte. Esta capa de se-

guridad se encarga de establecer los enlaces seguros, para lo cual es necesario el intercambio de paquetes entre los nodos, además de realizar las operaciones de cifrado. En el contexto de las redes para la IoMT, sumar procesos y envío de paquetes a la operación de la red puede afectar su desempeño en el despliegue de aplicaciones como monitoreo remoto de pacientes.

Para la evaluación de los cifradores ligeros es importante considerar aspectos relevantes de las redes IoMT como la baja capacidad de procesamiento, tasas de transmisión de datos bajas y el consumo de energía (al estar operando la mayoría de los nodos sensores con baterías). Sin embargo, con el mejor de nuestros conocimientos, no se han hecho evaluaciones extensivas respecto a la sobrecarga que diferentes técnicas de cifrado ligero podrían llegar a tener en el desempeño global de una red IoMT típica. Esta sobrecarga se puede medir en términos de End to End delay, latencia, consumo de potencia, incremento en el ciclo de trabajo de los dispositivos, entre otras.

1.3. Justificación

Las redes IoMT pueden ser una tecnología clave para el despliegue de aplicaciones que permitan mejorar la salud de la población a través de un diagnóstico y seguimiento oportunos, esto mediante la habilitación de servicios como el monitoreo remoto de pacientes. Sin embargo, para lograr esto es importante garantizar los aspectos de seguridad de los datos transmitidos en una red IoMT.

Considerando las restricciones de batería, procesamiento y tasa de transmisión de las redes IoMT, es de suma importancia conocer la carga adicional que representa la introducción de la capa de seguridad a la pila de tecnologías comúnmente utilizada para el despliegue de este tipo de redes.

Por lo anterior, realizar una evaluación de los efectos que las técnicas de cifrado ligero pueden llegar a tener en el desempeño de una red IoMT con nodos de capacidades limitadas, es fundamental para el desarrollo de soluciones efectivas para la transmisión segura de datos fisiológicos de pacientes sobre este tipo de redes.

1.4. Hipótesis

Debido a las diferencias en el funcionamiento de diferentes cifradores y el proceso de establecimiento y mantenimiento de enlaces seguros, además de una diferencia en el desempeño en términos de seguridad, los cifradores ligeros tendrán un impacto diferente en el desempeño de una red IoMT típica. Una evaluación del desempeño de un red IoMT multisalto permitirá determinar qué técnicas de cifrado ofrecen el mejor compromiso de seguridad-desempeño dentro del contexto de redes IoMT multisalto.

1.5. Objetivos

1.5.1. Objetivo General

Obtener una evaluación de diferentes algoritmos de cifrado ligero y el algoritmo de cifrado estándar convencional AES en escenarios de redes IoMT desde el punto de vista de desempeño de la red.

1.5.2. Objetivos específicos

- Desarrollar un entorno de pruebas realista para la evaluación correcta de los parámetros de cada uno de los cifradores.
- Seleccionar 5 cifradores ligeros entre los más relevantes en la literatura para dispositivos IoMT.
- Implementar los cifradores ligeros en microcontroladores reales con capacidades limitadas.
- Realizar una comparativa del desempeño de red obtenido cuando se utilizan los diferentes cifradores ligeros seleccionados en redes IoMT típicas con fuentes de tráfico heterogéneas.

1.6. Metodología

1.6.1. Definición de los cifradores ligeros a evaluar

Debido a que muchos de los cifradores ligeros aún están en desarrollo, en esta etapa se analizó cuáles de los cifradores están listos para su implementación en dispositivos reales.

1.6.2. Definición de la tecnología a utilizar para la implementación

En esta etapa se estudiaron y definieron las diferentes tecnologías en las cuales se implementarán los diferentes cifradores ligeros, así como los protocolos de comunicación que se usarán.

1.6.3. Estudio y definición de las métricas a evaluar de la red multisalto

En esta etapa se hizo una investigación de las diferentes métricas que pueden ser evaluadas en una red multisalto y cuáles de ellas pueden ser afectadas por la implementación de cifradores ligeros.

1.6.4. Diseño e implementación de la cama de pruebas

En esta etapa se definió un par de escenarios de evaluación correspondientes a despliegues de redes IoMT con nodos de capacidades limitadas típicos. A partir de esto se diseñará e implementará una cama de pruebas basada en simulación y una cama de pruebas física basada en dispositivos reales.

1.6.5. Implementación de los cifradores ligeros

En esta etapa se implementaron todos los cifradores ligeros que se definieron en la primera etapa. Se tratará de utilizar la implementación recomendada de los diferentes autores en lenguajes de alta eficiencia como C o si es necesario lenguajes de descripción por hardware.

1.6.6. Evaluación de los cifradores ligeros

Se obtuvieron métricas de desempeño para cada uno de los cifradores ligeros a evaluar utilizando las camas de prueba desarrolladas.

1.6.7. Análisis y reporte de resultados

Se analizo las métricas obtenidas para determinar si alguno de los cifradores evaluados se destaca sobre los demás. Una vez realizado el análisis, se reportarán las conclusiones más relevantes en la tesis y otros foros de interés.

1.7. Organización de la tesis

En el capítulo 2 se muestra el funcionamiento de las tecnologías tanto de telecomunicaciones como externas que componen la red IoMT que se utilizó en este trabajo, así como la manera en cómo estas se interrelacionan para lograr una red completa.

En el capítulo 3 se repasan los conceptos clave de la seguridad en el internet de las cosas que fueron utilizados en este capítulo para posteriormente analizar los protocolos mediante los cuales estos conceptos fueron implementados para finalmente mostrar los algoritmos que se van a evaluar y cómo fue su proceso de selección.

En el capítulo 4 se muestra de manera detallada los diferentes escenarios y la implementación de todas las tecnologías mencionadas en los capítulos anteriores para la construcción de la cama de pruebas utilizada en este trabajo.

En el capítulo 5 se describen de manera detallada cada una de las distintas pruebas a las que fueron sometidos los algoritmos que se estaban evaluando. Esto se realizó en dos partes principales. La primera parte se enfoca en las pruebas para validar la cama de pruebas y poder obtener distintas métricas preliminares, y la segunda se enfoca en las pruebas destinada a obtener las métricas de red y de consumo de energía.

En el capítulo 6 se muestran los resultados de las pruebas que se llevaron a cabo en el trabajo de tesis. Al igual que el capítulo 5 este se dividió en dos partes principales.

La primera parte muestra los resultados de las pruebas preliminares y la segunda los resultados de las pruebas de métricas de red y consumo de energía.

Finalmente en el capítulo 7 se muestran las conclusiones, los aportes y el trabajo futuro que propone este trabajo de tesis.

Capítulo 2. Redes de sensores de área personal

2.1. Introducción

El avance de los últimos años en circuitos integrados y tecnologías en telecomunicaciones ha dado como resultado la posibilidad de poder conectar dispositivos ordinarios al Internet y poder ofrecer diversas aplicaciones y servicios. Esto conllevó a un nuevo paradigma conocido como el Internet de las cosas (IoT) (Larios *et al.*, 2013) (Sethi y Sarangi, 2017). Los dispositivos en una red IoT pueden interactuar entre ellos proporcionando diferente información o agregando capacidades a la red (Larios *et al.*, 2013).

Una parte importante de estos dispositivos están interconectados al Internet a través de redes inalámbricas de sensores (WSN por sus siglas en inglés)(Larios *et al.*, 2013). Las WSN consisten en redes de pequeños dispositivos que se despliegan en un ambiente físico. Cada dispositivo individual se llama nodo y tiene capacidad para poder comunicarse con sus nodos vecinos y sensar, almacenar y procesar una variable de interés (Larios *et al.*, 2013).

Como se mencionó en la introducción, uno de los aspectos más importantes que es necesario atender en el paradigma de la IoT es la seguridad de los datos transmitidos. Esto es particularmente importante en el caso de aplicaciones médicas de la IoT (IoMT) como la que se muestra en la Figura 1. En este escenario, los datos de los nodos sensores deberían de ser asegurados desde el momento de su generación hasta que son entregados al especialista. Esto, considerando las restricciones de poder de procesamiento y energía de los nodos en la IoMT. Este es el tema abordado en el presente trabajo de tesis.

El término Internet de las cosas médicas (IoMT) se refiere a la colección de todos los dispositivos y aplicaciones médicas que se conectan a la red para ofrecer servicios médicos (Besher *et al.*, 2021). Con el avance de las tecnologías en electrónica y telecomunicaciones, el número de dispositivos IoMT y sus aplicaciones han aumentado considerablemente y con ello el manejo de información altamente sensible. Si esta información se ve comprometida se pueden tener consecuencias desafortunadas o resultados no deseados que, en el entorno de IoMT, pueden afectar la salud de los

usuarios (Besher *et al.*, 2021). Por esta razón es importante proteger la confidencialidad e integridad de la información en los sistemas IoMT.

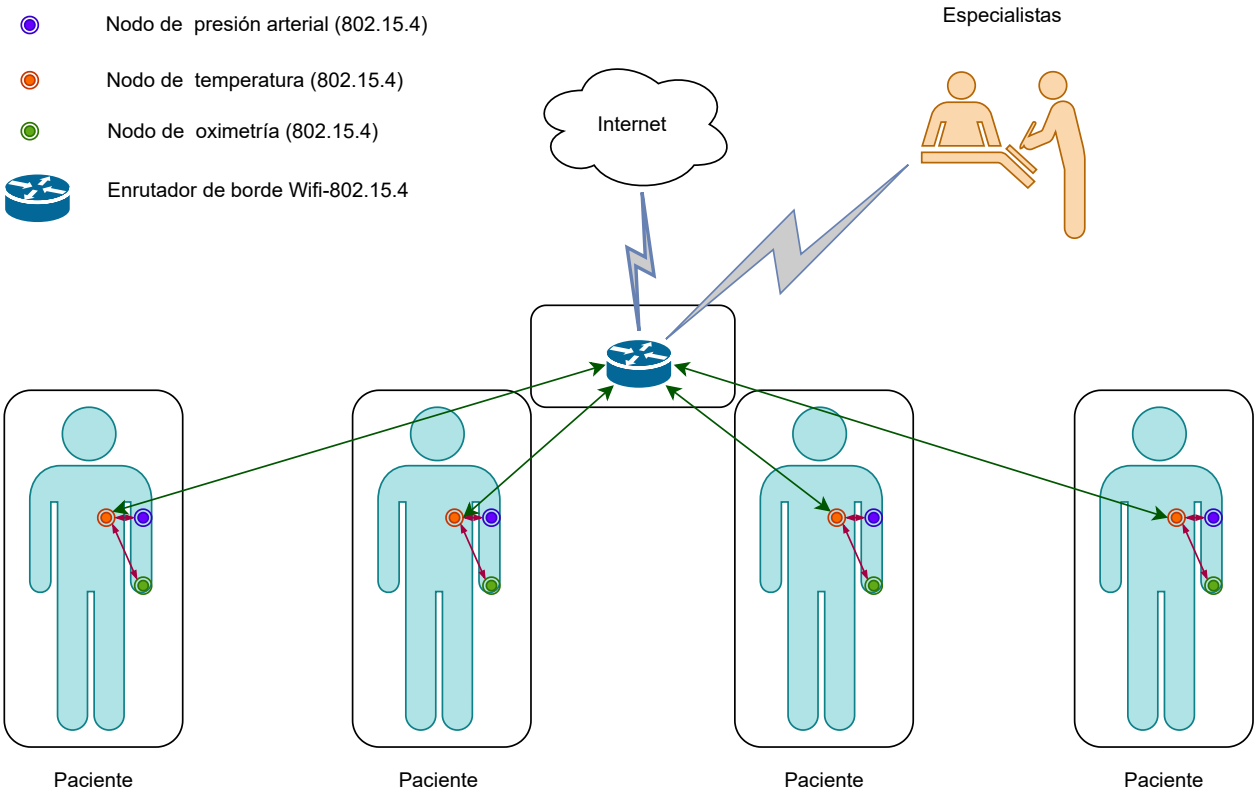


Figura 1. Ejemplo de una red WSN para IoMT.

Para explicar de mejor manera la contribución del presente trabajo en el área de seguridad para la IoMT, en este capítulo se dará una revisión de los aspectos más relevantes para esta investigación de las redes IoT e IoMT actuales y en el siguiente capítulo se detallarán los aspectos de seguridad en las redes IoMT.

2.2. Dispositivos IoT

Una parte fundamental de las redes IoT son los dispositivos que se utilizan, ya que es en estos dispositivos en donde se tienen que habilitar todos los servicios necesarios para que una red IoT funcione.

Los dispositivos IoT suelen ser clasificados en 3 diferentes (Bansal y Kumar, 2020),(Ojo *et al.*, 2018):

- **Dispositivos de gama alta**. Son aquellos dispositivos cuyos recursos computacionales son suficientes para poder correr sistemas operativos complejos como Linux o Android. Se utilizan principalmente como puertas de enlace o enrutadores de borde entre los servidores en la nube y los dispositivos sensores como se muestra en la Figura 1. Un ejemplo de estos pueden ser un teléfono móvil o una raspberryPI (Ojo *et al.*, 2018).

- **Dispositivos de gama media**. Los dispositivos de IoT de gama media son dispositivos con más restricciones de recursos en comparación con un dispositivo IoT de gama alta. Generalmente tienen su velocidad de reloj y RAM en el rango de cientos de MHz y KB, respectivamente. En las redes IoMT se utilizan como nodos sensores que toman muestras de complejidad intermedia como la oximetría o la presión arterial como se muestra en la Figura 1. Un ejemplo de esta gama de dispositivos es el arduino-MEGA y los Sensortag CC2650 utilizados en este trabajo de tesis y que detallaremos más (Ojo *et al.*, 2018).

- **Dispositivos de gama baja**. Define al grupo de dispositivos que están demasiado limitados en términos de recursos para ejecutar sistemas operativos tradicionales como Linux o Windows 10 IoT Core. Su memoria RAM y flash son de decenas o cientos de kilobytes y su arquitectura de procesamiento es de 8 bits o 16 bits con algunos dispositivos de última generación que admiten una arquitectura de 32 bits. Estos dispositivos se fabrican principalmente para aplicaciones básicas como el sensor de temperatura de un paciente que se muestra en la Figura 1. Se programan mediante el uso de firmware o una red de sensores inalámbricos de muy baja funcionalidad. Algunos ejemplos de estos dispositivos son el arduino-UNO o el OpenMote-B (Ojo *et al.*, 2018).

Este trabajo de tesis se centrará en los dispositivos de gama media y baja, los cuales están muy limitados en recursos computacionales. Por esta razón es importante analizar el impacto que tiene el agregar cualquier proceso extra tanto en uso de memoria como en uso de procesamiento. Es en este contexto que las soluciones de seguridad ligeras se han vuelto relevantes actualmente.

2.3. Redes IoT

Dentro de una red o arquitectura IoT se pueden considerar diferentes tecnologías para habilitar la conexión inalámbrica de los nodos hacia la Internet u otro resumidero de información como: Bluetooth Low Energy (BLE), Bluetooth, WiFi, 6LoWPAN, etc.

En la Tabla 1 se muestran algunas de las principales características técnicas con las que cuentan estas tecnologías. Si bien cada una de estas tecnologías tiene sus ventajas y desventajas, debido a el bajo consumo de energía con referencia a otros estándares, 6LoWPAN se puede considerar como un ejemplo típico de una tecnología habilitadora para aplicaciones IoT, (Bansal y Kumar, 2020), (Sethi y Sarangi, 2017). Por esta razón, en el presente trabajo se decidió utilizar 6LoWPAN como una tecnología típica para aplicaciones IoT en términos de consumo de energía, capacidad de procesamiento, conectividad, etc,(Sethi y Sarangi, 2017). En 6LoWPAN la información se puede transmitir de forma distribuída, utilizando uno varios saltos inalámbricos, lo cual hace más crítico proteger la información de posibles ataques maliciosos (p.ej. robo de información sensible).

Tabla 1. Principales características técnicas de las principales tecnologías inalámbricas.

Nombre del Standard	6LoWPAN (IEEE 802.15.4)	Bluetooth	Bluetooth Low Energy	Wi-Fi (IEEE 802.11)
Bandas de frecuencia	868/915 MHz - 2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz 5 GHz 6 GHz
Velocidad máxima de datos	20,40,250 Kbps	1,3,24 Mbps	1 Mbps	9608 Mbps
Alcance	10-100 m	1-100 m	1-10 m	20m
Caudal eficaz en aplicación (max)	151 Kbps	723Kbps (1.1) 2.1Mbps (2.0)	236 Kbps	600 Mbps
Número de nodos	65000	7+1	Limitado por aplicación	250

2.3.1. Redes 6LoWPAN

Las redes 6LoWPAN nacen de la idea de poder interconectar dispositivos de muy bajas prestaciones a la Internet, combinando el estándar de telecomunicaciones IEEE 802.15.4 y el protocolo de internet IPv6 (Olsson, 2014). El estándar 6LoWPAN permite crear redes con topologías complejas utilizando protocolos de red como el protocolo de “enrutamiento para redes con pérdidas y de bajo consumo” (RPL por sus siglas en

Inglés). En la Figura 2 se muestra la pila de protocolos utilizada en redes 6LoWPAN. Cabe resaltar la utilización de CoAP, DTLS y UDP en las capas superiores y la exclusión de HTTP, TLS y TCP. Esto se debe a las adaptaciones que se consideraron para proveer conectividad hacia la Internet de dispositivos con bajas prestaciones y bajas tasas de datos como los considerados en el estándar IEEE 802.15.4, tal cual se describe a continuación.

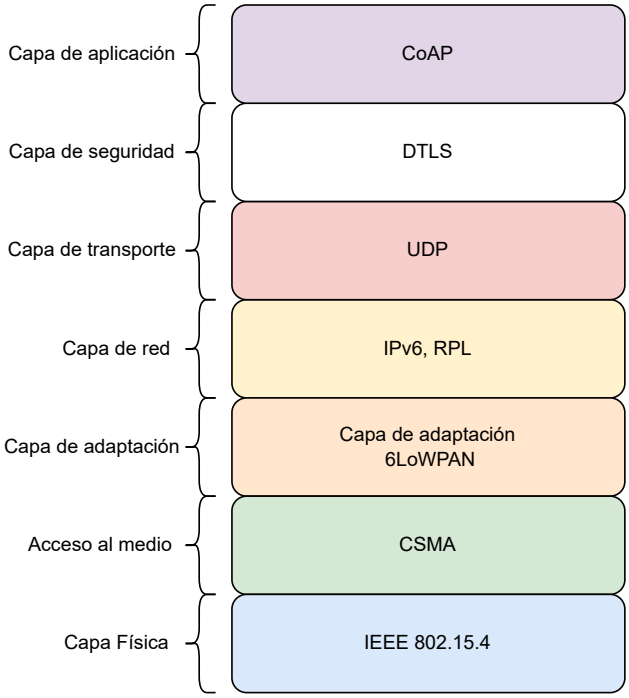


Figura 2. Pila de tecnologías utilizadas en una red 6LoWPAN.

2.3.1.1. Estándar IEEE 802.15.4

El estándar de telecomunicaciones IEEE 802.15.4 (IEEE Computer Society, 2020), provee una especificación para la capa física y la capa de enlace de datos de una red inalámbrica de área personal con baja tasa de datos (LR-WPAN por sus siglas en inglés).

Dependiendo de los requerimientos de la aplicación, las redes que siguen el estándar IEEE 802.15.4 puede tener las siguientes dos formas de comunicación:

- **Comunicación en estrella**. En una comunicación de estrella (ver Figura 3) la comunicación se establece entre los dispositivos a través de un único controlador central llamado coordinador PAN. Típicamente este coordinador será el único

dispositivo en la red que tendrá alimentación de una fuente principal como la red eléctrica del edificio aunque este no siempre es el caso, por lo que todos los demás nodos estarán alimentados por baterías y el ahorro en el consumo de estos es vital para su vida útil (IEEE Computer Society, 2020).

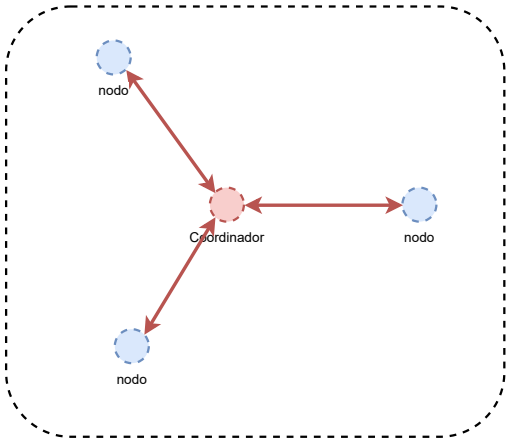


Figura 3. Comunicación en estrella del estándar 802.15.4.

- **Comunicación par a par**. En el caso de la topología par a par aunque existe un nodo coordinador, todos los dispositivos se pueden comunicar entre sí con todos los demás que tengan a su alcance sin la necesidad de usar al coordinador como intermediario, tal como se muestra en la Figura 4 (IEEE Computer Society, 2020). Estas redes son más complejas de implementar en comparación con las redes en estrella, sin embargo son utilizadas en aplicaciones como control y monitoreo industrial, redes de sensores, agricultura inteligente, etc (IEEE Computer Society, 2020). Como se explicará más adelante, esta topología también se caracteriza por permitir que dos dispositivos intercambien información utilizando múltiples saltos a través de la red (IEEE Computer Society, 2020) permitiendo habilitar estas características en las capas superiores.

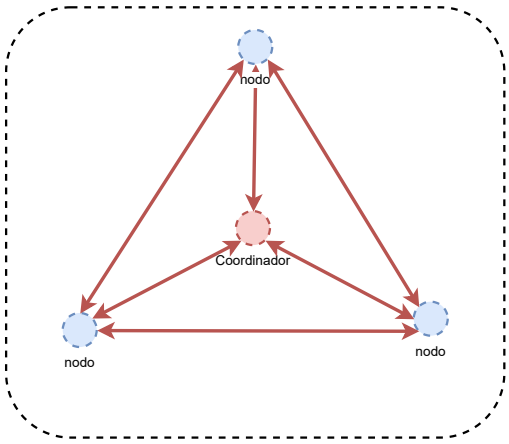


Figura 4. Comunicación par a par del estándar 802.15.4.

En cuanto a la capa física especifica que frecuencias del espectro electromagnético utilizar para la transferencia de información y con qué mecanismo de bajo nivel hacerlo. Específicamente determina el momento de activación y desactivación del transmisor, la detección de energía, la indicación de calidad de enlace, la selección de canal, enlace y transmisión así como la recepción de paquetes través del medio físico (IEEE Computer Society, 2020).

Las principales características que se destacan de la capa física de dispositivos IEEE 802.154 son las siguientes:

- Las frecuencias utilizadas:
  - 868-868.6 MHz (e.g. Europa)
  - 902-928 MHz (e.g. Norteamérica)
  - 2400-2483.5 MHz (e.g. a nivel mundial)
- En todas las bandas de frecuencia se utiliza la técnica de espectro ensanchado por secuencia directa (DSSS por sus siglas en ingles) con la finalidad de mitigar la interferencia de otras redes y tener mayor robustez frente al ruido. En la Tabla 2 se muestran los parámetros de transmisión para cada una de las bandas soportadas.

Tabla 2. Bandas de frecuencia y parámetros de transmisión para IEEE 802.15.4 (Casillas, 2012)

PHY (MHz)	Banda de frecuencia (MHz)	Parámetros de ensanchado		Parametros de transmision		
		Tasa de chips (kchips/s)	Modulación	Tasa de bits (Kbps)	Tasa de símbolos (ksímbolos/s)	Símbolos
868/915	868 - 868.6	300	BPSK	20	20	Binaria
	902 - 928	600	BPSK	40	40	Binaria
2450	2400 - 2483.5	2000	O-QPSK	250	62.5	Ortogonal

En la capa de control de acceso al medio (MAC por sus siglas en inglés) especifica dos servicios, el servicio de datos MAC y el servicio de administración de MAC. El servicio de datos MAC habilita la transmisión y recepción de las unidades de datos del protocolo MAC a través de la capa física. Otras de las tareas que realiza la capa MAC son administración de beacons, acceso al canal, gestión de intervalos de tiempo garantizados, validación de tramas, confirmación de recepción de datos, asociación y desconexión (IEEE Computer Society, 2020).

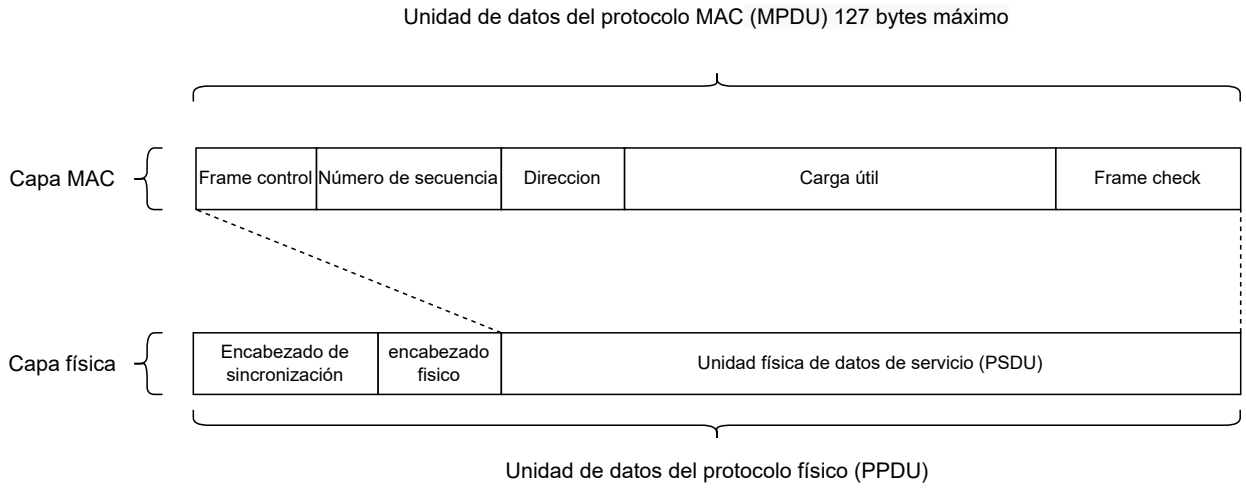


Figura 5. Estructura de una trama del protocolo IEEE 802.15.4.

Para llevar a cabo todas estas tareas se considera la trama como la unidad básica para el transporte de datos. Existen 4 tipos de estas tramas (datos, reconocimiento, beacon y tramas de comando). Estas utilizan la estructura que se muestra en la Figura 5, la cual consta de dos partes principales; la unidad de datos del protocolo físico (PPDU) la cual es la que finalmente se transfiere usando la capa física y que envuelve a la unidad de datos del protocolo MAC (MPDU) la cual no debe de tener una longitud mayor a 127 bytes y es donde se almacena la carga útil que se desea transmitir.

Para la transmisión de estas tramas se considera una estructura de supertrama (ver Figura 6) definida por el nodo coordinador y limitada por dos beacons (IEEE Computer Society, 2020). Esta supertrama se divide en dos partes principales; una porción de actividad en donde se realizan todas las transmisiones y una porción de inactividad en donde el coordinador y los nodos entran en modos de almacenamiento de energía. La porción de actividad es dividida a su vez en dos periodos; el periodo de acceso con contención (CAP) se debe de utilizar un protocolo de acceso múltiple por detección de portadora con evasión de colisiones (CSMA-CA) (Casillas, 2012) para transmitir todas las tramas y el periodo libre de contención (AFP) el cual puede tener una duración de 0 y se caracteriza por no utilizar CSMA-CA para la transmisión de tramas (IEEE Computer Society, 2020).

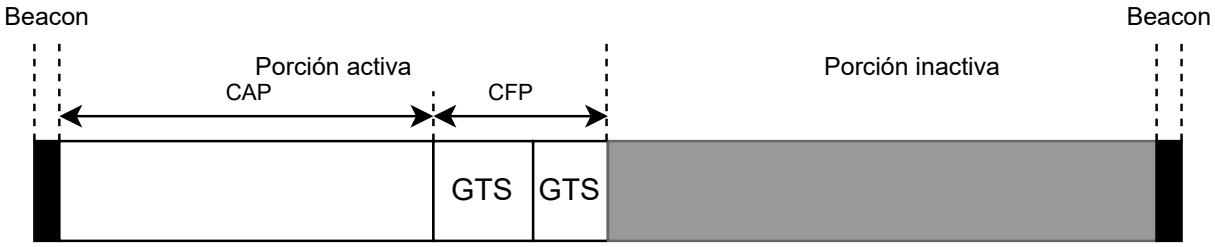


Figura 6. Ejemplo de una supertrama de 802.15.4.

2.3.1.2. Pv6 sobre LoWPAN (6LoWPAN)

Como se mencionó anteriormente 6LoWPAN nace de la idea de poder interconectar dispositivos de muy bajas prestaciones al Internet, combinando el estándar IEEE 802.15.4 y el protocolo de IPv6.

Como se muestra en la Figura 1 esta capa de adaptación se encuentra ubicada por encima de las capas que especifica el estándar IEEE 802.15.4 y por debajo de las capas del protocolo IPv6.

El principal problema de combinar estas tecnologías es que el tamaño del paquete de IPv6 puede ser de hasta 1280 bytes con 40 bytes de encabezado como se muestra en la Figura 6 mientras que el tamaño de un paquete de 802.15.4 es de sólo 127 octetos. Por lo tanto no es eficiente transmitir paquetes IPv6 directamente debido a que sería necesario fragmentarse en varias tramas, ya que solo los encabezados de estos paquetes ocuparían gran parte de la carga útil de una trama de IEEE 802.15.4.

Para resolver esta problemática la capa de adaptación de 6LoWPAN implementa las siguientes tres optimizaciones (Sethi y Sarangi, 2017):

- **Compresión del encabezado**. 6LoWPAN logra comprimir el encabezado de IPv6 eliminando algunas secciones del encabezado como las direcciones, ya que se pueden obtener desde la capa de enlace y de las transmisiones de paquetes de control (Olsson, 2014) (Hui y Culler, 2008).
- **Fragmentación y reensamblaje**. Como se mencionó anteriormente el máximo de bytes que se pueden transmitir en un paquete de IPv6 es de 1280 bytes mientras que el máximo tamaño de paquete de IEEE 802.15.4 es de 127 bytes, por lo tanto por medio de la capa de adaptación de 6LoWPAN se realiza una fragmentación en el caso en el que los paquetes de IPv6 superen el máximo de los paquetes 802.15.4 los cuales son reensamblados en el receptor (Hui y Culler, 2008) (Olsson, 2014).
- **Reenvío en capa de enlace**. La capa de adaptación también agrega la capacidad de poder reenviar paquetes entre nodos interconectados por el estándar 802.15.4 lo que ofrece la posibilidad de realizar rutas multisalto mediante la capa 3 tomando cada nodo como un salto de IP (Hui y Culler, 2008).

Una vez definido el funcionamiento principal de la capa de adaptación a continuación se describirá la capa de red que se consideró en la pila de protocolos.

2.3.1.3. Capa de Red IPv6

Esta capa es la que se va a encargar de manejar las direcciones IP y las rutas de toda la red (Regalado Jalca *et al.*, 2018), garantizando la comunicación tanto entre nodos contiguos como con nodos distantes en saltos múltiples. Como se mencionó en la sección anterior, esto se logra utilizando el protocolo de Internet versión 6 (IPv6) como protocolo de direccionamiento y como protocolo de enrutamiento en este trabajo de tesis se utiliza el protocolo de enrutamiento RPL (Olsson, 2014).

El protocolo de internet versión 6 (IPv6) es la versión más reciente del protocolo de internet el cual ofrece una mejora en el procesamiento de los paquetes y configuración

de las redes, las características más importantes a destacar para las redes 6LoWPAN son las siguientes:

- **Espacio de direcciones más grande**. En IPv4 el tamaño de una dirección es de 32 bits de largo lo que ofrece $2^{32} = 4,294,967,296$ direcciones posibles. En cambio las direcciones de IPv6 tienen una longitud de 128 bits lo que ofrece 2^{128} direcciones posibles, un número de 39 dígitos. Esto hace las direcciones de IPv6 prácticamente interminables, lo que beneficia en gran medida el apogeo que tienen las redes IoT actualmente al no tener limitaciones en la cantidad de dispositivos que pueden tener una dirección IPv6 única (Deering y Hinden, 1998).
- **Autoconfiguración de direcciones sin estado (SLAAC)**. Los clientes IPv6 se auto-configuran automáticamente. Esto significa que cada dispositivo tiene una dirección local de enlace que se autogenera basándose en características propias del hardware los cuales cuando se conectan a la red se realiza una resolución de conflictos en donde los enrutadores proporcionan prefijos de red a través de beacons denominados “anuncios de enrutadores” (Thomson *et al.*, 2007). Esta característica es especialmente útil en la configuración inicial de las redes 6LoWPAN dado que no se tiene que crear un procedimiento de asignación de direcciones en los enrutadores ahorrando tiempo y recursos en la configuración inicial.

2.3.1.4. Protocolo de enrutamiento RPL

El protocolo de enrutamiento IPv6 para redes con pérdidas y de bajo consumo (RPL por sus siglas en Inglés) especificado por el estándar RFC6550 (Winter *et al.*, 2012) se encarga de trazar las rutas entre los diferentes nodos de la red 6LoWPAN. Proporciona un mecanismo mediante el cual los paquetes de datos generados por los nodos en una red 6LoWPAN son enrutados hacia un sistema de distribución que provee conectividad a internet o en el sentido inverso, enrutando el tráfico proveniente desde el control central hacia los dispositivos destino dentro la red 6LoWPAN. Además de esto RPL también se utiliza para enviar paquetes entre nodos dentro la misma red 6LoWPAN (Olsson, 2014).

Este protocolo forma una topología de red de tipo árbol basándose en un grafo

acíclico dirigido (DAG por sus siglas en inglés) en donde todos los nodos tienen una ruta hacia el nodo raíz formando una DODAG (Destination Oriented DAG). Existen tres tipos de nodos en una red RPL (Cárdenas, 2019):

- El nodo raíz también llamado enrutador de borde (ver Figura 1). Tiene la particularidad de estar conectado directamente hacia el sistema de distribución y no estar dirigido a ningún otro nodo, debe existir al menos uno en la red. Comúnmente es un dispositivo de gama alta el cual debe de contar con una fuente de alimentación robusta ya que la cantidad de procesamiento que realiza será mayor al de los demás nodos (Cárdenas, 2019).
- Nodos enrutadores. Anuncian la información de la topología de red a sus vecinos y reenvían la información que les llega de otros nodos de acuerdo a la ruta establecida por el protocolo RPL. Un nodo raíz también puede ser nodo enrutador (Cárdenas, 2019).
- Por último se encuentran los nodos hoja los cuales son los que generan la mayoría de la información pero no tienen funcionalidades de enrutamiento (Cárdenas, 2019).

RPL admite dos diferentes modos de enrutamiento:

- **Modo de almacenamiento**. En el modo de almacenamiento todos los dispositivos de la red 6LoWPAN configurados con rango de enrutador o superior mantienen una tabla de enrutamiento y una tabla de vecinos. La tabla de enrutamiento se utiliza para buscar rutas a dispositivos y la tabla de vecinos se utiliza para realizar un seguimiento de los vecinos directos (Olsson, 2014), un ejemplo de este modo se puede observar en la Figura 7.

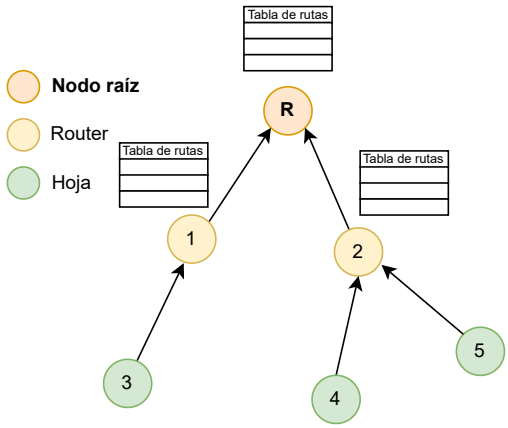


Figura 7. Modo almacenamiento de RPL.

- **Modo de no almacenamiento**. En este modo el único dispositivo con una tabla de enrutamiento es el enrutador de borde o nodo raíz (Olsson, 2014). Esta configuración tiene la ventaja de reducir la huella de memoria en los nodos que no son enrutadores dado que no tienen que almacenar la tabla de enrutamiento. Sin embargo en los enlaces descendentes, los cuales son los enlaces en donde el router envía un mensaje hacia un nodo dentro de la red, se debe de integrar toda la ruta en el encabezado de IPv6. Esto implica un aumento a lo largo de la trama que puede ser considerable para redes muy grandes. Un ejemplo de este modo se puede observar en la Figura 8. Cabe resaltar también que este es el modo seleccionado en la implementación de este trabajo de tesis.

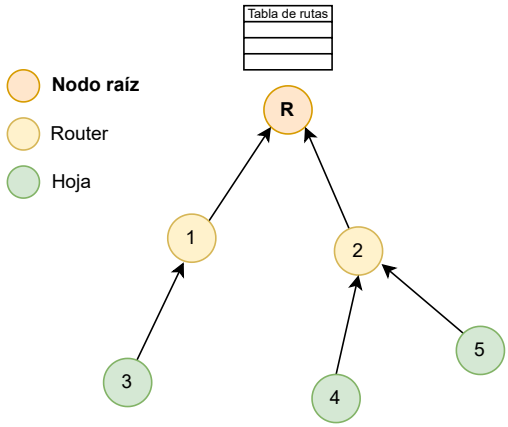


Figura 8. Modo de no almacenamiento de RPL.

2.3.2. Capa de Transporte

Al poder trazar rutas entre los diferentes nodos que existen en la red se requiere de una capa funcione de extremo a extremo y que permita administrar la transferencia de la información de las capas superiores por medio de las rutas y direcciones que ofrece la capa de red, a esta capa se le conoce como capa de transporte (Regalado Jalca *et al.*, 2018).

En el contexto de Internet existen dos protocolos principales usados en esta capa, el protocolo de control de transmisión TCP y el protocolo de datagramas de usuario UDP.

- El protocolo TCP es un protocolo orientado a conexión es decir que para intercambiar información de antemano se debe establecer una conexión entre el transmisor y el receptor. Está enfocado a ser usado para la comunicación confiable entre dos hosts, gracias al establecimiento de la conexión que ofrece la posibilidad de retransmisiones (Carrillo, 2017). Sin embargo esto puede causar una sobrecarga en las capas inferiores, es por esto que no se considera para el estándar 6LoWPAN.
- El protocolo UDP es un protocolo no orientado a conexión por lo que permite el envío de información en la red sin que se haya establecido una conexión, esto es debido que toda la información del direccionamiento se agrega a su encabezado. Debido a que no cuenta con un mecanismo de confirmación de entrega o recepción de paquetes, no tiene la capacidad de detectar la pérdida de paquetes y en transmisiones de muchos paquetes consecutivos estos pueden no llegar en orden sin embargo esto da flexibilidad para poder implementar diversas soluciones sobre capas superiores que se adapten de mejor manera a las aplicaciones en particular (Carrillo, 2017).

En el caso de las redes 6LoWPAN, el protocolo que se utiliza es UDP, debido a que éste es mucho más ligero tanto en procesamiento como en la longitud de su encabezado (Olsson, 2014).

2.3.3. Capa de seguridad (Opcional)

Por encima de la capa de transporte, las redes 6LoWPAN tienen la opción de incluir una capa que garantice la seguridad y autenticidad de la comunicación al cifrar la información de punto a punto y negociar las claves que se utilizaran al comienzo de una sesión de transmisión (Alamri, 2017).

Debido a que las redes 6LoWPAN utilizan UDP en la capa de transporte, la capa de seguridad que se debe utilizar debe seguir el estándar DTLS (Seguridad de la capa de transporte de datagramas). Este protocolo es una extensión del protocolo TLS (Seguridad en la capa de transporte) que agrega características para garantizar la correcta generación de claves en un protocolo no confiable como lo es UDP (Rescorla y Modadugu, 2012). Este estándar se analiza a más detalle en el siguiente capítulo (Alamri, 2017).

2.3.4. Capa de aplicación

Ninguna de las capas previamente descritas tendría una utilidad si no existiese una aplicación que requiera transmitir información a través de la red por medio de diferentes tipos de servicios implementados en los nodos. En el caso de las redes 6LoWPAN, la capa que se encarga de esto es la capa de aplicación (Regalado Jalca *et al.*, 2018). Además de permitir implementar los diferentes servicios que van a ofrecer los nodos a la red, la capa de aplicación se va a encargar de gestionar carencias de la capa de transporte como las pérdidas de paquetes.

Aunque existen diversos protocolos de aplicación como HTTP, estos ocasionan un gran aumento en el tamaño de los encabezados de las tramas. Debido a las limitadas tasas de datos que tienen las redes 6LoWPAN, no se puede permitir este tipo de sobrecarga. Además, los protocolos como HTTP no son compatibles con una capa de transporte no confiable como UDP. Por esta razón el IETF desarrolló el protocolo de aplicación restringida (CoAP). Este protocolo se describe con más detalle en el capítulo 4 (Alamri, 2017).

2.4. Sistemas operativos IoT

Debido a todas las tecnologías explicadas en la sección anterior en los últimos años los dispositivos IoT han estado volviéndose cada vez más complejos, además de que cada vez es más necesario garantizar la interoperabilidad de todos, no solo para poder acceder a internet, si no también para lograr que el desarrollo de aplicaciones sea los más rápido posible. Esto dio lugar a los sistemas operativos IoT. Un sistema operativo es un conjunto de programas desarrollados de antemano que funciona como intermediario entre el programa de la aplicación y el dispositivo físico en el que se quiere implementar (Bansal y Kumar, 2020).

En los dispositivos IoT un sistema operativo debe de ser lo suficientemente capaz de encargarse que el programa se pueda ejecutar en el dispositivo y pueda acceder a los recursos del mismo, garantizando que mientras el dispositivo sea compatible con el sistema operativo podría ser compatible con el programa (Bansal y Kumar, 2020).

Además de esto, los sistemas operativos IoT también deben de ser responsables de gestionar y monitorear el consumo de energía, habilitar las instrucciones a ejecutar para programar el dispositivo, y habilitar los recursos para poder comunicarse con los otros (Bansal y Kumar, 2020).

Y dado que los dispositivos IoT no son iguales, los sistemas operativos IoT deben de ser lo suficientemente flexibles para adaptarse a los requisitos de cada sistema sub-IoT (Bansal y Kumar, 2020).

Los sistemas operativos IoT se pueden clasificar como:

- IoT OS de gama alta: la mayoría de estos sistemas están basados en linux, estos sistemas operativos pueden correr sobre dispositivos de gama media y alta que son dispositivos con altos recursos en memoria y potencia, como por ejemplo las Raspberry PI (Bansal y Kumar, 2020).
- IoT Os de gama baja: Estos sistemas operativos pueden o no estar basados en linux y corren generalmente en dispositivos de gama media y baja, que son dispositivos con memoria y potencia limitada, como las plataformas de arduino (Bansal y Kumar, 2020), es en estos últimos en donde se ha concentrado el avance del

desarrollo de IoT en los últimos años.

Como se mencionó anteriormente este trabajo de tesis se enfocará en dispositivos de gama media y baja por lo que se utilizará un IoT OS de gama baja para la implementación de las diferentes tecnologías que se utilizarán, el proceso de selección para el sistema operativo que más se adaptó a las necesidades de este proyecto se describe más adelante.

2.5. Sumario

Como se explicó en este capítulo la mayor parte de los dispositivos IoT poseen capacidades limitadas en cuanto al espacio en memoria y potencia de procesamiento por lo que la implementación de protocolos con algoritmos de cifrado que se utilizan convencionalmente en internet no es una opción del todo viable debido a la gran cantidad de procesamiento que requieren (Olowu *et al.*, 2014). Es por esta razón que se pretende que el desarrollo de los algoritmos de cifrado ligero sean una gran oportunidad para poder implementar seguridad en este tipo de dispositivos de una manera eficiente y sin sacrificar demasiados recursos (Singh *et al.*, 2017). Por lo tanto en el capítulo siguiente se profundizará acerca de cómo se encuentra la seguridad actual en las redes IoT y los cifradores ligeros, además de que se mostrarán las soluciones que se están llevando a cabo a nivel de estándar para poder seleccionar a los algoritmos más adecuados.

Capítulo 3. Criptografía ligera

3.1. Introducción

Como se mencionó en los Capítulos 1 y 2, uno de los retos más grandes que enfrentan las tecnologías IoT, y en especial las de IoMT, es la seguridad de la información (Bansal y Kumar, 2020). Esto se debe principalmente a que la mayoría de los dispositivos y aplicaciones IoT tienen limitaciones en cuanto al poder de procesamiento y el consumo de energía.

En este capítulo se analizará el estado actual de la seguridad en redes IoT así como las nuevas tecnologías que se están proponiendo para resolver la actual carencia de soluciones en este aspecto. Posteriormente se analizará el funcionamiento de los cifradores ligeros, los cuales son una de las soluciones más importantes y prometedoras en este aspecto.

3.2. Confidencialidad, Integridad y Disponibilidad

Cuando se habla de seguridad en la red se deben considerar los aspectos de la tríada Confidencialidad, Integridad y Disponibilidad (CIA por sus siglas en inglés) (Olowu *et al.*, 2014), la cual es una de las consideraciones para guiar las políticas de seguridad de la información dentro de una organización.

En el diseño de soluciones IoMT también se debe de considerar la tríada CIA (ver Figura 9), esto debido a la naturaleza propia de la información transmitida en este tipo de aplicaciones. Por esta razón a continuación se resume brevemente cada uno de los conceptos considerados en la tríada CIA a manera de referencia. Cabe resaltar que el presente trabajo de investigación se centra en la evaluación de cifradores para aplicaciones en redes IoMT. Por está razón, el resto del capítulo se enfocará en describir la implementación actual de técnicas de cifrado en redes 6LoWPAN y la descripción de cifradores ligeros propuesto para aplicaciones IoT.

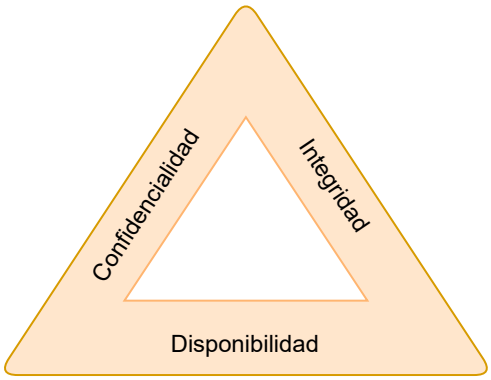


Figura 9. Tríada CIA.

3.2.1. Confidencialidad

La confidencialidad significa que solo las personas o sistemas autorizados pueden ver la información confidencial o clasificada. Los datos que se envían a través de la red no deben ser accedidos por personas no autorizadas. Un atacante puede intentar interceptar los datos utilizando diferentes herramientas disponibles en Internet y obtener acceso a su información. Uno de los métodos principales para tratar de evitar el robo de información es la utilización de técnicas de cifrado para proteger los datos. En particular, el objetivo de las técnicas de cifrado es que incluso si el atacante obtiene acceso a los datos, no podrá interpretarlos o utilizarlos si no se tiene acceso a los mecanismos y claves de descifrado.

3.2.2. Integridad

En términos de seguridad de la información, la integridad hace referencia a la modificación de la información o recursos. Normalmente se expresa en lo referente a prevenir el cambio no autorizado, esto incluye insertar, borrar o sustituir datos. Por lo tanto, el objetivo de la integridad es prevenir modificaciones no autorizadas de la información.

Cabe resaltar que, en relación con integridad, en un sistema IoMT se pueden distinguir dos vertientes:

- **Integridad de los datos.** Implica asegurar que los datos no son modificados durante su transmisión.

- **Integridad del origen.** Implica garantizar que los datos provengan de una fuente fidedigna.

3.2.3. Disponibilidad

En términos de seguridad de la información, la disponibilidad hace referencia a que la información del sistema debe permanecer accesible a elementos autorizados. El objetivo de la disponibilidad es prevenir accesos no autorizados/controlados de los recursos informáticos.

En términos de seguridad informática “un sistema está disponible cuando su diseño e implementación permite deliberadamente garantizar el acceso a datos y servicios determinados a usuarios autorizados, mientras se niega el acceso a usuarios no autorizados ” (Olowu *et al.*, 2014).

A manera de resumen, la seguridad en sistemas IoT debe buscar mantener un equilibrio adecuado entre confidencialidad, integridad y disponibilidad. Por ejemplo, no tendría sentido garantizar la confidencialidad para un archivo si eso implica que nadie puede acceder a él, ya que en este caso se estaría negando la disponibilidad. Sin embargo, dependiendo del entorno de trabajo y sus necesidades se puede dar prioridad a un aspecto de la seguridad o a otro. Por ejemplo, en ambientes militares suele ser siempre prioritaria la confidencialidad de la información frente a la disponibilidad. En contraste, en ambientes bancarios, muchas veces es prioritaria la integridad de la información frente a la confidencialidad o disponibilidad (p.ej. se considera menos dañino que un usuario pueda leer el saldo de otro usuario a que pueda modificarlo).

Considerando lo anterior, se puede mencionar que el presente trabajo de tesis se sitúa principalmente dentro del marco de confidencialidad e integridad de la tríada CIA. Esto debido a que se estudiarán las implementaciones de cifradores específicamente diseñados para entornos IoT. Por lo tanto, el resto del capítulo se enfocará en describir los mecanismos de cifrado en redes IoT considerados dentro del presente trabajo, asumiendo que los aspectos de disponibilidad se abordarán en otra instancia. En este sentido, a continuación se describe el estado del arte de las implementaciones de mecanismos de cifrado en redes IoT.

3.3. Cifrado en Redes IoT

En la actualidad se han propuesto diversos mecanismos para resolver la carencia de seguridad que tienen las tecnologías IoT. Dentro de estos mecanismos uno de los más importantes es el estudiado en el presente trabajo de investigación, el cual aborda la implementación de técnicas de cifrado con el fin de cubrir aspectos de seguridad e integridad de la información en la transmisión de datos sobre redes IoT.

A groso modo se pueden distinguir dos elementos importantes en un sistema de cifrado para aplicaciones IoT. El primero consiste en los métodos y protocolos que permiten el establecimiento de un flujo de información cifrado entre una fuente y un receptor. Este aspecto se aborda con la implementación del protocolo de seguridad de la capa de transporte de datagramas (DTLS por sus siglas en inglés). El segundo elemento consiste en el cifrador particular a utilizarse en las aplicaciones IoT. Debido a las características propias de los nodos en una red IoT típica (p.ej. un nodo 6LoWPAN), se busca que la implementación del cifrador no represente una sobrecarga significativa en el nodo y mantenga un buen nivel de seguridad.

A continuación se describirán los aspectos del protocolo DTLS más relevantes para este trabajo de tesis. Después se proporcionará una descripción de los cifradores considerados y de la lógica seguida para su elección.

3.3.1. Protocolo DTLS para aplicaciones de IoT

El protocolo DTLS busca asegurar la comunicación entre los dispositivos mediante la introducción de protocolos de comunicación seguros. Este protocolo utiliza mecanismos de cifrado previamente definidos para garantizar que la información no sea vulnerada por alguien capaz de interceptar los datos en medio de una transmisión (Bansal y Kumar, 2020). El protocolo DTLS es una adaptación del protocolo TLS para ser utilizada sobre el protocolo UDP, el cuál es utilizado en 6LoWPAN.

3.3.2. TLS

El protocolo de seguridad en la capa de transporte (TLS por sus siglas en Inglés) es el protocolo más utilizado actualmente para la seguridad del correo electrónico y la mayoría de las páginas web (Vignesh, 2017).

Está diseñado para proveer de confiabilidad e integridad a la información entre dos entidades comunicándose entre sí. Para establecer un flujo de comunicación cifrado entre dos entidades, un nodo manda al otro una petición de comunicación segura. Al nodo que solicita establecer el flujo seguro se le denomina cliente, mientras que al nodo receptor se le denomina servidor. TLS consta de dos mecanismos principales, el mecanismo de registro TLS y el mecanismo de "handshake"de TLS (Vignesh, 2017):

- El mecanismo de handshake es el que se encarga de establecer los parámetros con los cuales operará el protocolo de registro. Algunos de estos parámetros son: la velocidad de sesión, la versión de protocolo que se va a utilizar, y el algoritmo de cifrado que se va a utilizar. Si en una implementación de TLS se decide utilizar un mecanismo de clave pública, entonces durante el procedimiento de handshake se intercambian los secretos para generar las claves compartidas de los algoritmos de cifrado (Vignesh, 2017).
- El mecanismo de registro de TLS se va a ejecutar después de que el mecanismo de Handshake haya establecido los parámetros de la sesión. Este mecanismo se encarga de fragmentar, comprimir (si es necesario), cifrar y descifrar la información que se manden las entidades (Vignesh, 2017).

El protocolo TLS está diseñado para trabajar sobre el protocolo de control de transmisión (TCP por sus siglas en Inglés). Esto se debe a que TLS depende de las retransmisiones de TCP en el caso de que se pierda algún paquete durante el procedimiento de handshake.

3.3.3. DTLS

Como se mencionó anteriormente el protocolo TLS solo es compatible con el protocolo de control de transmisión TCP. Sin embargo, algunas entidades (por ejemplo

los nodos en una red 6LoWPAN) únicamente soportan el protocolo de datagramas de usuario (UDP por sus siglas en Inglés). Por está razón se desarrolló una variante de TLS denominada DTLS (Vignesh, 2017). Para poder implementar el protocolo TLS en una red UDP se afrontaron dos retos importantes (Vignesh, 2017):

- TLS no permite el descifrado de fragmentos individuales. Esto debido a que su verificación de integridad depende directamente del número de secuencia (Vignesh, 2017).
- El mecanismo de handshake de TLS asume que todos los paquetes se entregan de manera confiable. Por está razón TLS no tiene un mecanismo de retransmisión en caso de que los paquetes se pierdan. Esto puede ocasionar que en el mejor de los casos el handshake no se pueda llevar a cabo (Vignesh, 2017).

Estos problemas son importantes al realizar el handshake, por lo que DTLS corrige estos estos obstáculos con las siguientes funcionalidades:

- Para el problema de confiabilidad DTLS considera la implementación de un temporizador de retransmisión. Durante la fase inicial del handshake de DTLS, el cliente envía un mensaje de saludo hacia el servidor. Después, el cliente espera recibir una solicitud de verificación de saludo durante un periodo de tiempo predefinido. Si el cliente no recibe la solicitud de verificación de saludo antes de finalizar el periodo de espera, el temporizador expira se asume que se ha perdido el mensaje de saludo. El cliente retransmite el mensaje de saludo y cuando el servidor recibe la retransmisión sabe que debe transmitir nuevamente el mensaje de verificación de saludo (Rescorla y Modadugu, 2012).
- A cada mensaje del handshake se le asigna un número de secuencia específico. El nodo que recibe este mensaje de reconocimiento determina si es el siguiente mensaje que espera o no. Si el mensaje recibido es el mismo que el esperado, lo procesa. De lo contrario pone en cola el mensaje para el manejo futuro una vez que se hayan recibido todos los anteriores (Rescorla y Modadugu, 2012).
- El tamaño de algunos de los mensajes de handshake de DTLS son grandes en comparación con el tamaño de los datagramas UDP. Para abordar este problema,

cada mensaje del mecanismo de handshake de DTLS se fragmenta en varios paquetes de DTLS, cada uno de los cuales debe caber en un único datagrama de IP. Cada fragmento contiene tanto su desplazamiento como su longitud. Por tanto, el destinatario en posesión de todos los bytes de un mensaje de reconocimiento puede volver a ensamblar el mensaje original sin fragmentar (Rescorla y Modadugu, 2012).

3.3.4. Handshake de DTLS

Para completar el procedimiento de handshake y empezar el intercambio de datos seguro es necesario el intercambio de los paquetes mostrados en la Figura 10.

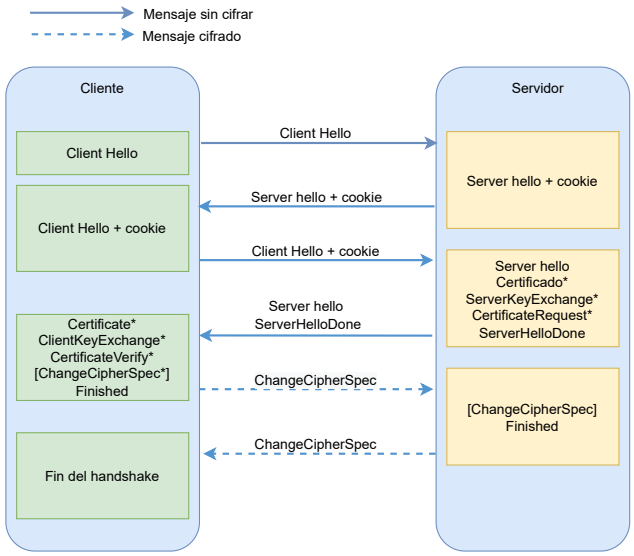


Figura 10. Proceso de Handshake de DTLS.

La comunicación es iniciada por un nodo al que se le llamará cliente. Este nodo manda un mensaje denominado “ClientHello” en el cual hace la petición al nodo receptor para establecer un flujo de comunicación seguro. El nodo que recibe el mensaje se le denomina servidor.

Es importante mencionar que si uno de estos paquetes no es recibido, no se puede completar el procedimiento de handshake. Por esta razón es que DTLS considera la retransmisión de paquetes. Por ejemplo, si el paquete ClientHello no es recibido por el servidor, esté no mandará el paquete ServerHello. Al no recibir el cliente este

paquete, realizará una retransmisión del paquete ClientHello. El número máximo de retransmisiones de cada paquete es de 4. Pasado este número de intentos con cualquiera de los paquetes se declara que no se puede establecer la comunicación segura y es necesario reiniciar el proceso.

El contenido de estos paquetes se describe a continuación. Cabe destacar que esta descripción se corresponde con el método de autenticación de clave pre-compartida PSK (“pre shared key”), que es la que se utilizó en este trabajo de tesis.

Primera fase. En la primera fase el cliente y el servidor intercambian paquetes tipo “hello” antes de hacer el intercambio de las claves necesarias para establecer el flujo seguro. Al nodo que envía el paquete “ClientHello” se le llama “cliente” y al nodo que recibe el paquete se le llama “servidor”. La Figura 11 muestra la estructura del paquete “ClientHello”.

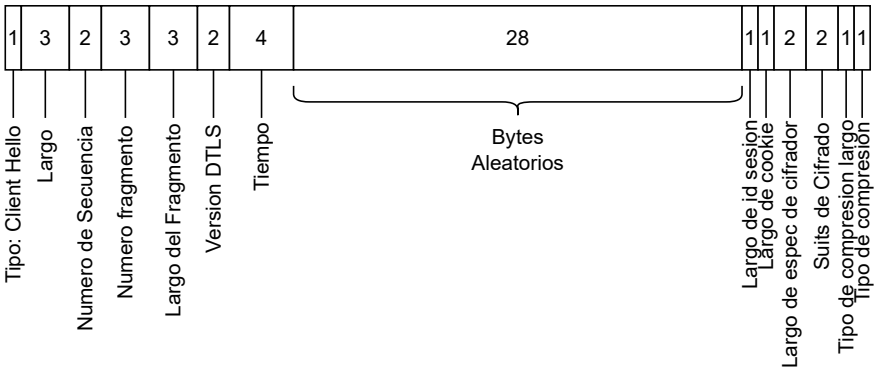


Figura 11. Estructura del paquete ClientHello.

Este paquete contiene encabezados del estándar como la longitud, el número de secuencia el cual siempre deberá de ser 0, número de fragmento, etc. Es de destacar la “carga útil” del paquete que consiste de 28 Bytes aleatorios los cuales se van a utilizar más adelante para la generación de claves.

Cuando el servidor recibe el paquete “Client Hello”, lo procesa y le responde al cliente con un paquete “HelloVerifyRequest” que se muestra en la Figura 12. Este paquete tiene de carga útil una “cookie” de 16 bytes aleatoria la cual el cliente tendrá que responder, esto con el fin de que no se manden solicitudes indiscriminadas de handshake y se procesen con costosas operaciones criptográficas. Esta “cookie” es

generada por un algoritmo que debe cumplir ciertas características para no poder ser duplicado al relacionar ambos dispositivos fuente.

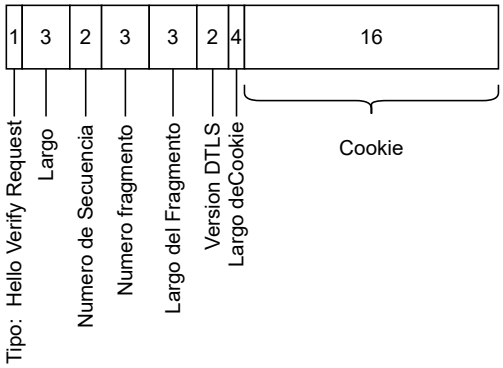


Figura 12. Estructura del paquete HelloVerifyRequest.

Cabe destacar que este procedimiento se diseñó de esta manera en DTLS con el objetivo de frenar ataques de DDOS cuando el nodo servidor tiene capacidades de cómputo considerables. Note que en nuestro caso, el servidor será un nodo con capacidades de cómputo reducidas, por lo cual incluso el recibir varias peticiones de handshake simultáneas le podría ocasionar problemas, aunque ninguna de ellas avance más en el procedimiento de handshake.

Una vez que el cliente haya recibido el paquete “HelloVerifyRequest”, este enviará un nuevo paquete “Clienthello” como el que se muestra en la Figura 13. En este paquete se incluyen otros 28 bytes aleatorios que se utilizarán más adelante para la generación de claves. Además, se incluirá la cookie que le envió el servidor anteriormente en el paquete “HelloVerifyRequest” (Karn y Simpson, 1999).

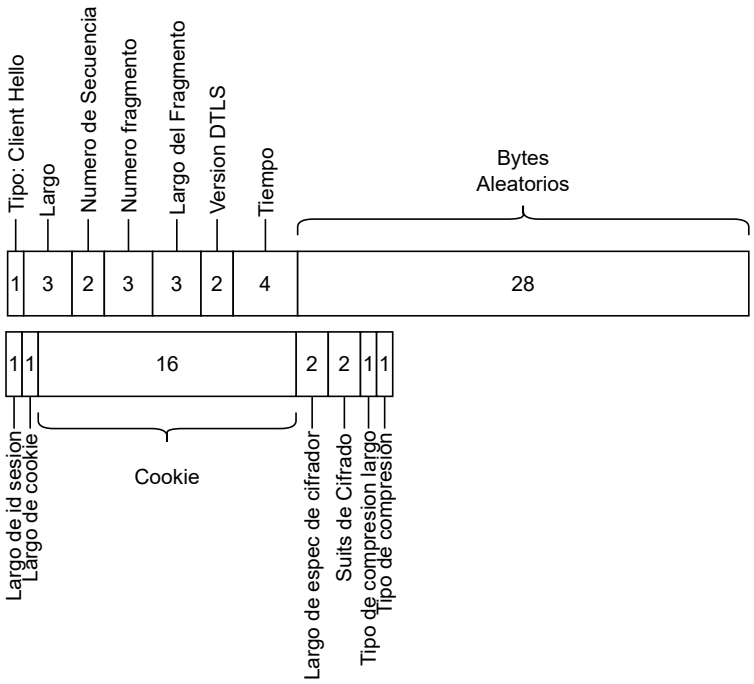


Figura 13. Estructura del segundo paquete ClientHello.

Cuando se recibe el segundo “ClientHello”, el servidor puede verificar que la “Cookie” es válida y el cliente puede recibir paquetes en la dirección IP indicada. Para evitar la duplicación del número de secuencia en caso de múltiples intercambios de “cookies”, el servidor debe usar el número de secuencia de registro en “ClientHello” como el número de secuencia de registro en su “ServerHello” inicial.

Segunda fase Una vez que el servidor verifica que la cookie es auténtica, revisará si existe compatibilidad con el cifrador y enviará un paquete “ServerHello” (ver Figura 14-a) con una carga útil de 28 bytes aleatorios que servirán para generar las claves y el identificador que especificará la suite de cifrado que se va a utilizar.

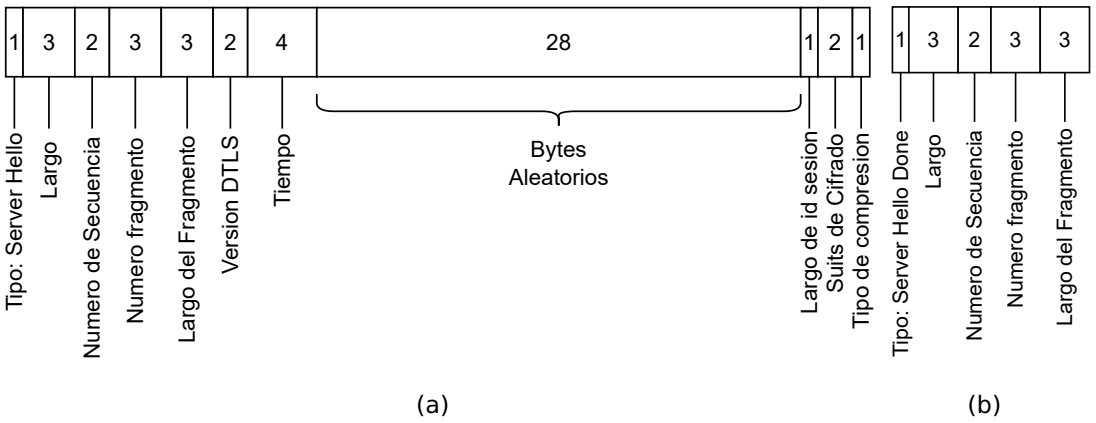


Figura 14. a) Estructura del paquete ServerHello b) Estructura del paquete ServerHelloDone.

Posteriormente enviará el paquete “ServerHelloDone” que se muestra en la Figura 14-b con el cual le indica al cliente que está listo para proceder con la negociación.

En cuanto el cliente reciba el paquete “HelloDone”, enviará un paquete denominado “ClientKeyExchange” como el que se muestra en la Figura 15 con la identidad del dispositivo servidor.

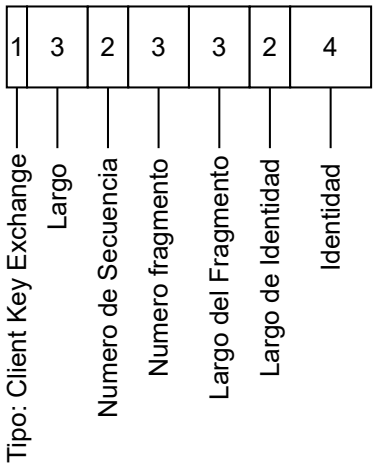


Figura 15. Estructura del paquete ClientKeyExchange.

En este punto tanto el cliente como el servidor empiezan a generar sus claves de la siguiente manera. lo primero que hacen es generar una clave denominada “premaster_secret” de la siguiente manera:

Partiendo que al momento de programar el dispositivo a este se le asigna una clave precompartida la cual tendrá una longitud N se va a concatenar o juntar con los

siguientes datos:

- Un entero de 16 bits de valor N.
- N octetos en 0.
- Un entero de 16 bits de valor N.
- La clave precompartida (PSK).

Ahora usando una función pseudoaleatoria denominada PRF, el cliente y el servidor calculan la clave llamada “master-secret” de la siguiente manera:

$$master\_secret = PRF(premaster\_secret,\ bytes\_aleatorios\_del\_cliente,\ bytes\_aleatorios\_del\_servidor)$$

Tanto los bytes aleatorios del cliente como los del servidor son los que se intercambiaron anteriormente.

Finalmente con la clave secreta maestra, que tiene 48 bytes de longitud el cliente y el servidor crean un conjunto de 3 claves para el servidor y 3 claves para el cliente:

- client_write_MAC_key: verificación de autenticación e integridad
- server_write_MAC_key: verificación de autenticación e integridad
- client_write_key: cifrado de mensajes mediante clave simétrica
- server_write_key: cifrado de mensajes mediante clave simétrica
- client_write_IV: Vector de inicialización utilizado por algunos cifrados AEAD
- server_write_IV: Vector de inicialización utilizado por algunos cifrados AEAD

Estas son las claves que se van a utilizar para cifrar y autenticar la información durante la sesión.

Fase final. En este punto, el cliente está listo para cambiar a un entorno seguro y cifrado. Todos los datos enviados por el cliente a partir de ahora se cifran mediante la clave compartida simétrica.

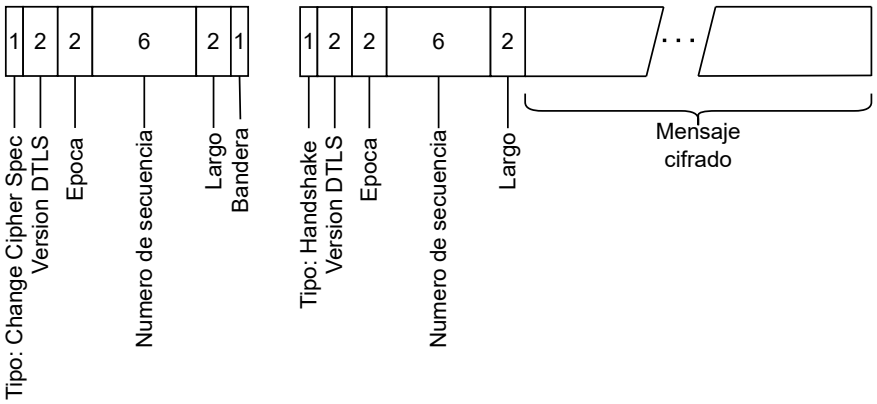


Figura 16. Estructura del paquete ChangeCipherSpec del cliente.

Posteriormente el cliente envía el paquete ChangeCipherSpec (cuya estructura se muestra en la Figura 16) el cual es el último mensaje del proceso de handshake del cliente, significa que el handshake ha finalizado. Este es también el primer mensaje encriptado de la conexión segura.

El servidor también está listo para cambiar a un entorno cifrado. Todos los datos enviados por el servidor a partir de ahora se cifran utilizando la clave compartida simétrica. Por lo que envía su propio paquete ChangeCipherSpec (que se muestra en la Figura 17) con datos ya cifrados, el cual es el último mensaje de todo el proceso de handshake y significa que el protocolo de handshake ha finalizado.

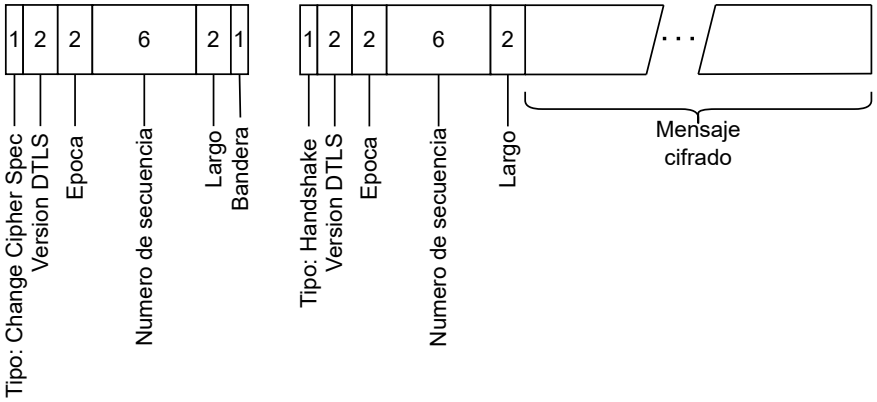


Figura 17. Estructura del paquete ChangeCipherSpec del servidor.

3.4. Cifrado ligero

La mayoría de los dispositivos IoT se caracterizan por tener pocos recursos para ejecutar grandes operaciones de cálculo, por lo tanto el uso de algoritmos de cifrado convencionales da como resultado un alto consumo de energía y un alto tiempo de procesamiento. Por esta razón, la ejecución de algoritmos convencionales puede comprometer el tiempo de operación autónoma de nodos que se alimentan con baterías e introducir un aumento significativo en el retardo del envío de paquetes (Aguirre, 2020). Debido a estos inconvenientes, se han estado desarrollando nuevos algoritmos de cifrado capaces de ser ejecutados en dispositivos de bajas prestaciones (como los utilizados en redes IoT) sin sacrificar la seguridad de la red. Estos algoritmos son conocidos como cifradores ligeros (Aguirre, 2020).

3.4.1. Clasificación de cifradores ligeros

Todos los algoritmos de cifrado, incluidos los algoritmos de cifrado ligero, parten de operaciones criptográficas simples denominadas primitivas criptográficas. Una primitiva criptográfica se puede definir como una función de bajo nivel, la cual actúa como los bloques de construcción para poder construir esquemas de cifrado más complejos (Bovy, 2020). Principalmente, existen cuatro tipos de primitivas para algoritmos de cifrado ligero: permutación, cifrador de bloque, cifrador de flujo y cifrador de bloque modificable (Bovy, 2020). Estas primitivas se muestran en la Figura 18 y se detallan a continuación.

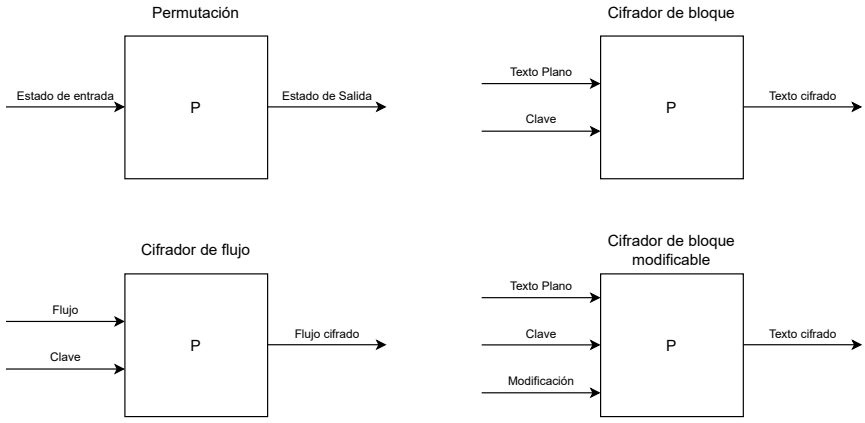


Figura 18. Primitivas más utilizados en algoritmos de cifrado ligero.

3.4.1.1. Permutación

Una primitiva de permutación consiste en una función que convierte una entrada S en una salida $\{0,1\}^b$, es decir, aplica una operación de permutación específica en la entrada y la entrega como salida, el largo en bits en la salida será igual al largo de bits en la entrada (Bovy, 2020).

3.4.1.2. Cifrador de bloque

Un cifrador de bloque es un conjunto de permutaciones indexadas por una clave. Tiene dos entradas: la cadena de n bits que se busca cifrar referida como “texto plano” y la clave k bits. El cifrador de bloque provee una salida de la misma longitud n que el texto plano (Bovy, 2020).

3.4.1.3. Cifrador de bloque modificable

Un cifrador de bloque modificable es un cifrador de bloque indexado por una modificación. Tiene tres entradas: un texto plano de n bits, una clave de k bits y una modificación. El cifrador de bloque modificable entrega una salida de la misma longitud n que el texto plano (Bovy, 2020).

3.4.1.4. Cifrador de flujo

Un cifrador de flujo tiene 2 entradas: la primera recibe un flujo de bits; la segunda es la clave para cifrar los datos. Un cifrador de flujo entregará datos en su salida conforme se van introduciendo en su entrada bit a bit (Bovy, 2020).

3.4.2. Niveles de seguridad en los algoritmos de cifrado

Los algoritmos de cifrado, incluyendo los algoritmos de cifrado ligero, pueden clasificarse dependiendo del nivel de seguridad que estos proveen a los datos. Este nivel de seguridad se estima principalmente por el largo de su clave de cifrado. Suponiendo que el algoritmo de cifrado solo se puede romper mediante una búsqueda exhaustiva

de la clave utilizada para el cifrado, se dice que el algoritmo de cifrado ofrece una nivel de seguridad de n cuando la clave tiene una longitud de n-bits (Lenstra, 2007). Un ejemplo es el algoritmo de cifrado AES-256 que se utiliza ampliamente en la actualidad. Este algoritmo tiene un nivel de seguridad de $n = 256$ ya que la clave con la que cifra sus datos tiene una longitud de 256 bits. Esto implica que si un atacante quiere romper el cifrado por fuerza bruta, entonces tendrá que buscar la clave en 2^{256} combinaciones posibles, lo cual tomaría millones de años utilizando las mejores computadoras de la actualidad. Sin embargo, el nacimiento de nuevas tecnologías y el incremento en la capacidad de cómputo, hace que con el transcurso del tiempo el nivel de seguridad cada vez represente una menor cantidad de seguridad real efectiva (Lenstra, 2007).

Como se presentará más adelante, la mayoría de los algoritmos de cifrado ligero ofrecen un nivel de seguridad de 128, el cual es el valor mínimo recomendado para garantizar la seguridad de la información en la actualidad (Barker y Dang, 2015).

3.4.3. Cifrado autenticado con datos asociados

El cifrado autenticado con datos asociados - AEAD proporciona un mecanismo de comprobar la autenticidad e integridad de la carga útil además de proteger su confidencialidad (McGrew, 2008). Esto se logra utilizando algunos datos asociados (AD), también conocidos como “datos autenticados adicionales” los cuales no son cifrados (McGrew, 2008). Por ejemplo, en el caso de protocolos de red, los datos asociados podrían ser puertos, direcciones de red, encabezados de paquetes, números de secuencia, etcétera, los cuales no deben de estar cifrados para el funcionamiento adecuado del protocolo.

El cifrado AEAD tiene dos operaciones principales: cifrado autenticado y descifrado autenticado, las cuales se describen a continuación.

3.4.3.1. Cifrado autenticado

Las entradas y salidas de la operación de cifrado autenticado se muestran en la Figura 19. En la operación de cifrado autenticado, el cifrador convierte el texto plano

(información a proteger) de entrada en una cadena bits aparentemente aleatorios. Esto evita que si algún atacante logra interceptar el mensaje, este no pueda interpretar la información contenida en la cadena bits garantizando así la confidencialidad del mismo.

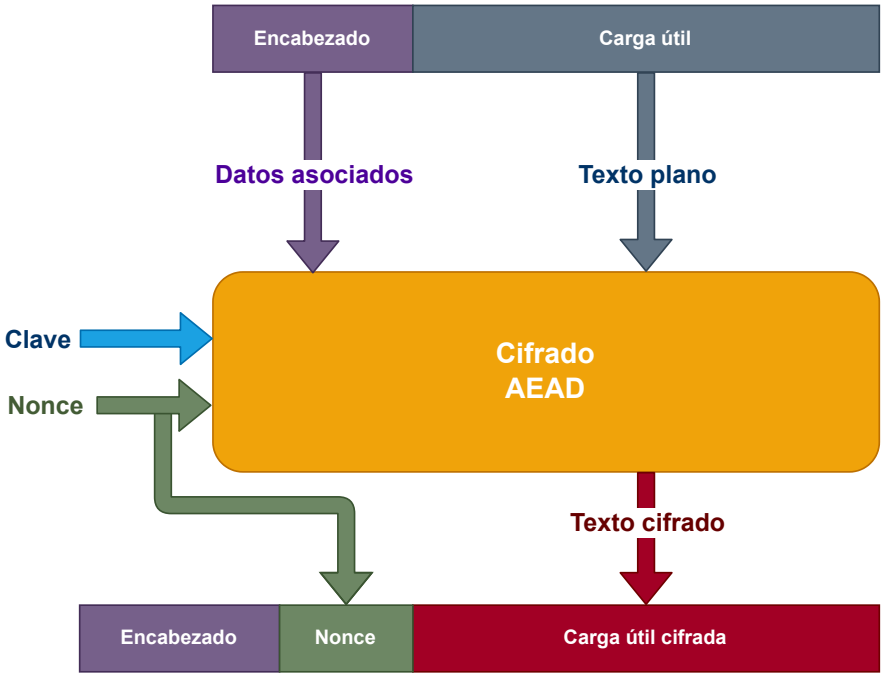


Figura 19. Entradas y salidas de la operación de cifrado de un esquema cifrado autenticado por datos asociados.

Además, la operación de cifrado autenticado protege la integridad del mensaje al utilizar parte de su encabezado,conocido en la literatura como “datos asociados”, en la operación de cifrado. Esto hace que si el texto plano cifrado o los datos asociados llegasen a cambiar, el resultado de la operación de descifrado fallaría, detectando así que la información fue comprometida. Cabe destacar que los datos asociados con los que se cifra la información no son cifrados y pueden estar conformados por números de secuencia, versiones de protocolo, direcciones de red, etc. En el caso de DTLS, los datos asociados se conforman por la cadena (Dierks y Rescorla, 2008):

```
datos asociados  = seq_num + TLSCompressed.type +
                           TLSCompressed.version + TLSCompressed.length;
```

donde el operador “+” significa concatenación.

Como se puede ver en la Figura 19, la operación de cifrado autenticado tiene 4

entradas:

- **Un texto plano P**, que contiene los datos que se van a cifrar y autenticar.
- **Los datos asociados A**, que contiene datos que se van a utilizar para autenticación. Estos datos no son cifrados.
- **Una clave secreta K**, que se va a utilizar para cifrar el texto plano (información). La clave secreta se debe generar de manera aleatoria o con algoritmos de generación pseudoaleatorios.
- **Un nonce N**, que es un número arbitrario que se debe utilizar solo una vez. El nonce se genera cada vez que se invoca la operación de cifrado autenticado y debe de ser distinto e independiente del valor particular de la clave.

La salida del bloque cifrado es el **“texto cifrado C”**, que es una cadena de bits con longitud igual o mayor al texto sin formato. Si la operación no tiene éxito, el cifrador debe retornar una indicación de que la operación no pudo ser realizada.

3.4.3.2. Descifrado autenticado

La operación de descifrado como se muestra en la Figura 20 invierte a la operación de cifrado. Además de recuperar la información original, esta operación detecta si la información o los datos asociados fueron modificados, garantizando su integridad.

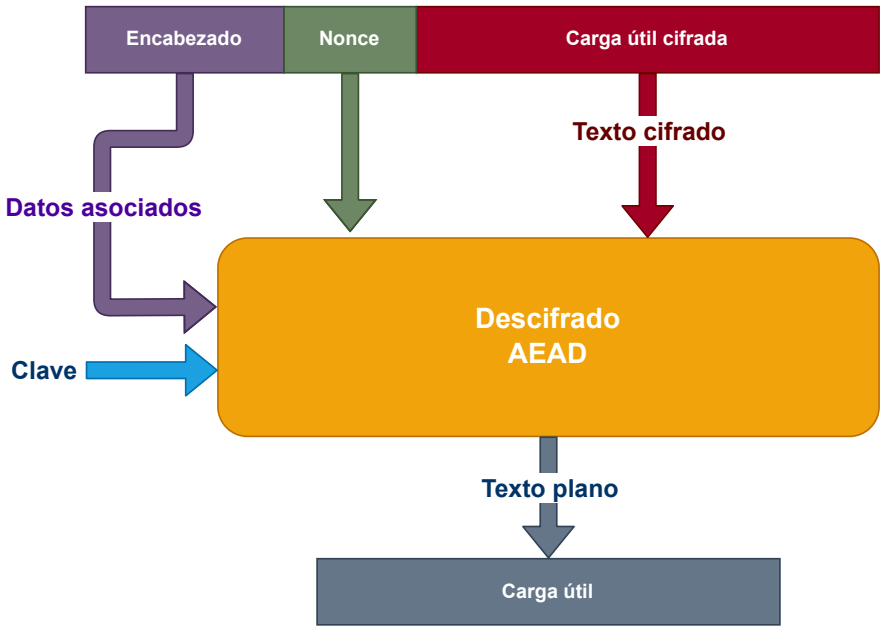


Figura 20. Entradas y salidas de la operación de descifrado de un esquema cifrado autenticado por datos asociados.

La operación de descifrado autenticado requiere las siguientes entradas :

- **La clave secreta K**, utilizada en la operación de cifrado.
- **El nonce N**, que se utilizó en la operación de cifrado (los cuales están contenidos en el encabezado).
- **Los datos asociados A**, que se utilizaron en la operación de cifrado (los cuales están contenidos en el encabezado).
- **El texto cifrado C**, contenido en el paquete generado en la operación de cifrado.

Con estas entradas se produce una única salida que es un **texto plano P**, el cual es exactamente el mismo que se introdujo en la operación de cifrado. Si el descifrado no fue llevado con éxito se debe de devolver una indicación de operación no realizada.

3.5. Concurso de cifradores ligeros del NIST

El instituto nacional de estándares y tecnología de los Estados Unidos (NIST por sus siglas en ingles) es un organismo fundado en 1901, cuya misión es ampliar y promover

la innovación y competencia industrial mediante mediciones científicas, estandarización y tecnología (NIST, 2009). Sus estándares son respetados e implementados en todo el mundo, un ejemplo de esto es el estándar actual de cifrado por bloques que se utiliza en los protocolos seguros de Internet llamado AES (Estándar de Cifrado Avanzado)(FIPS, 2001).

Como se mencionó anteriormente, la falta de estandarización en el ámbito de la seguridad en la IoT es uno de los retos más importantes que se tiene en la actualidad para estas tecnologías (Bansal y Kumar, 2020). Para enfrentar este reto en 2015 el NIST inició un concurso para estandarizar un algoritmo de cifrado ligero para utilizarse en las implementaciones en donde los recursos computacionales sean limitados. Este concurso consta de tres rondas de las cuales ya se realizaron 2. Inicialmente, 57 algoritmos de cifrado ligero fueron sometidos para ser evaluados en cuanto a seguridad y rendimiento para obtener el algoritmo más apto.

Para ser admitidos en el concurso, los equipos desarrolladores de los algoritmos debieron entregar una versión de referencia bajo los siguientes requerimientos de documentación (NIST, 2018):

- Especificación completa escrita de los algoritmos que incluyera todas las operaciones matemáticas, ecuaciones, tablas, y diagramas que se necesitan para implementar el algoritmo.
- El documento también debía contener un diseño racional del cifrador y una explicación para todas las decisiones importantes de diseño al ser construido.
- Para los algoritmos que tienen parámetros personalizables, el documento debía especificar concretamente los valores para estos parámetros. Además se debía describir los parámetros más importantes y una selección de rangos posibles entre seguridad y rendimiento.
- Se requería describir las ventajas y limitaciones del algoritmo en términos de seguridad, rendimiento y costos de implementación.
- Adicionalmente, se solicitó proveer una implementación de referencia para promover el entendimiento del algoritmo propuesto. El código fuente debía estar

acompañado de un set de vectores de prueba los cuales serían generados por el algoritmo.

Además de los requerimientos de documentación, para ser un candidato válido era necesario cumplir con los siguientes requisitos de implementación.

- Ser compatible con el esquema de cifrado autenticado de datos asociado (AEAD) que se explicó anteriormente.
- La función de cifrado debe de tener las siguientes 4 entradas:
  - **Un texto plano P**, que contiene los datos que se van a cifrar y autenticar.
  - **Los datos asociados A**, que contienen datos que se van a utilizar para autenticación. Estos datos no son cifrados.
  - **Un nonce N**, que es un número arbitrario que se debe utilizar solo una vez. El nonce se genera cada vez que se invoca la operación de cifrado autenticado y debe de ser distinto e independiente del valor particular de la clave.
  - **Una clave secreta K**, que se va a utilizar para cifrar el texto plano (información). La clave secreta se debe generar de manera aleatoria o con algoritmos de generación pseudoaleatorios.
- La función de cifrado debe de retornar el texto cifrado con estas 4 entradas.
- Los participantes podían enviar una familia de hasta 10 algoritmos AEAD, donde los miembros de la familia varían en parámetros externos, como por ejemplo el largo de la clave o el largo del nonce.
- Los algoritmos no deben de especificar un largo de clave menor a 128 bits.
- Los algoritmos deben de admitir todas las cadenas de bytes posibles como entradas siempre y cuando cumplan con el largo requerido.

Cuando se comenzó este trabajo de tesis, el concurso se encontraba en su segunda etapa en donde solo quedaba 32 candidatos de los 57 originales. Basándose en (Calik *et al.*, 2020), en este punto se escogieron 5 candidatos para poder realizar la evaluación de este trabajo, considerando los enfoques por bloque, flujo y permutación. Estos

algoritmos se describen en la siguiente sección. Cabe mencionar que durante el transcurso del desarrollo de este trabajo se dieron a conocer los 10 finalistas del concurso, en donde los 5 candidatos que se decidieron evaluar en primera instancia fueron también escogidos como finalistas.

3.6. Algoritmos de cifrado seleccionados

Como se mencionó en la sección anterior, para seleccionar los algoritmos evaluados en este trabajo de tesis se revisó la referencia (Calik *et al.*, 2020), en donde se realizó una evaluación comparativa del rendimiento general en microcontroladores de los 32 candidatos de la segunda ronda del concurso del NIST. A partir de esta revisión se obtuvieron las siguientes métricas relevantes:

- Tamaño del código final en la implementación: es el espacio en memoria que ocupa el código del algoritmo después de haber sido compilado para cada arquitectura.
- Velocidad de cifrado en ciclos por byte: Se refiere a cuántos ciclos de procesador son necesarios para realizar una operación de cifrado para diferentes cantidades de bytes.

Considerando estas métricas e información adicional proporcionada en (Calik *et al.*, 2020) se escogieron los 5 cifradores descritos a continuación.

3.6.1. GIFT-COFB

GIFT-COFB es un diseño AEAD basado en cifrado de bloques que utiliza GIFT-128 como cifrado de bloques subyacente. Utiliza una clave de cifrado ***K*** de 128 bits, un nonce ***N*** de 128 bits, unos datos asociados ***A*** de longitud arbitraria y un texto plano ***P*** de longitud arbitraria como entradas, devolviendo un texto cifrado C de la misma longitud que ***P*** y una etiqueta ***T*** de 128 bits anexado a éste (Isical, 2019).

GIFT-COFB puede verse como una integración eficiente del modo COFB y el cifrado de bloque GIFT-128. GIFT-128 mantiene un estado de 128 bits y una clave de 128 bits

(Isical, 2019), por lo que ofrece un nivel de seguridad de 128. GIFT es una familia de cifrados de bloque parametrizados por el tamaño del estado y el tamaño de la clave. Todos los miembros de esta familia son ligeros y se pueden implementar de manera eficiente en aplicaciones ligeras (Isical, 2019).

Por otro lado, el modo COFB calcula la “retroalimentación combinada” (de la salida de cifrado de bloque y el bloque de datos) para elevar el nivel de seguridad. En realidad, esto ayuda a diseñar un esquema con un tamaño de estado bajo y a tener una implementación de estado bajo (Isical, 2019).

3.6.2. Xoodyak

Xoodyak es un esquema de cifrado ligero y versátil adecuado para entornos restringidos. Se puede utilizar para hash, cifrado, cálculo MAC y cifrado autenticado (Daemen *et al.*, 2019).

Internamente, Xoodyak usa el algoritmo de permutación Xodoo. El enfoque de diseño de esta permutación de 384 bits está inspirado en Keccak-p, mientras que está dimensionado como Gimli para la eficiencia en procesadores de gama baja. La estructura consta de tres planos de 128 bits cada uno, que interactúan por columnas de 3 bits a través de operaciones de mezcla y no lineales, que por lo demás se mueven como tres objetos rígidos independientes (ver Figura 21). Su función redonda se adapta muy bien a procesadores de 32 bits de gama baja, así como a hardware dedicado compacto (Daemen *et al.*, 2019).

El modo de operación de alto nivel de Xodoo se llama Cyclist. Una de sus características principales es que no se limita al cifrado autenticado, si no que ofrece servicios detallados estilo Strobe, y admite HASH (Daemen *et al.*, 2019).

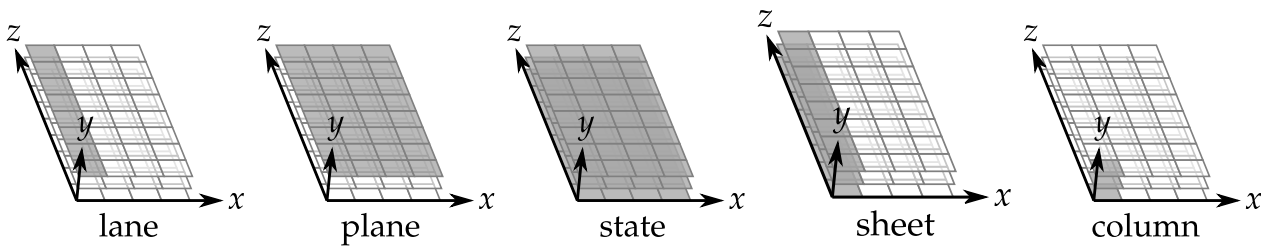


Figura 21. Diagrama simplificado de un estado de XOODOO.

Xoodyak cuenta con las siguientes características (Daemen *et al.*, 2019):

- Es compacto: solo requiere un estado de 48 bytes, un puntero de entrada y uno de salida. La construcción doble subyacente permite que los bytes lleguen inmediatamente integrándose en el estado sin la necesidad de mensajes en cola.
- Prevé protecciones de canal lateral:
  - Ofrece resistencia a las fugas. Durante una sesión, la clave secreta es un objetivo en movimiento, ya que no hay una clave fija. Entre sesiones, prevé un mecanismo para rotar claves.
  - La misma clave puede ser usada varias veces, se puede agregar fácilmente protección contra los ataques de implementación. La función de redondeo de grado 2 de Xoodoo hace que los esquemas de enmascaramiento y umbral sean de costo computacional relativamente bajo.
- La especificación es pequeña y simple, mientras soporta todas las operaciones de cifrado simétrico con un nivel de seguridad de 128.
- Este modo ofrece gran flexibilidad y puede ser adaptado para necesidades específicas de una aplicación. Por ejemplo, admite sesiones y etiquetas intermedias en cifrado autenticado de forma transparente. Las etiquetas intermedias permiten reducir el búfer en el extremo receptor para almacenar el texto sin formato antes de verificar la etiqueta.
- Está basado en un algoritmo de permutación fuerte y eficiente:
  - Xoodoo se basa en los mismos principios que Keccak-p, por lo tanto, se comprenden bien sus propiedades de propagación.
- En caso de uso indebido (es decir, un nonce modificado o liberación de textos cifrados descifrados no verificados), la clave no se puede recuperar mediante criptoanálisis. La autenticación no depende de un nonce.
- Como limitación Xoodyak es inherentemente serial a nivel de construcción.
- Otra limitación es que realiza cifrado de flujo, por lo que la reutilización accidental de nonce puede resultar en una fuga de hasta 24 bytes de texto sin formato.

3.6.3. ASCON

ASCON es una familia de cifradores de autenticación y hashing diseñados para ser ligeros y de fácil implementación, incluso con contramedidas adicionales contra ataques de canal lateral. Sus principales características son (Dobraunig *et al.*, 2019):

- Cifrado y hash autenticados (longitud de salida fija o variable) con una única permutación ligera.
- Modos de operación basados en esponjas con una permutación de SPN personalizada.
- Modo comprobable seguro con finalización con clave para mayor robustez.
- Fácil de implementar en software y hardware.
- Ligero para dispositivos restringidos: estado pequeño, permutación simple, modo robusto.
- Rápido en hardware.
- Rápido en software: S-box de 5 bits dividido en bits y canalizable para arquitecturas de 64 bits.
- Escalable para una seguridad más conservadora o un mayor rendimiento.
- Resistencia al tiempo - sin consultas ni adiciones en la tabla.
- Resistencia de canal lateral - caja S optimizada para contramedidas.
- Para el concurso del NIST ofrece tres variantes diferentes.
- Sobrecarga mínima (longitud del texto cifrado = longitud del texto sin formato).
- Pase único, en línea (cifrado y descifrado), basado en nonce, sin inversión.
- Para el concurso del NIST ASCON ofrece un nivel de seguridad de 128.

3.6.4. Grain-128

Grain-128AEAD es un miembro de la familia de cifradores de flujo Grain. Este soporta cifrado autenticado con datos asociados. La familia Grain de cifrados de flujo se ha analizado exhaustivamente desde su introducción en el proceso eSTREAM, donde la variante de clave de 80 bits Grain v1, junto con MICKEY 2.0 y Trivium, se seleccionó en la cartera final de algoritmos (categoría de hardware). Desde entonces, también se han propuesto Grain-128 y Grain-128a, ambos con clave de 128 bits por lo que ofrece un nivel de seguridad de 128 y el último con autenticación de mensaje opcional. Grain-128 se considera roto por los ataques de cubo dinámico, y se ha demostrado que para Grain-128a sin autenticación, también hay ataques más eficientes que la fuerza bruta (Hell *et al.*, 2019).

En comparación con las variantes anteriores, Grain-128AEAD modifica la inicialización del cifrado de modo que la clave se vuelve a introducir al final de la inicialización. El propósito de esta reintroducción de la clave es no permitir que la clave secreta se reconstruya inmediatamente en caso de que se conozcan los estados de LFSR y NFSR. Esta es una característica inspirada en el cifrado de flujo Lizard (Hell *et al.*, 2019).

3.6.5. TinyJambu

Jambu es un modo de cifrado autenticado con cifrado de bloque pequeño. TinyJambu es una pequeña variante del modo Jambu. El modo TinyJambu se basa en una permutación con clave. El tamaño del estado de TinyJambu es solo dos tercios del de Jambu, el tamaño del bloque de mensajes de TinyJambu es la mitad del del modo Jambu. Cuando se reutiliza nonce, TinyJambu proporciona una mejor seguridad de autenticación que el modo Jambu. La seguridad de autenticación del modo TinyJambu es mejor que el modo Dúplex cuando se reutiliza nonce (para el mismo tamaño de permutación y tamaño de bloque de mensaje) (Huang y Tao, 2019).

La permutación se basa en un registro de desplazamiento de retroalimentación no lineal de 128 bits. Esta permutación ligera se utiliza en el modo TinyJambu. La permutación con clave admite tres tamaños de clave posibles: 128 bits, 192 bits, 256 bits (Huang y Tao, 2019).

Para las aplicaciones en las que una clave fija está incrustada en los dispositivos, y para las aplicaciones en las que se almacena una clave secreta en un dispositivo para proteger múltiples mensajes, TinyJambu usa solo un registro de 128 bits para el cifrado autenticado (Huang y Tao, 2019).

3.6.6. Algoritmo de cifrado por defecto

Además de los algoritmos de cifrado ligero que se mencionaron anteriormente, también se va a utilizar el algoritmo de cifrado AES_128_CCM_8. Este es uno de los algoritmos de cifrado obligatorios para el estándar DTLS (McGrew y Bailey, 2012), y viene como algoritmo de cifrado por defecto en la implementación de TinyDTLS, que es la que se utilizó en este trabajo de investigación.

AES_128_CCM_8 es una versión del algoritmo de cifrado estándar AES que tiene un nivel de seguridad de 128 lo que significa que su clave es de 128 bits de longitud. Además de esto, también agrega autenticación mediante la técnica de contador con código de autenticación de mensajes de encadenamiento de bloques de cifrado (CCM por sus siglas en inglés) (Dworkin, 2007).

3.7. Sumario

En la Tabla 3 se muestran a manera de resumen las características más importantes para este trabajo de tesis de los algoritmos de cifrado que se evalúan en los siguientes capítulos. Como se puede observar los algoritmos elegidos están basados en diversos tipos de primitivos por lo que se espera analizar también si esto tiene alguna afectación en las métricas tomadas en las pruebas, además de esto también se destaca que la mayoría de los algoritmos ofrecen un nivel de seguridad de 128 a excepción del algoritmos de seguridad TinyJambu el cual tiene una variante de 160 y 256 las cuales también se implementarán para ver si existen alguna variacion en los resultados de las diferentes pruebas que se van a realizar.

Tabla 3. Caracteristicas destacadas de los algoritmos de cifrado.

Nombre	TIPO	AEAD support	Tamaño de clave en bits	Tamaño de nonce
ASCON	Basado en permutacion	x	128	128
Xoodyak	Basado en permutacion	x	128	128
GIFT-COFB	Basado en bloques	x	128	128
TinyJambu	Basado en bloques	x	128,160,256	96
Grain-128	Basado en flujo	x	128	96
AES-128	Basado en bloques	x	128	64

Capítulo 4. Cama de pruebas

4.1. Introducción

Para poder realizar la evaluación de los cifradores considerados en el presente trabajo de investigación, se decidió realizar una cama de pruebas física que permitiera realizar una evaluación de desempeño en condiciones cercanas a un despliegue realista de una red IoMT. Las características de dicha cama de pruebas son descritas en este capítulo.

A continuación, se describe:

- La pila de tecnologías utilizadas para habilitar la cama de pruebas físicas y las razones por las que estas tecnologías fueron seleccionadas.
- La metodología que se utilizó para implementar la pila de tecnologías en las diferentes plataformas utilizadas en la cama de pruebas.
- La implementación de los cifradores evaluados en este trabajo de investigación.

4.2. Escenario general de aplicación propuesto para el desarrollo de la cama de pruebas

Para la realización de la cama de pruebas se consideró un escenario IoMT en donde se transmitirán 3 señales generadas por 3 nodos sensores colocados en pacientes para su monitorización remota por parte de especialistas. Las señales consideradas son oximetría, temperatura y tensión arterial. Los tres sensores forman una red de área personal hacia un router de borde para conectividad hacia la Internet u otra red local, tal cual se muestra en la Figura 22.

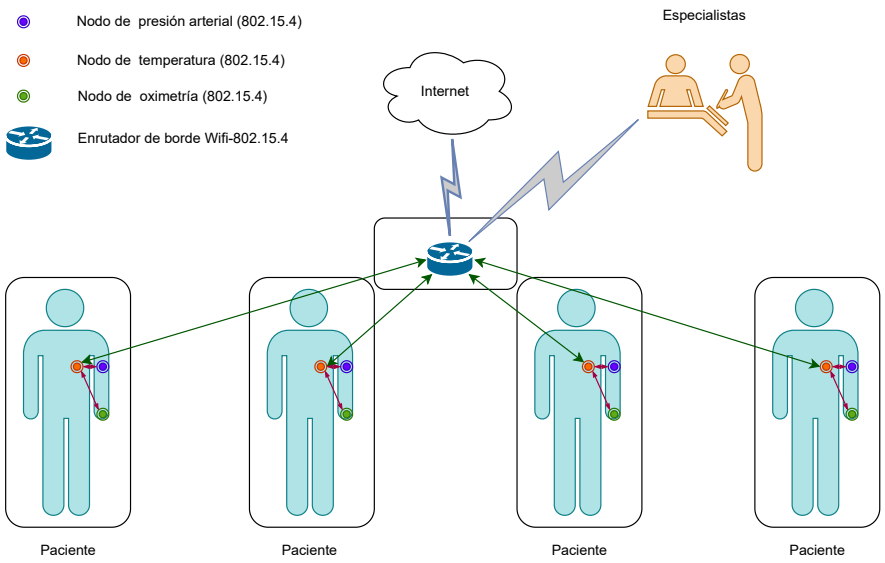


Figura 22. Ejemplo de una red WSN para IoMT.

Cada nodo sensor genera una tasa de datos diferente acorde al tipo de variable que se está monitorizando. En particular, del análisis de la literatura se utilizó la tasa de datos propuesta en (Casillas, 2012) que se muestra en la Tabla 4, la cual presenta para un escenario como el planteado, la generación de datos típica de cada sensor es:

- **Baumanómetro.** Genera 64 bits (8 bytes) por muestra. Se considera que una tasa de muestreo de cada 10 minutos es adecuada para el escenario planteado.
- **Oximetro.** Se usará una tasa de muestreo de 60 muestras por segundo. Se utilizará una resolución de 16 bits por muestra por 2 canales (32 bits) lo que es equivalente a 4 bytes en total.
- **Termómetro.** Genera 16 bit (2 bytes) por muestra. Se considerará que una tasa de muestreo de 1 muestra cada minuto es adecuada para el escenario planteado.

Tabla 4. Perfil de tráfico de las señales utilizadas.

Nodo sensor	Tamaño de muestra	Tasa de muestreo	Tasa de bits por segundo
Oximetro	32 bits (4 Bytes)	60 muestras por segundo	1920 bps
Baumanometro	64 bits (8 Bytes)	1 muestra cada 10 minutos	0.1066 bps
Termómetro	16 bits (2 Bytes)	1 muestra por minuto	0.2666 bps

4.3. Pila de tecnologías consideradas para la implementación de la cama de pruebas

Como se explicó en Capítulo 2, para la realización de este trabajo de tesis se eligió la pila de tecnologías 6LoWPAN que se muestra en la Figura 23, bajo las siguientes consideraciones:

- Capa física y de acceso al medio (MAC) basadas en el estándar IEEE 802.15.4 trabajando en la frecuencia de 2.4 GHz.
- Capa de adaptación 6LoWPAN para la compatibilidad con las capas de red.
- Capa de red que utiliza los protocolos IPv6 y RPL (“Routing Protocol for Low-Power and Lossy Networks”).
- Utilización de UDP en la capa de transporte.
- Utilización del protocolo DTLS en la capa de seguridad.
- Implementación del estándar CoAP en la capa de aplicación (CoAP es una capa de aplicación para dispositivos limitados).

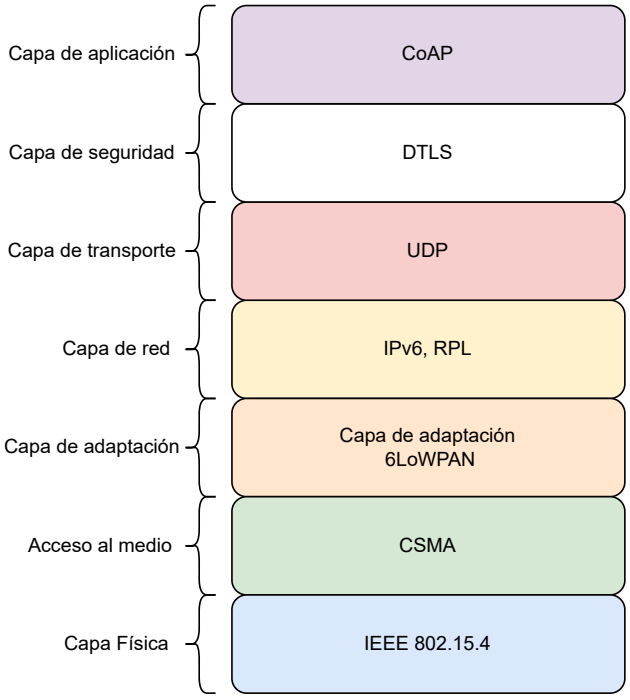


Figura 23. Pila de tecnologías utilizadas.

Considerando el escenario de evaluación propuesto y la pila de tecnologías 6LoWPAN considerada, se determinó que para la implementación de la cama de pruebas sería necesario abordar los siguientes puntos:

- Una capa de aplicación que permitiera la implementación de los servicios propuestos en el escenario mencionado con las características adecuadas.
- Una implementación del protocolo DTLS que permitiera la integración de los diferentes cifradores considerados para su evaluación.
- Un sistema operativo IoT que permitiera la implementación de las soluciones de la capa de aplicación y DTLS sobre una red basada en el estándar 6LoWPAN.
- Una plataforma de desarrollo con disponibilidad y previamente validada que incluya un radio y un microcontrolador. El microcontrolador debe:
  - Permitir cargar y correr el sistema desarrollado para la realización de las pruebas.
  - Ser capaz de soportar la pila de tecnologías considerada.
  - Ser de código abierto para poder modificarlo sin afectar su licencia.
  - Ser relevante en la literatura.
- Una plataforma de simulación capaz de soportar la ejecución del sistema operativo completo en un entorno virtual realista.

Con lo anterior en mente, se decidió usar como base la plataforma física Sensortag CC2650 de Texas Instruments. Está plataforma se encuentra disponible en el Laboratorio de Redes Inalámbricas y Sistemas Embebidos del Grupo de Investigación Avanzada en Redes de Telecomunicaciones y Sistemas. La plataforma cumple con las características mencionadas previamente y soporta diversos sistemas operativos orientados al despliegue de redes de sensores. En este sentido, se decidió utilizar el sistema operativo para dispositivos IoT Contiki-NG ya que (al igual que la plataforma Sensortag CC2650) cubre todos los requerimientos antes mencionados para la realización de la cama de pruebas. Estas plataformas se describen a continuación, en conjunto con las implementaciones para DTLS y CoAP que se implementaron sobre ellas para realizar la evaluación de los cifradores ligeros.

4.3.1. Plataforma física Sensortag CC2650

Para la implementación en un entorno real de las diferentes pruebas que se realizaron se seleccionó la tarjeta de evaluación de Texas Instrument Sensortag CC2650 (TI.com, 2016) que se muestra en la Figura 24. Esta selección se basó, además de que cumple con los requisitos descritos en la introducción de este capítulo, en el hecho de que existen varios trabajos de investigación que han utilizado estos dispositivos, y en la experiencia con el uso de dicha plataforma dentro del grupo de investigación en el cual se realizó la presente tesis (Cárdenas, 2019),(Carrillo, 2017),(Mischie, 2017).

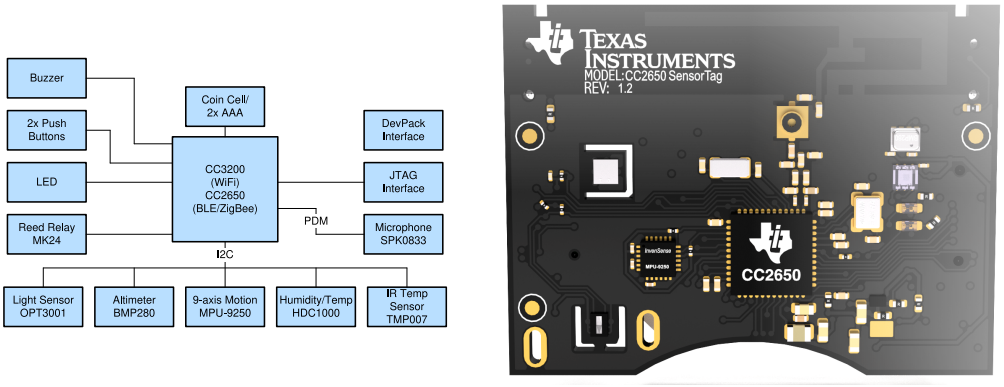


Figura 24. Plataforma física utilizada, Sensortag CC2650.

La plataform Sensortag CC2650 se destaca por las siguientes características:

- Cuenta con el microprocesador ARM Cortex-M3 CC2650 de ultra bajo consumo.
- Cuenta con 128 KB de memoria ROM y 20 KB de memoria RAM para el almacenamiento y ejecución de aplicaciones.
- Cuenta con 10 sensores de bajo consumo incluidos: un termómetro, un sensor de luz, un sensor de humedad, acelerómetros, giróscopos y magnetómetros, entre otros.
- Tiene incorporado un radio compatible con redes IEEE 802.15.4 y BLE.
- Compatible 100 % con redes 6LoWPAN y Contiki-NG.

Para el caso de la implementación de 6LoWPAN sobre la plataforma Sensortag CC2650, es necesario habilitar el modo de funcionamiento para redes IEEE 802.15.4

en la capa MAC y en la capa física. En particular se estará trabajando en la banda de 2.4 GHz en la capa física.

A continuación se hará una descripción breve de las principales características del sistema operativo seleccionado para la implementación de la cama de prueba y el raciocinio detrás de su selección.

4.3.2. Sistema operativo - Contiki-NG

Considerando la plataforma Sensortag CC2650 utilizada en este trabajo y la pila de protocolos 6LoWPAN que debe de correr sobre la plataforma, se buscó un sistema operativo IoT que cumpliera con las siguiente características.

1. Que soportara la implementación de la pila de protocolos mostrada en la Figura 23, la cual incluye CoAP, una implementación del protocolo de seguridad DTLS y la pila de 6LowPAN.
2. Que fuese de código abierto para poder ser modificado.
3. Que fuese relevante en la literatura.
4. Que fuese compatible con la plataforma Sensortag 2650.
5. Que permita habilitar el modo de funcionamiento IEEE 802.15.4 en la plataforma Sensortag CC2650.
6. Que permita hacer ajustes en la capa MAC.

Se realizó una evaluación de los sistemas operativos IoT más relevantes en la literatura actualmente, basados en (Bansal y Kumar, 2020), (Zikria *et al.*, 2019) y (Gaur y Tahiliani, 2015) llegando a la siguiente tabla comparativa:

Tabla 5. Comparativa de los sistemas operativos considerados.

Sistema operativo	Caracteristicas					
	1	2	3	4	5	6
Riot OS	parcial	X	X	parcial	parcial	X
Contiki-NG	X	X	X	X	X	X
Zephir	parcial	X	X	parcial	norte	X
Lite OS	norte	X	X	norte	parcial	parcial

Como se puede observar el sistema operativo Contiki-NG es el que mejor cumple con las características requeridas, por lo que este fue seleccionado para la implementación.

Contiki-NG es un sistema operativo IoT multiplataforma y de código abierto, de arquitectura modular con características parciales de tiempo real. Está centrado en la implementación de protocolos estándar y de comunicación de bajo consumo (seguros y fiables), como IPv6 / 6LoWPAN, RPL y CoAP. Además, cuenta con una amplia documentación y una comunidad activa (Contiki-ng, 2018).

4.3.3. TinyDTLS en Contiki-NG

Para poder implementar los cifradores ligeros se requería de una implementación del protocolo de seguridad DTLS que fuera compatible con el sistema operativo Contiki-NG y en donde fuese posible la modificación del método de cifrado por defecto. La única opción que se encontró que ofreciera dicho potencial, fue la implementación TinyDTLS desarrollada por la fundación Eclipse (Eclipse, 2016) ya que es compatible con Contiki-NG y el código fuente está abierto.

Esta implementación funciona de manera modular junto al sistema operativo Contiki-NG como se muestra en la Figura 25. Cuando se requiere establecer una comunicación cifrada, Contiki-NG pasa los paquetes (datos) generados por la capa de aplicación CoAP al módulo de TinyDTLS para que éste le aplique los mecanismos de seguridad revisados en Capítulo 3. Así, el módulo de TinyDTLS genera un paquete con los datos CoAP cifrados para posteriormente enviarlo a la capa de transporte UDP de Contiki-NG y continuar con el proceso de transmisión.

En el sentido inverso, la capa de transporte de Contiki-NG le entregará al módulo de TinyDTLS los paquetes cifrados recibidos para que los descifre y autentique. Una vez realizado esto, TinyDTLS le entrega los datos desencriptados (texto plano) a la capa CoAP para que los procese.

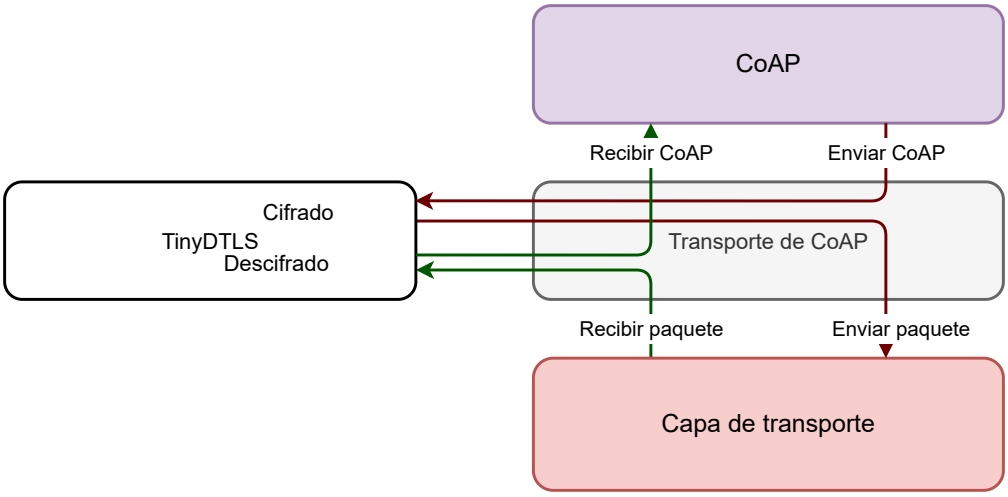


Figura 25. Implementación de TinyDTLS en Contiki-NG.

Es importante destacar que, previo al intercambio de datos cifrados (enlace seguro), el módulo de TinyDTLS se encarga de ejecutar el protocolo de handshake de DTLS que se explicó en el Capítulo 3. Además, el módulo de TinyDTLS también gestionará la sesión de cada uno de los nodos con los que se establece un enlace seguro. A manera de resumen el módulo de TinyDTLS agrega las siguientes funciones al sistema operativo Contiki-NG:

- Iniciar una sesión segura hacia un nodo determinado.
- Cerrar una sesión segura con un nodo determinado.
- Enviar un mensaje seguro hacia un nodo con una sesión segura iniciada.
- Recibir un mensaje seguro desde un nodo con una sesión segura iniciada.

4.3.4. Capa de aplicación - CoAP

Como se explicó en el Capítulo 2, una capa de aplicación se va a encargar de funcionar como plataforma para poder implementar diferentes servicios en el dispositivo. Estos servicios pueden ser consultados por otros dispositivos con diferentes características siempre y cuando soporten la misma capa de aplicación.

El protocolo de aplicación restringida(CoAP), es un protocolo de aplicación especializado para dispositivos restringidos. Fue seleccionado para la cama de pruebas debido

a su amplia utilización; a que es 100 % compatible con la capa de seguridad TinyDTLS; y a que tiene una implementación completa en Contiki-NG.

4.3.4.1. Modelo de mensajes de CoAP

CoAP se basa en la comunicación entre nodos finales y se caracteriza entre otras cosas por contar con dos tipos de mensajes: confirmables y no confirmables (Shelby *et al.*, 2014). En un mensaje confirmable (ver Figura 26) el transmisor envía un mensaje y el receptor al procesar este mensaje debe de responder con un ACK de confirmación. Si el transmisor no recibe el ACK en un tiempo predeterminado, entonces mandará una retransmisión. El tiempo entre retransmisiones se reducirá exponencialmente hasta recibir el ACK o hasta que se alcance un número máximo de retransmisiones. Tanto el número máximo de retransmisiones como el tiempo de respuesta se configuran en cada implementación (Shelby *et al.*, 2014).

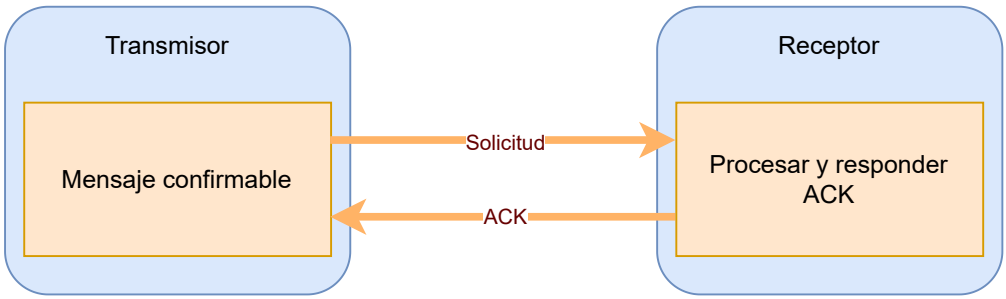


Figura 26. Mensaje confirmable de CoAP.

En un mensaje no confirmable (ver Figura 27) el receptor del mensaje no responde con un ACK al procesar el mensaje. Este tipo de mensajes es adecuado para aplicaciones tipo “streaming”, donde se busca recibir los paquetes dentro de cierto marco de tiempo y se puede tolerar cierta pérdida de paquetes a cambio de reducir la sobrecarga en la red causada por las retransmisiones (Shelby *et al.*, 2014).

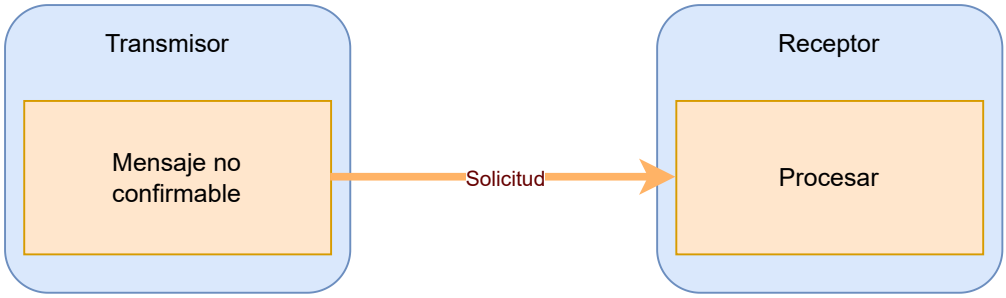


Figura 27. Mensaje no confirmable de CoAP.

4.3.4.2. Modelo solicitud/respuesta de CoAP

CoAP provee un modelo de interacción de solicitud/respuesta para comunicación entre nodos. Este modo de interacción es similar al intercambio de datos en una arquitectura cliente-servidor que utiliza el protocolo HTTP. Similar a HTTP, CoAP hace uso de métodos GET, PUT, POST y DELETE, con los cuales un “cliente” (nodo) CoAP manda una o más peticiones a un “servidor” (el nodo con el cual el cliente se quiere comunicar) CoAP. El servidor le da servicio a las peticiones mandando mensajes CoAP. De manera resumida el cliente CoAP utiliza:

- GET para solicitar información.
- POST para solicitar que se procese la información que se incluye en la solicitud.
- PUT para solicitar que el recurso que se indique en la solicitud sea actualizado o reescrito con la información que se incluye en la solicitud.
- DELETE para solicitar la eliminación del recurso dado en la solicitud (Shelby *et al.*, 2014).

A pesar de la similitud entre la utilización de los métodos de HTTP y CoAP, vale la pena mencionar que la sintaxis de dichos métodos es diferente entre protocolos.

Las solicitudes pueden enviarse tanto en mensajes confirmables, como no confirmables. Si el cliente lo manda en un mensaje confirmable y el servidor tiene la respuesta de una solicitud inmediatamente, este responderá con un ACK que tiene incluida la respuesta a esta solicitud. Si el servidor requiere más tiempo para obtener la

respuesta, este responderá con un ACK vacío y mandará la respuesta en un mensaje confirmable separado como se muestra en la Figura 28 (Shelby *et al.*, 2014).

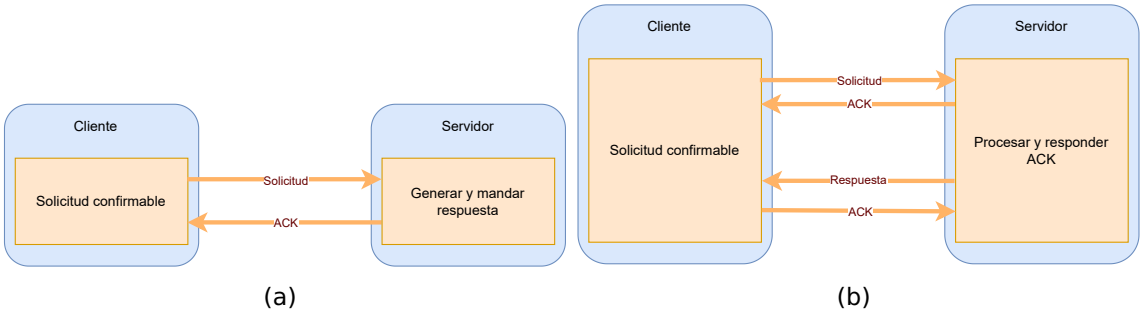


Figura 28. (a) Solicitud con respuesta en ACK b) Solicitud con respuesta separada.

4.3.4.3. Modelo observador de CoAP

Además del modelo solicitud/respuesta, en CoAP se especificó un modelo observador. Este modelo fue pensado para suplir las limitaciones del modelo solicitud/respuesta en escenarios en donde el cliente está interesado en el estado actual de un recurso en específico constantemente (Hartke, 2015).

Un ejemplo de esto es un servidor que ofrece el servicio de temperatura en donde el cliente desea saber de este servicio cada segundo. Con el método de solicitud/respuesta el cliente necesitaría mandar una solicitud cada segundo, lo cual representaría un coste extra de procesamiento en ambos nodos. En cambio con el modelo de observador un nodo cliente sólo se tiene que “registrar” a un servicio en específico, entonces el servidor le “notificará” al cliente ya sea a intervalos regulares o si se da un cambio con la actualización de la información. Esto reduciendo drásticamente el número de transmisiones como se muestra en la Figura 29 (Hartke, 2015).

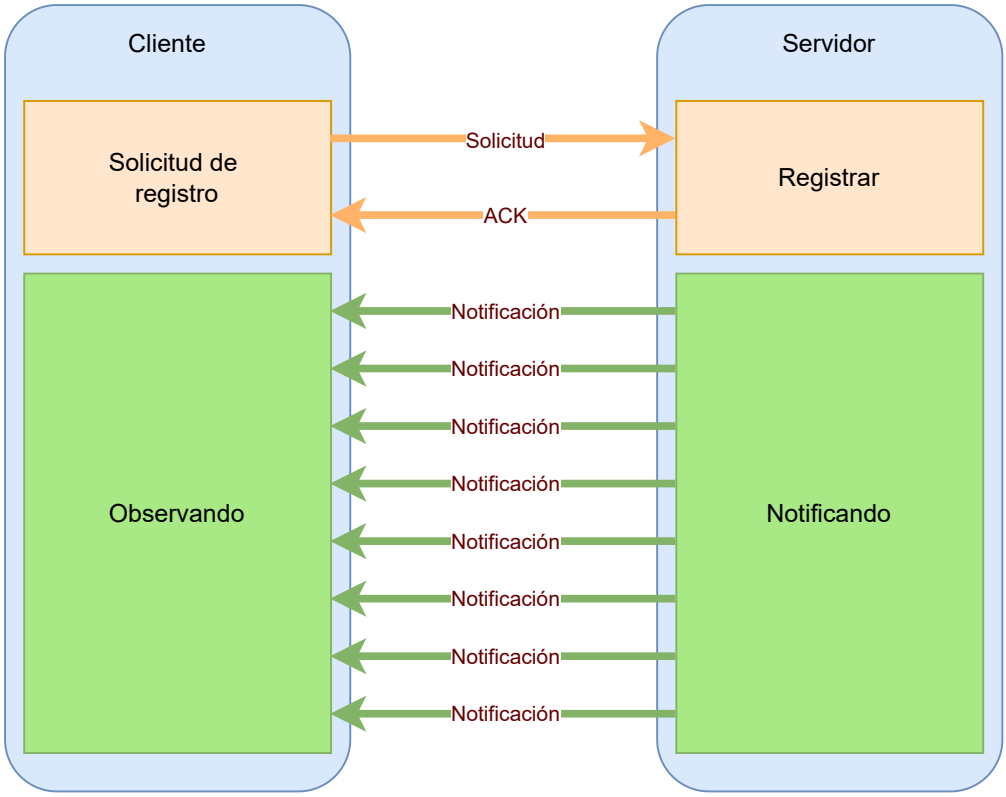


Figura 29. Modelo observador de CoAP.

4.4. Plataformas de simulación

Para validar los resultados obtenidos con la cama de pruebas físicas, se consideró utilizar una plataforma de simulación que cumpliese con las siguientes características:

- Que fuera compatible total o parcialmente con las tecnologías que se van a utilizar.
- Que fuera lo más parecido posible a una implementación en el mundo real.

Se encontraron dos candidatos posibles que cumplen con estas características: el simulador incorporado en el sistema operativo Contiki-NG llamado “Cooja”; y un nuevo simulador desarrollado por la empresa Antmicro llamado “Renode”. A continuación se provee una breve descripción de ambos.

4.4.1. Simulador Cooja

Cooja es un simulador de red enfocado en redes de sensores inalámbricos y es parte del proyecto Contiki-NG. Está escrito en el lenguaje Java y cuenta con las siguientes características:

- Se basa en la simulación de dispositivos reales en donde se puede ejecutar el sistema operativo Contiki-NG. Sin embargo, ya no está en desarrollo, por lo que actualmente solo es compatible con “Cooja mote”, el cual es un dispositivo virtual especialmente diseñado para el simulador.
- Posee una interfaz gráfica amigable para el usuario facilitando la configuración de la red y los diferentes dispositivos que interactúan en ella.
- Simulación del canal radio con capacidad de configurar pérdidas por distancia y colisiones.
- Capaz de entregar métricas detalladas sobre el uso del canal radio y el consumo de energía de los dispositivos en la simulación.
- Debido a su arquitectura, el comportamiento interno de los dispositivos simulados puede no ser exacto en comparación con el mundo real.

Cabe destacar que actualmente Cooja no se encuentra más en desarrollo, por lo que la comunidad de Contiki-NG no recomienda su uso para futuras investigaciones. Considerando el uso en la literatura de Cooja, para este trabajo se evaluó la utilización de este simulador. Sin embargo, como se discute en la sección 2.1.1 del capítulo 6, se determinó que cooja no es compatible con una simulación a nivel hardware. Por lo que se decidió utilizar el simulador Renode, descrito a continuación.

4.4.2. Simulador Renode

Renode es un nuevo simulador desarrollado por la empresa Antmicro escrito en c#. Se centra en simular diferentes dispositivos reales compatibles de la manera más realista posible tanto interna como externamente (Antmicro, 2018). Cuenta con las siguientes características:

- Capaz de simular diversos dispositivos compatibles a nivel de hardware real. Por esta razón se puede instalar cualquier sistema operativo compatible en dichos dispositivos.
- Es compatible con la interconexión de analizadores de tráfico de red como Wireshark, lo que abre un amplio abanico de posibilidades para la obtención de métricas detalladas de red.
- Actualmente es compatible con 3 tipos de simulación de radio:
  - Modo simple: Entrega correctamente todos los paquetes a todos los nodos de la simulación.
  - Modo en rango: Se define un rango por nodo y se entregarán todos los paquetes correctamente a los dispositivos que estén dentro de dicho rango.
  - Modo rango con pérdida: Similar al modo de rango, pero agrega una pérdida probabilística configurable al alejarse del nodo transmisor.
- Capacidad de crear simulaciones deterministas mediante el manejo de semillas generadoras de números aleatorios.

Es importante destacar que Renode es un desarrollo relativamente nuevo con un buen potencial para la realización de simulaciones de dispositivos IoT.

4.5. Implementación de la cama de pruebas

Se decidió separar la implementación de la cama de pruebas en tres partes:

- La primera parte consistió en implementar el escenario de aplicación descrito en la Sección 2 del presente capítulo, utilizando la implementación estándar de Contiki-NG para 6LoWPAN con comunicación segura proporcionada por TinyDTLS/AES-128 (la configuración estándar).
- La segunda parte consistió en la implementación de los 5 cifradores seleccionados en TinyDTLS. En la implementación se buscó no modificar el código fuente proporcionado por los desarrolladores de los cifradores, esto con el objetivo de no afectar adversamente su desempeño e introducir sesgos al realizar las comparaciones.

- La tercera parte consistió en la implementación de Contiki-NG en las diferentes plataformas que se seleccionaron para realizar las pruebas. En particular se crearon diferentes rutinas para cada uno de los nodos que conforman la cama de pruebas, donde se incluye la implementación de TinyDTLS con los cifradores a evaluar.

4.5.1. Implementación estándar de la cama de pruebas

Para la implementación estándar de la cama de pruebas fue necesario desarrollar el firmware que correría los 4 tipos de nodos diferentes considerados en el escenario:

- **Nodo baumanómetro.** Este es el nodo que recogerá y transmitirá la información de la tensión arterial medida en un paciente.
- **Nodo termómetro.** Este es el nodo que recogerá y transmitirá la información de la temperatura en un paciente.
- **Nodo oxímetro.** Este es el nodo que recogerá y transmitirá la información de oximetría medida en el paciente.
- **Nodo router de borde.** Este es el nodo que recibe la información generada por cada uno de los nodos sensores para su transmisión hacia la Internet.

Es conveniente mencionar que debido a que esta tesis se enfoca en la evaluación del desempeño de los cifradores ligeros en una red 6LoWPAN con dispositivos de bajo procesamiento, en la implementación de la cama de pruebas no se habilitó la transmisión de las variables hacia la Internet. De este modo el router de borde se convierte en el resumidero final al cual llegarán los paquetes generados por los nodos sensores.

En general, el firmware desarrollado para cada nodo incluye las implementaciones de Contiki-NG necesarias para correr la pila de tecnologías 6LoWPAN descritas en la Sección 3 del presente capítulo (ver Figura 23). En este sentido, el firmware de todos los nodos, independientemente de su tipo, incluye las implementaciones estándar de Contiki-NG para las capas: física (IEEE 802.15.4); de acceso al medio (CSMA-CA); adaptación (6LoWPAN); red con (IPv6, RPL); y transporte (UDP).

En el caso de la capa de seguridad, si bien todos los nodos utilizarán la implementación de TinyDTLS de Contiki-NG, si habrá una diferencia en el rol que tomarán ciertos nodos. Similarmente, en caso de la capa de aplicación CoAP, habrá una diferencia entre los servicios y el firmware que utilizará cada nodo en la cama de pruebas. Por lo tanto, las diferencias en el firmware desarrollado para cada nodo se darán en la capa de seguridad y de aplicación, lo cual se describe a continuación.

4.5.1.1. Implementación de la capa de aplicación CoAP

Recordemos que el protocolo CoAP define 2 tipos de roles para los nodos que forman una red 6LoWPAN:

- Nodo servidor. Son los nodos que ofrecen algún tipo de servicio. Por ejemplo, un nodo sensor puede ofrecer el servicio de lectura y envío de temperatura, tensión arterial, etc.
- Nodo cliente. Son los nodos que solicitan los servicios proporcionados por los nodos servidores. Por ejemplo, un nodo cliente le puede pedir a un nodo servidor que registre la temperatura y la envíe para su procesamiento.

A partir de estos roles, es claro que los **nodos baumanómetro, termómetro y oxímetro** deberán ser **configurados como nodos servidores**. En este sentido, para el envío de cada una de las señales monitorizadas, será necesario que un nodo cliente haga la solicitud de transmisión hacia cada nodo sensor. Alternativamente, un nodo cliente se podría registrar como observador para que el nodo servidor le envíe notificaciones de las mediciones de las variables monitorizadas.

Un nodo cliente puede ser un dispositivo en la nube que solicita la información cada vez que lo requiera. Sin embargo, como lo mencionamos anteriormente, para efectos de esta tesis no se realizará la transmisión de la información hacia la Internet. Por este motivo, **el nodo que recibirá la información** de cada una de las variables monitorizadas y tomará el **rol de nodo cliente en la cama de pruebas será el router de borde**.

4.5.1.2. Implementación de los nodos servidores (nodos sensores)

Para la implementación de los nodos servidores se consideraron los 3 tipos de señales a monitorizar con las características que se muestran en la Tabla 4 de la Sección 2 del presente capítulo. Con base en esta tabla se determinó el tipo de transmisión CoAP que se utilizará para transmitir la información desde los nodos sensores (nodos servidores) hacia el resumidero de información (nodo cliente). Por lo tanto fue necesario desarrollar un firmware diferente para cada nodo servidor considerando el tipo de variable monitorizada. Cabe mencionar que los nodos sensores **no enviarán datos reales**, en su lugar se transmitirá una **datos de una señal emulada que para fines de prueba y experimentación sera la misma en cada una de las pruebas**.

Para el **nodo baumanómetro** se desarrolló un firmware que habilita el **“servicio de baumanómetro”** en CoAP. Se decidió que este servicio utilice el modo de solicitud-respuesta de CoAP, debido a que la tasa de muestreo para la tensión arterial es de 1 muestra (consistente en 64 bits = 8 bytes) cada 10 minutos.

Para el **nodo termómetro** se desarrolló un firmware que habilita el **“servicio de temperatura”** en CoAP. Se decidió que este servicio utilice el **modo de solicitud-respuesta** de CoAP, debido a que la tasa de muestreo para la temperatura es de 1 muestra (consistente en 16 bits = 2 bytes) cada 1 minuto.

Diferente a los nodos baumanómetro y termómetro, el **nodo oxímetr**o tiene una tasa de muestreo de 60 muestras 4 bytes por segundo. Esto implica que si se utilizará el modo de solicitud respuesta de CoAP para habilitar el **“servicio de oximetría”**, un cliente tendría que solicitar paquetes con una o más muestras frecuentemente. Por cada solicitud se generarían al menos 2 paquetes (en caso de que no hubiera pérdidas) y posiblemente se tendrían retransmisiones (ver Sección 3 del presente capítulo). En contraste, si el cliente se registra como observador del nodo oxímetro, entonces el nodo oxímetro le mandará los paquetes con las muestras de oximetría sin necesidad de que el cliente las solicite. Por lo tanto, utilizando el modo observador se puede reducir el tráfico ofrecido a la red por la transmisión de esta variable en comparación con el modo solicitud-respuesta. Por tanto, el **servicio de oximetría utilizará el modo observador**.

Otro punto a observar es que en el caso de la señal de oximetría, si bien el servidor podría mandar un paquete por muestra, esto en la mayoría de los casos es ineficiente. Por lo tanto, para la implementación de la cama de pruebas se decidió que sería conveniente poder variar el número de muestras a transmitir por paquete con el fin estudiar el comportamiento de los cifradores y la red al variar el tamaño del paquete. En este sentido el servicio de oximetría permite variar el número de muestras por paquete transmitidas en el modo observador, y por lo tanto la tasa de generación de paquetes también será variable.

4.5.1.3. Implementación del nodo cliente (router de borde - resumidero de información)

Como se mencionó previamente, el router de borde será el que reciba la información generada por los nodos sensores (nodos servidores). Por lo tanto este nodo tendrá dos roles:

- **Como router de borde** se encarga de formar y administrar la red 6LoWPAN, lo que incluye admitir nuevos nodos y actualizar y mantener las rutas generadas por el protocolo RPL. Para este rol se utilizaron las implementaciones estándar de Contiki-NG.
- **Como nodo cliente** debe de recolectar la información generada por cada nodo servidor (sensor).

Considerando que en su rol de router de borde no se hicieron modificaciones a las implementación de Contiki-NG, en lo subsecuente a este nodo se le referirá como nodo cliente. Para que el nodo cliente recolecta la información de los nodos servidores, fue necesario desarrollar el firmware para que:

- Haga las solicitudes de información a los nodos termómetro y baumanómetro con las tasas de solicitud mostradas en la Tabla 6
- Se registre como observador con el nodo oxímetro y reciba los paquetes enviados por él (recordemos que el número de muestras por paquete y la tasa de generación de paquetes son variables).

Tabla 6. Señales implementadas en los nodos.

Tipo de Nodo	Tipo de transmisión	Tamaño de paquete	Tasa de solicitud
Baumanometro	Solicitud-Respuesta	8 bytes	1 cada 10 minutos
Termómetro	Solicitud-Respuesta	2 bytes	1 por minuto

4.5.1.4. Habilitación de DTLS

Para habilitar DTLS en la red 6LoWPAN es necesario compilar la imagen de Contiki-NG con el módulo de TinyDTLS habilitado. Una vez habilitado este modo, el procedimiento de handshake de TinyDTLS se ejecuta cuando un nodo solicita por primera vez la transmisión de un paquete seguro. En el caso de la cama de pruebas el nodo que solicita la transmisión de paquetes seguros es el nodo cliente. Por lo tanto, el **nodo cliente inicia la ejecución del procedimiento de handshake** cuando cada vez que le pide por primera vez a un nodo sensor la transmisión de un paquete seguro. Esto se da en el primer intercambio de datos que se tiene después de que un nodo termina el procedimiento para anexarse a la red. Una vez realizado el handshake, la transmisión de los paquetes subsecuentes será cifrada (segura) siempre y cuando el cliente lo solicite, esto sin necesidad de hacer nuevamente el handshake. Cabe hacer notar, que por la manera modular en la que está implementado TinyDTLS, no es necesario meterse a fondo con peculiaridades del proceso de cifrado AEAD para mandar un texto plano encriptado. En particular, CoAP solo pasa la carga útil a ser cifrada y TinyDTLS genera el texto cifrado que es entregado a la capa de transporte para su transmisión. Esto significa que la implementación de otros cifradores diferentes a los ya incluidos en TinyDTLS no es una tarea trivial.

4.5.2. Implementación de los cifradores ligeros

Como se mencionó en la subsección anterior, la implementación de los cifradores ligeros en TinyDTLS no fue trivial, ya que no existe una guía clara para realizar esto. Sin embargo, ya que el código de TinyDTLS es abierto, fue factible determinar el flujo de procesamiento y las diferentes funciones que se invocan tanto en el proceso de handshake como en el proceso de cifrado. Después de realizar un análisis de dicho flujo y las funciones invocadas, se determinó que para implementar los cifradores ligeros la mejor estrategia sería tratar de “envolverlos” para que trabajaran de mane-

ra transparente en TinyDTLS. Esto se explica a continuación. TinyDTLS utiliza cifrado AEAD con AES-128 por defecto. Una vez que se realiza el handshake y se generan las claves de cifrado, para *enviar un mensaje cifrado*, dentro de TinyDTLS se invoca a una función llamada “dtls_encrypt” del tipo int para aplicar cifrado autenticado a los datos proporcionados por CoAP. Esta función tiene como entrada los siguientes datos como argumentos.

- **Un texto plano src**, que contiene los datos que se van a cifrar y autenticar. Para este trabajo se corresponde a la trama completa del protocolo de aplicación CoAP.
- **Los datos asociados aad**, que contiene los datos que se van a utilizar para autenticación. Estos datos no son cifrados y se generan siguiendo el estándar de DTLS (ver Capítulo 3 Sección 4.3).
- **Una clave secreta key**, que se va a utilizar para cifrar el texto plano. Esta llave se genera durante la ejecución del protocolo de handshake, tal como se explica en el capítulo 2.
- **Un nonce nonce**, que es un número arbitrario que se debe utilizar solo una vez. En el caso del presente trabajo el nonce se genera siguiendo el estándar de DTLS, McGrew (2008).

A partir de estos datos esta función retorna el texto cifrado que se va a transmitir y la longitud de éste en numero de bytes. Después TinyDTLS crea el paquete que será entregado a la capa de transporte y que incluye que incluye los datos asociados (encabezado), nonce y texto cifrado.

Cuando se recibe un mensaje cifrado, dentro TinyDTLS se utiliza la función “dtls_decrypt” para retirar el cifrado de los datos en el paquete que llegó al receptor. Esta función pide los siguientes datos como argumento:

- **Una clave secreta key**, utilizada en la operación de cifrado.
- **El nonce nonce**, que se utilizó en la operación de cifrado.
- **Los datos asociados aad,** que se utilizaron en la operación de cifrado.

- **El texto cifrado src**, contenido en el paquete generado en la operación de cifrado.

Con estos datos la función dtls_decrypt intentará descifrar el texto cifrado y si tiene éxito retorna el texto plano y su longitud. En el caso de que el descifrado no tenga éxito o exista un problema con la autenticación de los datos asociados, se retornará un indicador de error en forma de una longitud en numero de bytes de -1.

Por otro lado se revisó cómo funcionaban las implementaciones de referencia y los requisitos que solicitó el NIST (NIST, 2018) para los candidatos seleccionados entre los que se destacan los siguientes:

- Se debió requisitar un archivo llamado “api.h” con 5 definiciones, el cual deberá llevar lo siguiente (NIST, 2018):
  - La longitud de la clave CRYPTO_KEYBYTES.
  - La longitud del nonce CRYPTO_NSECBYTES.
  - Cuánto aumenta el texto cifrado en comparación con el texto plano CRYPTO_ABYTES.
  - Si el texto plano se sustituye por el texto cifrado CRYPTO_NOOVERLAP debe valer 1 (0 si es lo contrario).
  - Y por último CRYPTO_NSECBYTES que siempre debe valer 0.
- Además debió proporcionar un archivo cuyo nombre tiene que ser “encrypt.c” debe tener las siguientes funciones (NIST, 2018).
  - Una función denominada “crypto_aead_encrypt” la cual se encarga de cifrar el texto plano a partir de los siguientes datos entregados como parámetros.
    - **Un texto plano m,** que contiene los datos que se van a cifrar y autenticar.
    - **Los datos asociados ad,** que contienen los datos que se van a autenticar pero no a cifrar.
    - **Una clave secreta k,** que se va a utilizar para cifrar el texto plano (información).

    - **Un nonce npub,** que es un número arbitrario que se debe utilizar solo una vez.
- Una función denominada “crypto_aead_decrypt” la cual se encargará de descifrar el texto cifrado a partir de los siguientes datos entregados como parámetros.
    - **Una clave secreta k,** utilizada en la operación de cifrado.
    - **El nonce npub,** que se utilizó en la operación de cifrado.
    - **Los datos asociados ad,** que se utilizaron en la operación de cifrado.
    - **El texto cifrado c,** contenido en el paquete generado en la operación de cifrado.

Después de analizar el código de las implementaciones de referencia de los cifradores ligeros, se concluyó que, sería viable “envolver” las dichas implementaciones de referencia para que funcionen de manera “transparente” o “quasi-transparente” en TinyDTLS. Esto debido a la similitud de los argumentos utilizados por las implementaciones de referencia de los cifradores ligeros con los argumentos utilizados por las funciones “dtls_encrypt” y “dtls_decrypt” de TinyDTLS. En particular, la metodología de implementación que se utilizó fue la siguiente:

- Crear una macro de compilación en c para cada uno de los candidatos y sus variantes, esto para poder seleccionar a nivel compilación qué variante reemplazaría al cifrado AES-128 en cada compilación. Un ejemplo de las definiciones de las macros de compilación agregadas se muestra en la Tabla 7

Tabla 7. Código de ejemplo de las macros de compilación agregadas.

```
#ifdef TLS_EXP_CIPHER_GIFTCOFB
#ifdef TLS_EXP_CIPHER_XOODYAK
#ifdef TLS_EXP_CIPHER_ASCON128
#ifdef TLS_EXP_CIPHER_ASCON128A
#ifdef TLS_EXP_CIPHER_ASCON80
#ifdef TLS_EXP_CIPHER_GRAIN128
```

- Sustituir las funciones dtls_decrypt y dtls_encrypt por una función del mismo nombre pero que en lugar de utilizar el cifrado AES-128 utilice las versiones de

cifrado de las implementaciones de referencia. La Tabla 8 muestra un ejemplo del código de una función modificada para cifrado con GIFTCOFB.

Tabla 8. Ejemplo de función dtls_encrypt modificada para cifrado con GIFTCOFB.

```
#ifdef TLS_EXP_CIPHER_GIFTCOFB
#include "giftcofb/armcortexm_fast/crypto_aead.h"
int
dtls_encrypt(const unsigned char *src, size_t length,
       unsigned char *buf,
       const unsigned char *nonce,
       const unsigned char *key, size_t keylen,
       const unsigned char *aad, size_t la)
{
 //printf("encriptando con giftcofb");
 unsigned char msgp[length];
 memcpy(msgp,src,length);
 unsigned long long lencip;
 crypto_aead_encrypt(buf,&lencip,msgp,length,aad,la,NULL,nonce,key);
  return lencip;
}
```

- Modificar el tamaño de los diferentes arreglos encargados de gestionar y almacenar los datos y las claves de cifrado para adaptarlos a cada uno de los cifradores analizados. Para esto se utilizaron los datos que nos ofrece la implementación de referencia en el archivo api.h.
- Modificar los archivos de compilación para poder configurar qué tipo de cifrado utilizar en cada compilación.
- Crear nuevos subdirectorios para incluir los archivos necesarios de referencia para cada uno de los candidatos.

Siguiendo esta metodología se pudieron implementar todos los cifradores seleccionados con sus diferentes variantes como se muestran en la Figura 30.

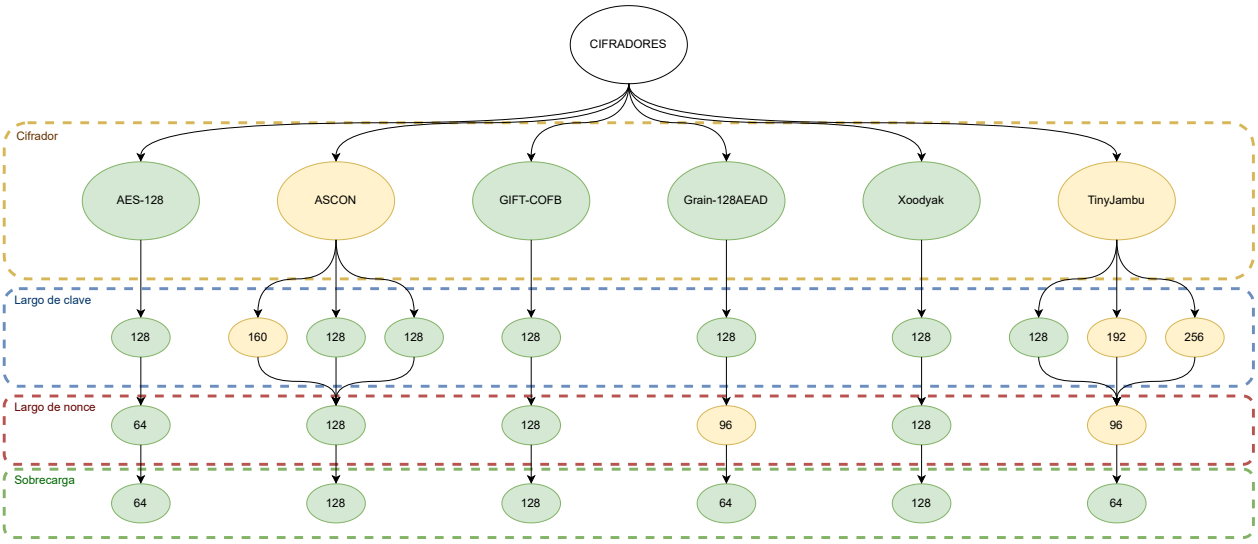


Figura 30. Cifradores implementados en TinyDTLS.

Un aspecto importante a resaltar es que debido a que se cambió el tamaño de los buffers de memoria, se pueden ocasionar fallos de memoria al ejecutar el código. Por lo que esta es una implementación de los algoritmos de cifrado útil para los efectos de la investigación presentada, pero no se recomienda su utilización en aplicaciones finales para entornos de despliegue. En la Figura 30 se resalto en amarillo los cifradores en los cuales se cambio el mayor número de arreglos de memoria, esto debido a que son los menos similares a AES-128 en cuanto a longitud de clave y longitud de nonce.

Como se mencionó anteriormente, TinyDTLS funciona como un módulo adicional en el sistema operativo Contiki-NG por lo que no se necesitó hacer cambios adicionales en el sistema operativo para poder ejecutar la versión modificada.

4.5.3. Implementación en el simulador Cooja

Como se mencionó anteriormente, actualmente el simulador Cooja no es tan utilizado debido a que su desarrollo se encuentra estancado. De hecho, al día de realización de este trabajo, solo se encontró disponible un solo tipo de dispositivo virtual para poder usar con el simulador llamado Cooja Mote (Simon Duquennoy, 2018). Cooja Mote es un dispositivo virtual sin limitaciones de memoria cuya mayor desventaja es que funciona como un proceso interno del simulador Cooja, por lo que no se puede emular a un dispositivo real (Simon Duquennoy, 2018). Dado a que el Cooja Mote no tiene

limitaciones de memoria y el simulador está diseñado para ser 100 % compatible con Contiki-NG la implementación sobre esta plataforma fue directa.

4.5.4. Implementación en el simulador Renode

La plataforma de simulación Renode (ver Figura 31) funciona como una plataforma en la cual se pueden emular dispositivos compatibles que existen en la realidad dando como resultado simulaciones muy precisas y similares al mundo real (Antmicro, 2018). La idea inicial fue implementar en el simulador la misma plataforma física que se utilizó, esto es, el Sensortag CC2650. Lamentablemente dicha plataforma aún no está incluida en Renode, por lo que se decidió utilizar la plataforma compatible más similar con la que contase el simulador.

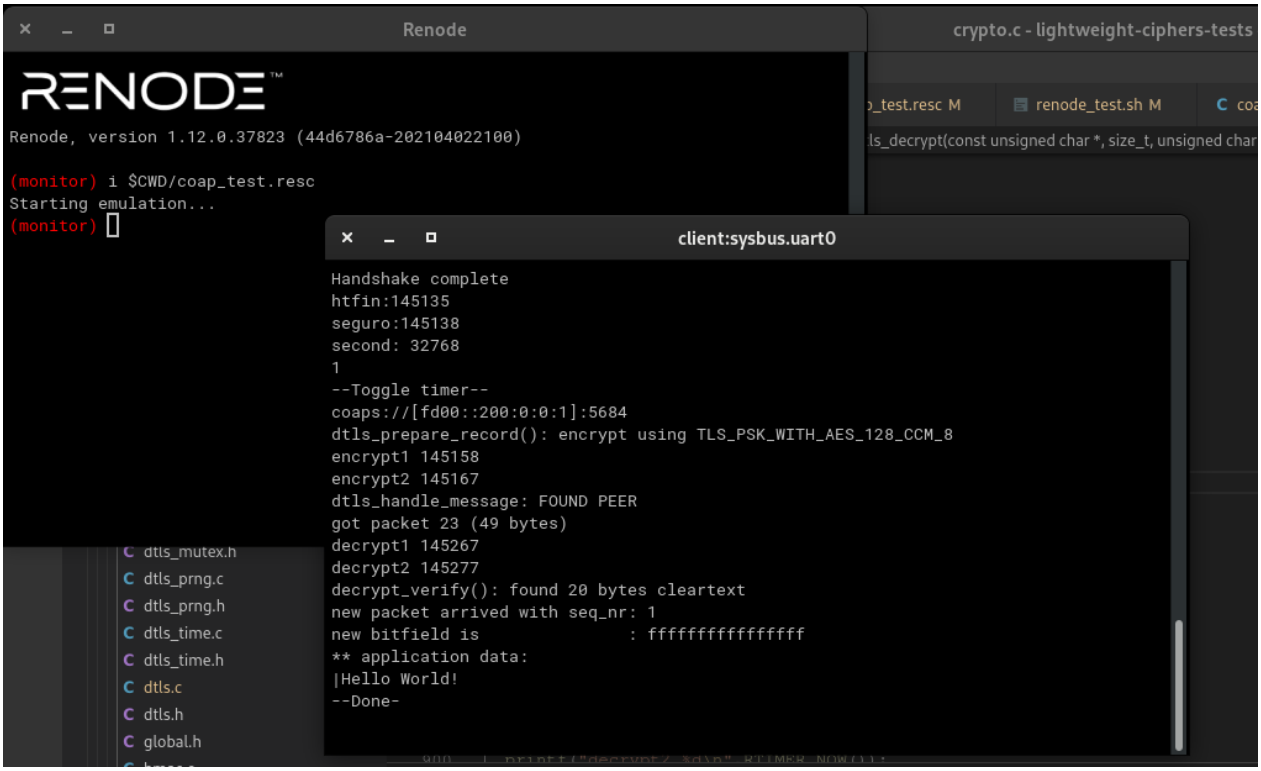


Figura 31. Imagen de la interfaz de línea de comando de Renode.

La plataforma que se decidió utilizar en el simulador fue el mote CC2538 de texas Instrument, el cual es una plataforma ARM de bajo consumo que cuenta con las siguientes características:

- Procesador de 32 bits ARM-cortex-M3.

- 32 KB de RAM (el Sensortag CC2650 tiene 20KB).
- 256KB de ROM (el Sensortag CC2650 tiene 128KB).
- Compatible 100 % con redes 6LoWPAN y Contiki-NG.

En general esta plataforma tiene prestaciones equivalentes de procesamiento que la plataforma Sensortag CC2650, siendo la diferencia principal la memoria de las dos plataformas. Ambas plataformas soportan 6LoWPAN y Contiki-NG. Para implementar el sistema operativo en esta plataforma de simulación se utilizó la siguiente metodología:

- Compilar la imagen de Contiki-NG especificando la arquitectura como CC2538.
- Especificar al simulador mediante un archivo de configuración el tipo de nodo que se va a emular y en donde está el binario compilado que utilizará.
- Especificar el tipo de canal y métricas que se van a recabar.

Cabe hacer notar que la implementación en esta plataforma de simulación es muy similar a la que se realizaría en una plataforma real.

4.5.5. Implementación en la plataforma Sensortag CC2650

Para implementar el sistema operativo en la plataforma física se siguió la siguiente metodología:

- Compilar la imagen de Contiki-NG especificando la arquitectura como CC2650.
- Utilizar un software especializado para poder escribir sobre la memoria del dispositivo, el que se utilizó en este trabajo fue “Uniflash”.
- Escribir la imagen compilada correspondiente para cada nodo.
- Probar que los nodos formarán la red y operarán en el modo esperado.
- Hacer las correcciones necesarias derivadas de problemas detectados en las pruebas.

4.6. Sumario

Como sumario en este capítulo se introdujo el escenario de evaluación y las soluciones tecnológicas disponibles para su implementación en una cama de pruebas. En particular se revisaron las opciones de sistemas operativos escogiendo Contiki-NG para la implementación de la cama de pruebas. Se revisó la capa de aplicación CoAP de Contiki-NG y se describió el firmware desarrollado para la cama de pruebas. Se estudió la implementación TinyDTLS utilizada por Contiki-NG y se propuso una forma de implementar los 5 algoritmos de cifradores ligeros seleccionados en el capítulo anterior. Finalmente, se describió cómo se implementó la cama de pruebas en una plataforma física y dos plataformas de simulación. En el siguiente capítulo se detalla cuales fueron y cómo se llevaron a cabo cada una de las pruebas seleccionadas para evaluar la sobrecarga causada por los algoritmos de cifrado ligero en una red 6LoWPAN.

Capítulo 5. Análisis de sobrecarga de los cifradores ligeros en WSN

5.1. Introducción

En los capítulos anteriores se describieron de manera detallada todas las tecnologías que se utilizaron en este trabajo de investigación, así como la manera en la que fueron implementadas e integradas.

En este capítulo se explica detalladamente cada una de las pruebas que se llevaron a cabo para determinar cómo cambia el comportamiento de la red y de sus nodos al implementar diferentes algoritmos de cifrado ligero. Para lograr esto, se realizaron las pruebas considerando dos fases principales:

- La primera fase tuvo como objetivo validar la cama de pruebas que se detalló en el Capítulo 4, además de recoger métricas preliminares del comportamiento de la red al variar entre los diferentes cifradores.
- La segunda fase tuvo el objetivo de tomar métricas de red y del consumo de energía en la cama de pruebas. Esta fase permitió evaluar la sobrecarga que se puede producir al implementar los diferentes algoritmos de cifrado ligero en una red IoMT como la que se planteó en el presente trabajo.

5.2. Pruebas preliminares

En primera instancia se llevaron a cabo una serie de pruebas preliminares, con el objetivo de validar la cama de pruebas y dar información preliminar respecto a la sobrecarga que se genera al implementar los cifradores ligeros. Estas pruebas se describen a continuación.

5.2.1. Tamaño de imagen compilada

La primera prueba que se realizó fue calcular cómo varía el tamaño del archivo resultante de la compilación del sistema operativo con diferentes algoritmos de cifrado.

En lo subsecuente, a este archivo se le llamará imágen. Para calcular esta métrica se llevó a cabo el siguiente procedimiento:

- Se compiló una versión de sistema operativo para cada uno de los candidatos de algoritmos de cifrado considerados en la evaluación.
- Utilizando la herramienta “size” (la cual está incluida en el entorno de desarrollo de Contiki-NG (Oikonomou, 2018)) se calculó el uso que hacen las imágenes de memoria RAM y la memoria ROM del dispositivo. Está información se registró para su análisis.

5.2.2. Velocidad de cifrado/descifrado

La prueba de velocidad de cifrado fue diseñada para dar una idea inicial de que tanta diferencia existe entre los algoritmos de cifrado ligero en comparación con el algoritmo de cifrado AES-128 (que es el que utiliza por defecto tinyDTLS en su implementación). Además, esta prueba permitió probar el funcionamiento de la implementación de la cama de pruebas. Para su realización, se utilizaron la plataforma física Sensortag CC2650 y el simulador Renode, debido a que el simulador Cooja no es compatible con la simulación a nivel hardware.

5.2.2.1. Métrica de la prueba

Se seleccionó el tiempo de ejecución del algoritmo de cifrado como métrica. La métrica se generó utilizando un timer para medir el tiempo desde el inicio de la función de cifrado/descifrado hasta que se registra el regreso de la función.

Para poder obtener marcas de tiempo lo más precisas posibles, se decidió hacer la implementación a nivel del sistema operativo Contiki-NG. Para esto se utilizó el módulo **Rtimer** (Elsts, 2020a) de Contiki-NG, el cual es un contador de ticks del sistema que inicia desde el momento en que se inicia el sistema operativo. La tasa de conversión de ticks del Rtimer a segundos varía para cada arquitectura. Específicamente, 1 segundo equivale a 32768 ticks en el simulador Renode utilizando la plataforma CC2538,

mientras que para la plataforma fisica Sensortag CC2650 1 segundo equivale a 65536 ticks.

Entonces, para medir el tiempo, durante una simulación o ejecución en la plataforma física se imprime en un archivo el valor del registro de Rtimer en puntos de ejecución de código predeterminados. De esta manera el archivo contiene las marcas de tiempo que permiten calcular cuanto dura la ejecución de una parte específica del programa. El tiempo transcurrido se calcula con la ecuación 1

$$Tiempo\ transcurrido\ en\ ms = \frac{T2 - T1}{NoTicks} * 1000 \quad (1)$$

En donde $NoTicks$ es el valor en ticks equivalentes a un segundo para cada arquitectura, $T1$ y $T2$ son las marcas de tiempo impresas en el registro.

A partir del tiempo transcurrido y tomando en cuenta que los paquetes tienen n cantidad de bytes de carga útil y el tiempo transcurrido por operación de cifrado en milisegundos Tms podemos estimar la velocidad en bits por segundo (bps) de cada cifrador con la ecuación 2

$$Velocidad\ en\ bps = \frac{n * 8}{Tms/1000} \quad (2)$$

5.2.2.2. Obtención de la métrica

Para obtener la métrica se llevó el siguiente procedimiento:

- Se creó una rutina para el cliente en donde realizaba 100 solicitudes al servidor de comunicación cifrada.
- El cliente manda una carga útil cifrada de 23 bytes para cada solicitud y el servidor responde con una carga útil de 20 bytes.
- Para calcular el tiempo de cifrado se implementó una marca de tiempo en el punto en donde se llama a la función de cifrado y otro punto en donde retorna la función de cifrado.

- Para calcular el tiempo de descifrado se utilizó una técnica similar poniendo la marca de tiempo en donde se llama la función de descifrado y otra marca en donde retorna dicha función.
- Se construyó un script de python para poder automatizar la prueba ejecutando la siguiente rutina.
  - Para cada uno de los candidatos de cifrado ligero compilar su imagen de Contiki-NG.
  - Indicarle a la plataforma dónde está esta imagen compilada.
  - Indicarle a la plataforma en donde guardar los archivos de registro.
  - Correr la prueba el tiempo suficiente para ejecutar las rutinas y registrar los ticks requeridos para el cifrado y el descifrado de 100 paquetes.llev.
  - Recopilar las métricas de los registros.
  - Volver a comenzar con otro candidato.

5.2.3. Prueba de establecimiento de enlace seguro

Como se explicó en el Capítulo 3, para establecer un enlace seguro con DTLS es necesario efectuar antes un procedimiento de handshake. Durante este procedimiento se configuran los métodos de cifrado y se generan las claves de una sesión de DTLS. Debido a su importancia, se determinó que es relevante analizar el efecto que podrían tener diferentes algoritmos de cifrado en el tiempo de ejecución para el establecimiento del enlace seguro, analizando de forma específica también el proceso de handshake. Esto considerando que los últimos dos paquetes del procedimiento de handshake ya son cifrados y descifrados por el cliente y el servidor. Por lo tanto si se cambia el algoritmo de cifrado por uno que en teoría fuera más rápido o más lento, entonces se esperaría poder registrar un cambio en la duración del todo el proceso.

5.2.3.1. Métricas a evaluar

Para realizar la evaluación del efecto del algoritmo de cifrado en el proceso de establecimiento del enlace seguro, se midió el tiempo de ejecución del proceso completo. Adicionalmente, se registró por separado el tiempo de ejecución del proceso

de handshake, el cual forma parte del proceso de establecimiento de enlace seguro, como se muestra en la Figura 32.

La métrica de tiempo de establecimiento de enlace seguro se planteó al considerar que las redes de sensores IoMT deberían de garantizar la seguridad desde el momento de conformación de la red. Esto debido a la alta sensibilidad de los datos que se transmiten en este tipo de redes. Por otro lado, la métrica de tiempo de handshake nos ofrecerá una idea más clara de qué porcentaje de tiempo consume el handshake en comparación con las otras operaciones de red que se realizan para la conformación de la misma. Ambas métricas se detallan a continuación.

Establecimiento de enlace seguro El tiempo de establecimiento de enlace seguro es el tiempo que transcurre desde el encendido del cliente y el servidor hasta que estos están listos para intercambiar información segura entre ellos. En la Figura 32 se muestran las etapas consideradas para calcular esta métrica, las cuales incluyen: la formación de la red 6LoWPAN; el handshake DTLS; y una solicitud confirmable entre el cliente y el servidor.

Cabe destacar que este tiempo engloba todo el procedimiento del handshake. Dado que tanto el primer como el último mensaje son enviados y procesados por el cliente, este tiempo se midió utilizando solo este nodo.

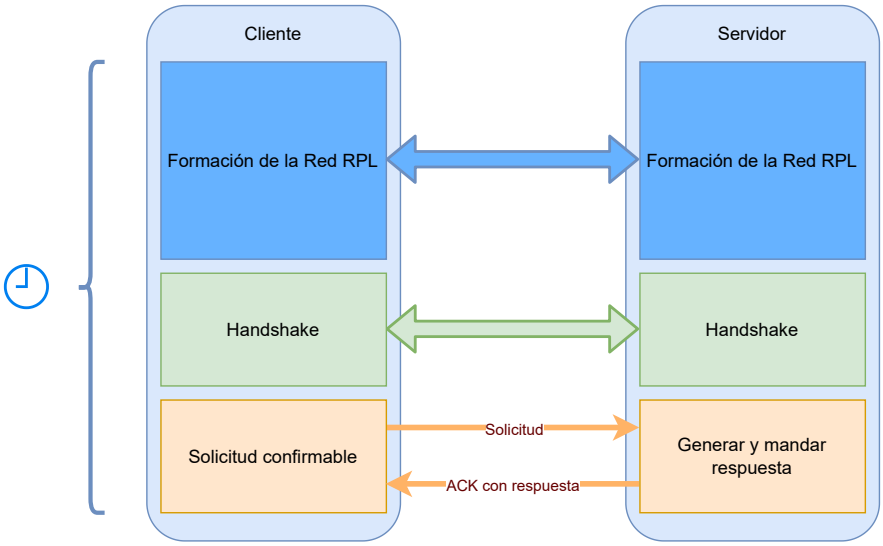


Figura 32. Tiempo definido como enlace seguro.

Duración del handshake Se define como el tiempo en milisegundos que transcurre entre la instrucción de iniciar un handshake y la llamada de handshake exitoso del sistema operativo Contiki-NG (ver Figura 33).

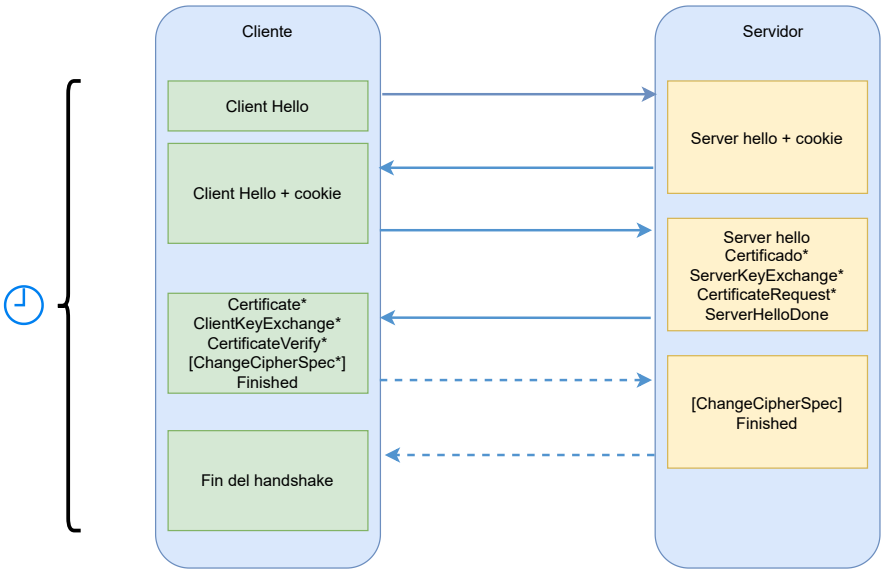
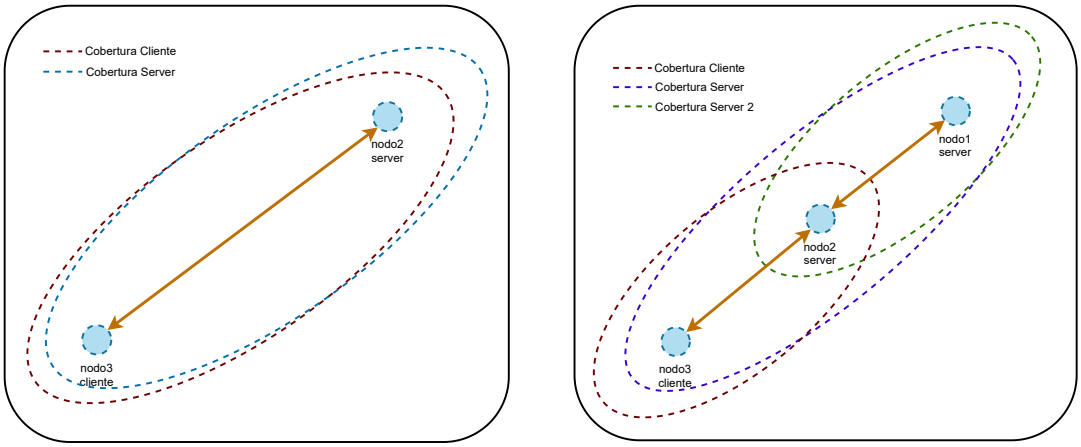


Figura 33. Tiempo definido como la duración del handshake.

5.2.3.2. Obtención de las métricas

En la medición del tiempo de establecimiento de enlace seguro se consideró variar el número de saltos para evaluar el efecto de la retransmisión de paquetes hacia el resumidero.

La Figura 34 muestra gráficamente los escenarios a un salto y multi-salto que se utilizaron para la obtención de la métrica.



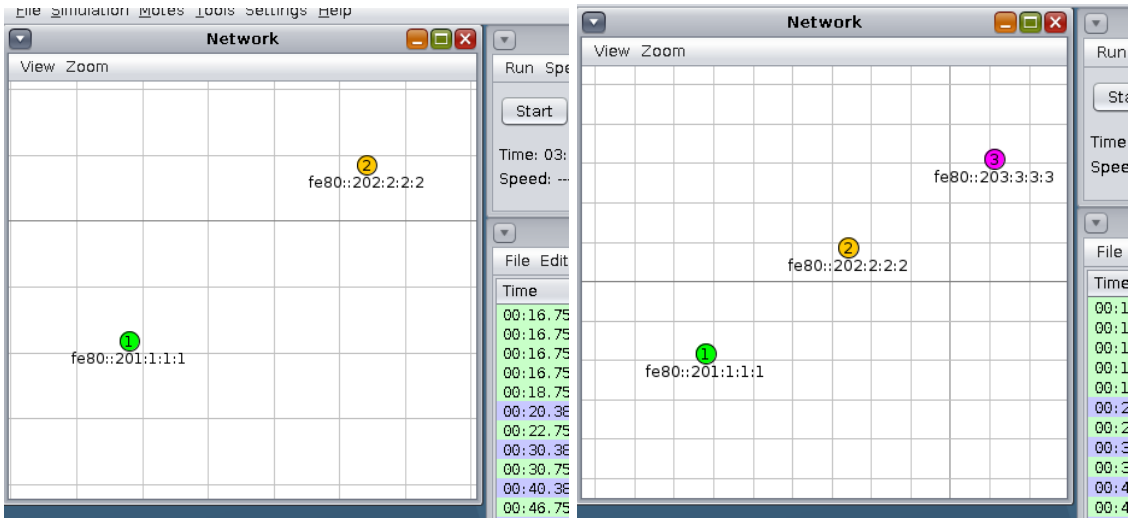
(a) (b)

Figura 34. Escenario a) a un salto, b) multi-salto considerados para la obtención de las métricas.

Para el desarrollo de esta prueba se implementaron cada uno de los cifradores a evaluar las dos plataformas de simulación (Cooja y Renode) y en la plataforma física Sensortag CC2650 descritas en el Capítulo 4. A continuación se da una descripción de estas implementaciones.

5.2.3.3. Implementación en el simulador Cooja

Para la implementación en el simulador Cooja se crearon los dos diferentes escenarios que se muestran en la Figura 35.



(a) (b)

Figura 35. Implementación de los escenarios en el simulador cooja a) a un salto, b) multi-salto.

Posteriormente se creó un script de Python que utiliza la interfaz de línea de comando que proporciona el simulador para automatizar el siguiente procedimiento (representado gráficamente en la Figura 36):

- Para cada uno de los candidatos de cifrado ligero compilar su imagen de Contiki-NG.
- Indicarle al simulador dónde está esta imagen compilada.
- Indicarle al simulador en donde guardar los archivos de registro.
- Correr la simulación el tiempo suficiente para realizar la prueba cuya finalización se detectaba considerando el número de datos entregados por los nodos.
- Recopilar las métricas de los registros.
- Volver a comenzar con otro candidato.
- Este procedimiento se llevó a cabo 10 veces.

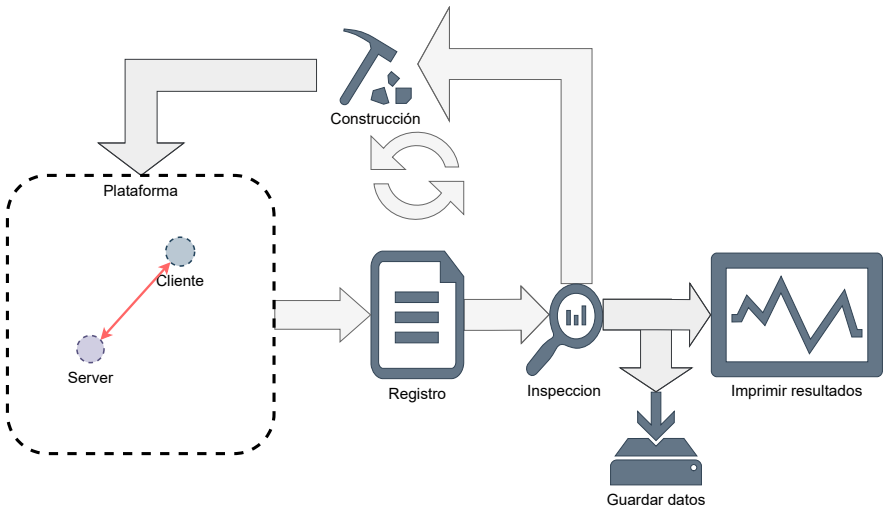


Figura 36. Proceso utilizado para las pruebas.

5.2.3.4. Implementación en el simulador Renode

Al igual que con el simulador Cooja, para implementar la prueba en el simulador Renode se necesitaron crear archivos de configuración en donde se especifican los parámetros de la simulación. Entre estos se destacan los siguientes:

- Mote CC2538 ubicados acorde a los escenarios mostrados.
- Canal en modo rango, para poder simular las redes multisalto.
- Ubicación en donde encontrar la imagen compilada.
- Ubicación en donde guardar los archivos de registro.

Una vez que se preparó el entorno para ambos escenarios se procedió a crear un script en Python muy similar al que se implementó en la primera plataforma. Este script se encargó de: compilar las versiones para cada candidato de Contiki-NG; correr la simulación; y recopilar las métricas (ver Figura 37) un total de 10 veces.

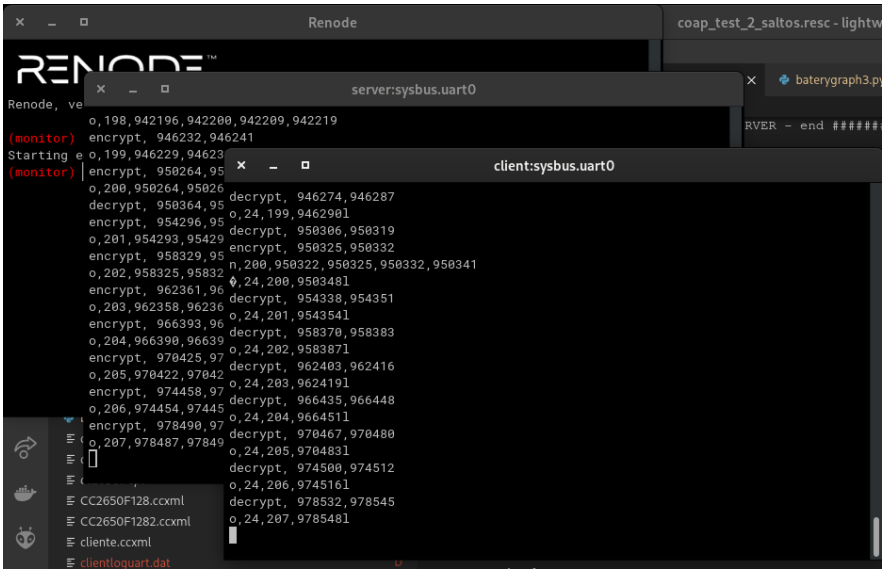


Figura 37. Captura de la salida del simulador Renode en la prueba.

5.2.3.5. Implementación en Sensortag CC2650

Para la implementación en dispositivos físicos se utilizó la configuración que se muestra en la Figura 38 la. Esta configuración considera lo siguiente:

- Se configuró la potencia de transmisión de cada nodo a su nivel más bajo.
- Del análisis de transmisión realizado, se determinó que la distancia de comunicación a 1 salto es de 1.25 m.
- Todos los nodos están conectados mediante USB a la computadora para poder guardar los registros, los cuales se toman mediante el puerto serial a una velocidad de 115200 bps.
- Para la implementación multisalto se requirió utilizar dos computadoras, debido a que no se contaba con cables lo suficientemente largos para estar conectados a una sola. Estas dos computadoras se conectan vía WiFi entre ellas para coordinar la programación y reinicio de los nodos en el momento indicado.

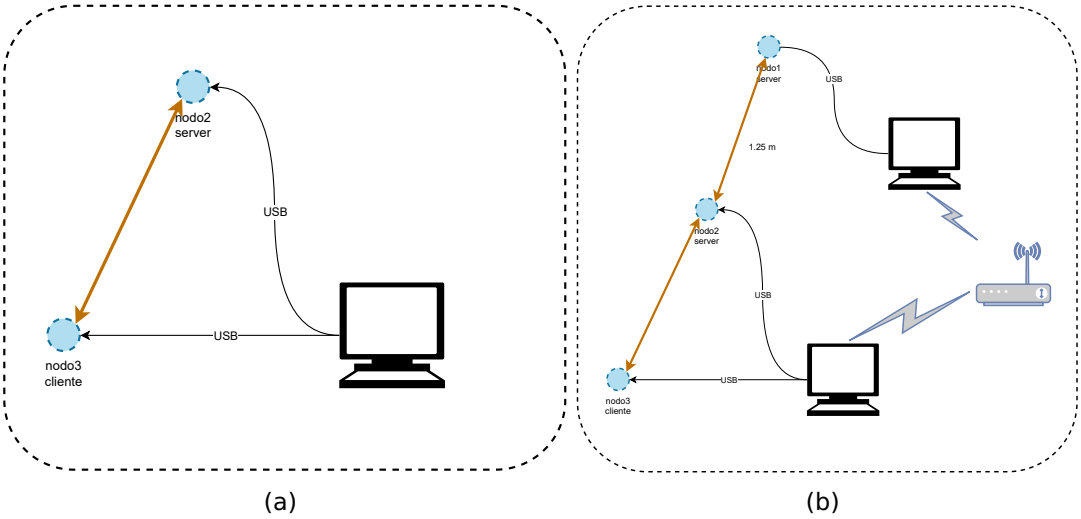


Figura 38. Diagrama de la implementación en físico de los escenarios a) a un salto, b) multi-salto.

Una vez instalado el setup en físico se utilizó un script muy similar a los anteriores para automatizar la programación, reinicio y toma de registro de los nodos un total de 10 veces.

5.3. Pruebas de desempeño de los cifradores

El segundo conjunto de pruebas realizado en este trabajo de investigación está enfocado a medir el efecto de los cifradores en un escenario de IoMT. Este segundo conjunto de pruebas se describen en esta sección.

Para evaluar la sobrecarga que podrían producir los cifradores ligeros en una red IoMT, se partió de un escenario en donde 3 nodos sensores proporcionan servicios de tres variables fisiológicas. Se realizaron 2 conjuntos de pruebas, las pruebas de sobrecarga y las de medición de consumo de energía. Para ambos se utilizaron los mismos escenarios y perfiles de tráfico. Por lo que antes de describir las pruebas y las métricas de cada una de ellas, se detallan los escenarios de evaluación y los perfiles de tráfico.

5.3.1. Empaquetado y sobrecarga de la capa DTLS

Como se revisó en el Capítulo 4, para las pruebas de desempeño realizadas se definieron fuentes de tráfico correspondientes a una red IoMT con 3 dispositivos médicos

correspondientes a un baumanómetro, un termómetro y un oxímetro.

En la Tabla 9 se muestra el perfil de tráfico de cada uno de estos nodos y la tasa de generación de muestras en los escenarios implementados.

Tabla 9. Perfil de tráfico considerado para la prueba de sobrecarga

Nodos	Tamaño de muestra	Tipo de transmisión	Muestras por paquete	Tasa de generación de muestras.	Tasa de bits por segundo ofrecida a la red IoMT.
Oximetro	32 bits (4 Bytes)	Streaming (Observador)	1 muestras	60 por segundo	1920 bps
Baumanometro	64 bits~(8 Bytes)	Solicitud-Respuesta	1 muestras	1 cada 10 minutos	0.106 bps
Termómetro	16 bits~(2 Bytes)	Solicitud-Respuesta	1 muestras	1 por minuto	1.06 bps

Como se puede observar en la Tabla 9, para el caso del nodo baumanómetro y el nodo termómetro la tasa de datos es suficientemente baja para no ocasionar una congestión significativa a la red. Sin embargo en el caso de nodo de oximetría, dado que su tasa de datos es superior a 1 kbps, enviar una muestra por trama puede no ser la solución más adecuada. Por esta razón se decidió analizar cuál es el máximo número de muestras de oximetría que se pueden enviar a la vez dentro de un paquete transmitido en la red IoMT implementada en la cama de pruebas.

5.3.1.1. Tasa máxima de muestras

El estándar IEEE 802.15.4 utilizado en las capas de acceso al medio (MAC) y física (PHY) de los nodos de la cama de pruebas, define una carga útil máxima para la capa PHY de 127 bytes. A nivel de la capa MAC tendremos que solo se pueden utilizar para la carga útil 93 bytes debido a que los demás bytes se utilizan para encabezado.

En estos 93 bytes se tendría que almacenar la trama UDP, la cual teóricamente puede tener una longitud de hasta 65535 bytes. Sin embargo, si un paquete UDP con una longitud mayor a 93 bytes llega a la capa de adaptación 6LoWPAN, entonces esta capa fragmenta la trama en múltiples paquetes antes de pasarlos a la capa MAC. Para estudiar de forma específica el análisis de los efectos que los diferentes cifradores podrían tener en el desempeño de la red, en la capa de transporte se consideraron

paquetes que no requieren ser fragmentados en la capa de adaptación.

Considerando que la trama UDP comprimida por el protocolo 6LowPAN tiene un encabezado de 7 bytes de longitud, en la cama de pruebas cada nodo dispondría de 86 bytes para transmitir datos de capas superiores a la capa UDP, como lo son las capas de seguridad y aplicación. Además, en la cama de pruebas se va a utilizar el protocolo CoAP como capa de aplicación (ver Capítulo 4), por lo que se tendrán que reservar 17 de los 86 bytes para almacenar su encabezado, quedando un total de 69 bytes para poder transmitir la carga útil de la aplicación.

Note que para el streaming de oximetría, dado que se tienen que transmitir 4 bytes por muestra, solo se podrán transmitir como máximo un total de 17 muestras a la vez en un solo paquete.

5.3.1.2. Sobrecarga de la capa DTLS

La disponibilidad para la carga útil estimada en la subsección anterior se basa en una implementación sin DTLS. Por lo que para este trabajo es considerada como la capacidad máxima, en la capa de aplicación, por paquete que no requiere fragmentación.

Es importante remarcar que, la capa DTLS se inserta entre la capa de aplicación y la capa de transporte. Esto implica que la sobrecarga por DTLS es considerada por las capas inferiores como información proveniente de la capa de aplicación. Por lo tanto, para este trabajo la sobrecarga por DTLS se referirá a los bytes extras necesarios para cifrar los paquetes generados por la capa de aplicación y a los encabezados agregados por DTLS.

En particular, al habilitar el protocolo DTLS en nuestra aplicación, se tendrán que reservar 21 bytes para el encabezado de DTLS. Esto implica que solamente quedarían 48 bytes de carga útil para la transmisión de datos en la capa CoAP cuando se utiliza DTLS, como se muestra en la Figura 39.

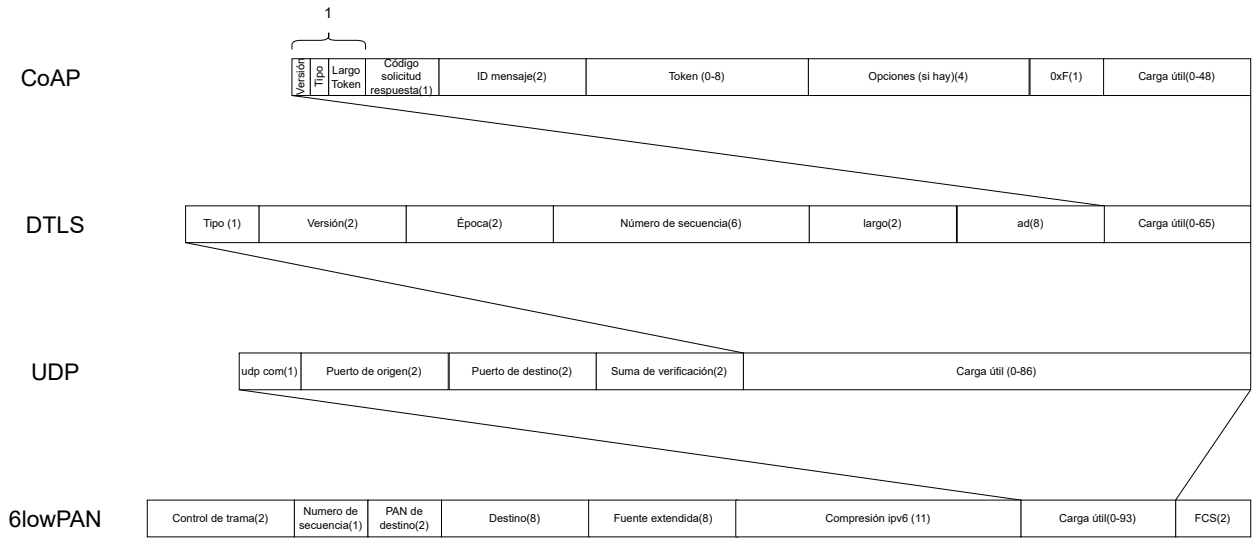


Figura 39. Estructura de la trama utilizada.

Si bien se podría asumir que se podrían transmitir hasta 12 muestras cifradas por paquete considerando que se tienen 48 bytes de carga útil disponibles, además del encabezado, en la capa DTLS se agregará cierta cantidad de bytes al texto plano en el proceso del cifrado. La cantidad de bytes agregada dependerá del cifrador utilizado tal como se muestra en la Tabla 10. Como se puede observar los candidatos que se utilizaron en este trabajo se dividen en 2 grupos principales; los que agregan 2 bytes(16 bits) y los que agregan 1 byte (8 bits). Esto ocasiona que en lugar de poder transmitir 12 muestras de oximetría por paquete, como máximo se podrán transmitir 11.

Tabla 10. Bits que agrega cada cifrador al texto plano

Cifrador	Sobrecarga en bits
Sin cifrado	0
AES-128	8
GIFT-COFB	16
Xoodyak	16
ASCON128	16
ASCON128a	16
ASCON80	16
Grain-128AEAD	8
TinyJambu128	8
TinyJambu192	8
TinyJambu256	8

Basados en esta información se decidió que para las pruebas que evaluarán el

desempeño de los cifradores se utilizaran 3 empaquetados distintos para el nodo de oximetría: 1, 6 y 9 muestras por paquete. Esto nos podrá brindar información del comportamiento de la red al aumentar o disminuir la transferencia de paquetes que se tienen que enviar por segundo.

5.3.2. Escenarios de evaluación

Para estimar la sobrecarga generada por los cifradores se plantearon dos escenarios en los cuales 4 nodos forman una red IoMT con diferentes topologías. En ambos escenarios se consideran 3 nodos sensores (baumanómetro, oxímetro y termómetro) y un resumidero de información. Tal cual se explica en el Capítulo 4, los nodos baumanómetro, oxímetro y termómetro fueron programados como servidores CoAP. Por otro lado, el nodo resumidero se programó como cliente CoAP, por lo cual se registrará como observador al nodo de oximetría y solicitará los datos de los otros dos nodos acorde a la frecuencia de solicitud definida en la Tabla 9. Los dos escenarios se describen a continuación.

5.3.2.1. Escenarios con topología en malla

El primer escenario que se evaluó fue una topología en malla tal cual se muestra en la Figura 40. En este caso cada uno de los nodos tiene a su alcance a nivel de radio a todos los demás, pudiendo transmitir paquetes entre sí directamente.

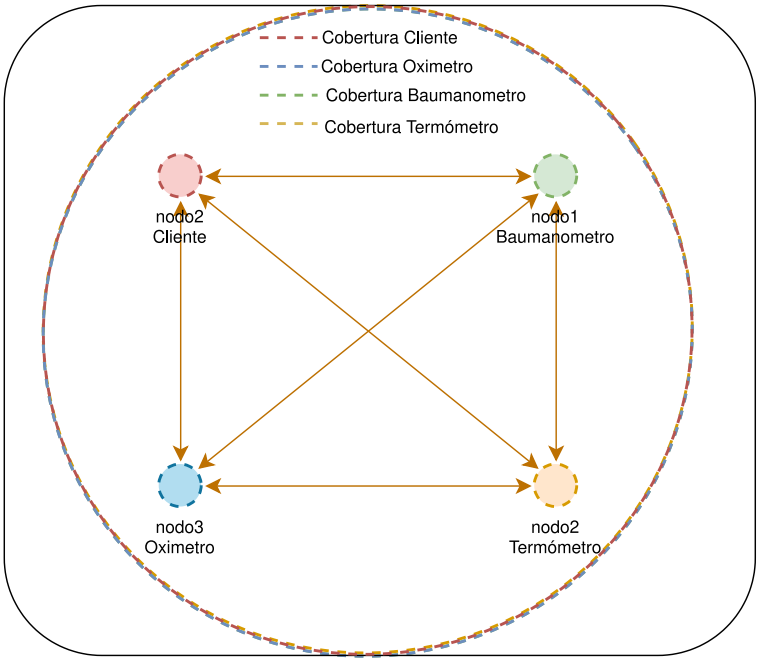


Figura 40. Escenario de malla.

5.3.2.2. Escenarios multisalto

El otro escenario que se implementó fue el de una red multisalto. En este escenario los 4 nodos solo están al alcance de sus dos nodos contiguos como se muestra en la Figura 41.

Cabe destacar también que en este escenario se ubicó en los extremos de la red al nodo cliente y al nodo de oximetría.Esto es debido a que el nodo de oximetría es el que genera la tasa de transferencia más grande, y por lo tanto nos proporciona más elementos de evaluación al inducir más tráfico multisalto en la red.

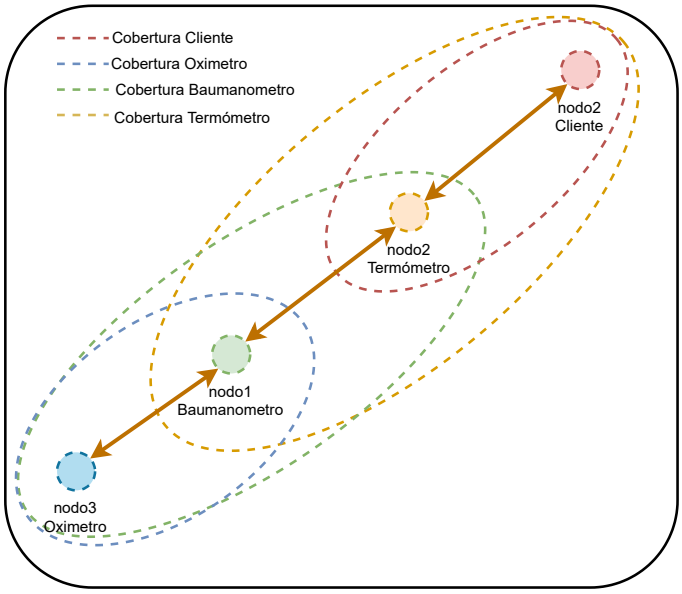


Figura 41. Escenario multisalto.

5.3.3. Cifradores evaluados.

Para realizar las pruebas de sobrecarga iniciales se consideraron todos los cifradores seleccionados. Desafortunadamente, los siguientes candidatos presentaron inestabilidad en la implementación que se propuso para dichas pruebas, la cual consistía en que algunos nodos se quedaban colgados en el arranque lo que se atribuye a fallos en la compilación:

- Grain-128AEAD
- ASCON128
- ASCON128a
- ASCON80

Debido a que la resolución de las inestabilidades estaba fuera del alcance de la presente tesis, se decidió descartar a dichos cifradores de las pruebas restantes. Bajo la consideración anterior, las diferentes variantes de cifradores que se incluyeron en la evaluación se enuncian a continuación:

- AES-128
- Xoodyak
- GIFT-COFB
- TinyJambu128
- TinyJambu192
- TinyJambu256

Adicionalmente, como referencia se consideró un escenario en donde no se utiliza cifrado.

Para cada uno de los cifradores y la opción de no-cifrado se probará la transmisión de oximetría para el empaquetado de 1, 6 y 9 muestras por transmisión. Por lo tanto, se tendrá un total de 21 variantes para la prueba considerando los 6 cifradores, el no-cifrado y las 3 opciones de empaquetado para la oximetría.

5.3.4. Métricas a evaluar

Para esta segunda fase de pruebas se consideraron diferentes métricas como el “End to End delay”, el tiempo de procesamiento, el jitter y la latencia en paquetes solicitud-respuesta. A continuación se describen estás métricas.

5.3.4.1. Tiempo de procesamiento y End to End delay

Las primeras dos métricas que se consideraron fueron el “End to End delay” y el tiempo de procesamiento. El **tiempo de procesamiento** se define como el tiempo que transcurre desde que la carga útil se pone en la cola de transmisión hasta que se transmite. Por otro lado, el **“End to End delay”** se define en este trabajo como el tiempo que transcurre desde que un paquete sale desde el nodo transmisor hasta que llega al nodo receptor (Liu *et al.*, 2013) y se entrega después del proceso de descifrado.

Estas métricas se calcularon utilizando el nodo oxímetro (servidor) y el nodo cliente (resumidero de información) utilizando la siguiente metodología:

- Se toma en el nodo de oximetría una marca de tiempo,$T1$, en el instante en que se coloca en la cola de transmisión la carga útil.
- Después, en el mismo nodo de oximetría se toma una segunda marca de tiempo,$T2$, en el instante en que se transmite el paquete.
- Posteriormente, en el nodo cliente se tomó una tercera marca de tiempo, $T3$, en el momento en el que el paquete llega al programa principal del sistema operativo, (lo que se considera como paquete entregado).

La Figura 42 describe gráficamente los instantes en los cuales se registran los tiempos $T1$, $T2$ y $T3$.

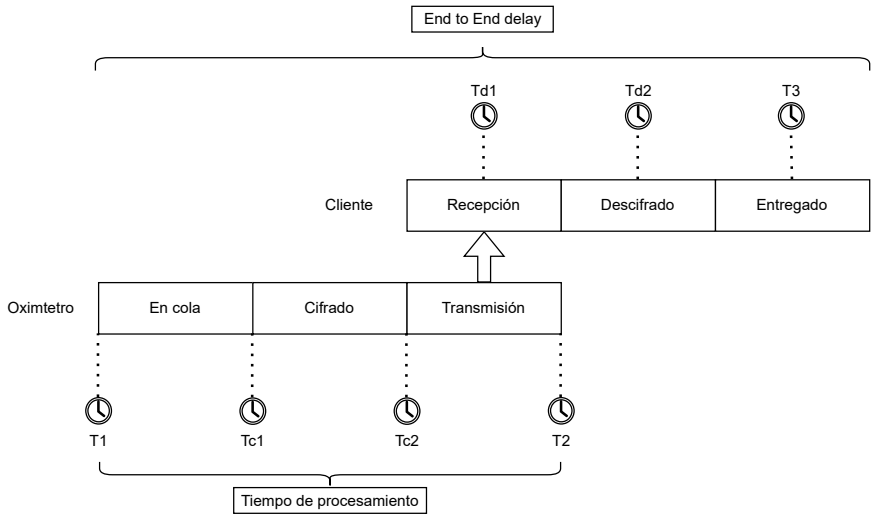


Figura 42. End to end delay y tiempo de procesamiento.

Para obtener el tiempo de procesamiento se calcula la diferencia que existe entre $T2$ (el tiempo de transmisión) y $T1$ (el tiempo en el que se puso el paquete en la cola). para cada paquete que se transmite durante la prueba. Al final se calculó el promedio de todas las transmisiones mediante la expresión de la ecuación 3:

$$Tiempo\ de\ procesamiento\ promedio = \frac{\sum_{i=1}^{n}(T2_i - T1_i)}{n} \tag{3}$$

Para el cálculo del End to End delay se siguió el procedimiento descrito en (Liu *et al.*, 2013), considerando que ambos nodos tienen relojes independientes los cuales no están sincronizados. En particular, en este trabajo de tesis el End to End delay

promedio para un cifrador determinado se calcula obteniendo primero las diferencias entre $T3$ y $T1$ (ver Figura 42) para cada paquete. Ya que se tienen estas diferencias se hace una "normalización" de las mismas al sustraer la diferencia mínima, d_{min}, de todas las diferencias calculadas. De esta manera se lleva a 0 el valor mínimo de las diferencias calculadas. Finalmente, el End to End delay se calcula obteniendo el promedio de las diferencias normalizadas como se muestra en la ecuacion 4.

$$End\ to\ End\ delay\ promedio = \frac{\sum_{i=1}^{n}((T2_i - T1_i) + d_{min})}{n} \tag{4}$$

Cabe resaltar que existen métodos que pueden mejorar aún más la precisión de esta métrica, como el lograr la sincronización de ambos relojes. Sin embargo, considerando que se contaba con un tiempo limitado para la elaboración de este trabajo se optó por este procedimiento, que además refleja condiciones existentes en las redes IoMT, donde los relojes no estarían sincronizados.

5.3.4.2. Jitter

El jitter es una métrica que tiene diferentes definiciones en la literatura. En este trabajo se tomó la definición de (Khan *et al.*, 2013), que considera al jitter como la variabilidad en el tiempo que tarda un paquete desde que sale del servidor hasta que llega al nodo cliente.

Considerando las 3 marcas de tiempo mostradas en la Figura 42, se calculó el jitter como se muestra en la ecuación 5.

$$Jitter\ promedio = \frac{\sum_{i=1}^{n}((T3_{i+1} - T1_{i+1}) - (T3_{i+1} - T1_{i+1}))}{n-1} \tag{5}$$

Cabe resaltar que en este caso aunque los nodos no están sincronizados, el cálculo de jitter resultante es independiente de los relojes de los nodos como se menciona en (Liu *et al.*, 2013).

5.3.4.3. Latencia en paquetes solicitud-respuesta

Dado que las tres métricas anteriores se calcularon sobre el streaming de oximetría, se decidió añadir una métrica más para evaluar la latencia de la transmisión cifrada de paquetes solicitud-respuesta entre el nodo de temperatura y el nodo cliente. En particular se calculó la latencia, Basado en, (Khan *et al.*, 2013), la latencia se definió como el tiempo que tarda en completarse una solicitud-respuesta de CoAP desde el momento en que el nodo cliente añade la solicitud a la lista de tareas hasta que recibe la respuesta desde el nodo de temperatura (servidor), tal cual se muestra en la Figura 43.

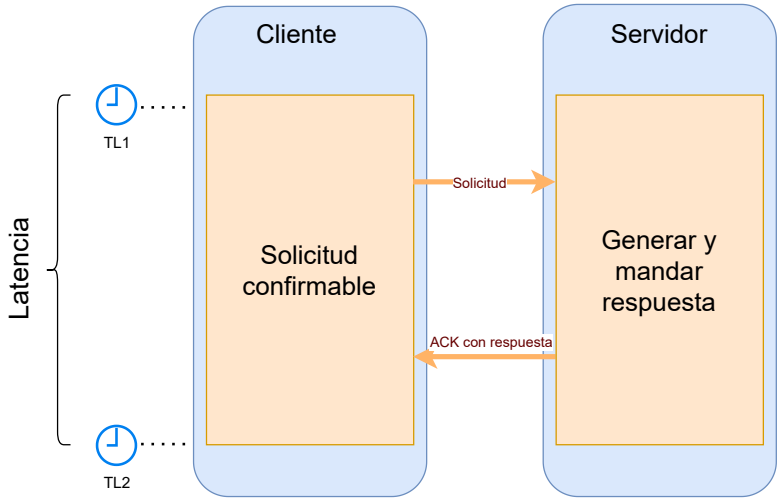


Figura 43. Tiempo considerado como latencia.

Para el cálculo de esta métrica se tomaron 2 marcas de tiempo en el nodo cliente. La primera fue en la llamada la función de solicitud del servicio de temperatura, la cual denominaremos TL1, y la segunda cuando se ejecuta la función de recepción de solicitud del servicio de temperatura, la cual denominaremos TL2. Por lo tanto, la métrica de latencia será el promedio de la diferencia entre estos dos tiempos para cada una de las solicitudes que se envían durante la ejecución de una prueba (ver Ecuación 6).

$$Latencia\ promedio = \frac{\sum_{i=1}^{n}(TL2_i - TL1_i)}{n} \tag{6}$$

Una característica a considerar de esta métrica es que dado que las marcas de tiempo solo se toman en un nodo, las discrepancias a causa de la no sincronización de los relojes no está presente.

5.3.5. Prueba de sobrecarga

Para la implementación de estas pruebas se utilizó la plataforma física sensortag CC2650 utilizando la siguiente metodología:

- Para la implementación del escenario de malla se posicionaron los dispositivos con una distancia aproximada de 10 cm entre cada uno de ellos como se muestra en la Figura 44. Cada uno de los nodos sensores y el cliente se conectaron a un debugger mediante un cable USB, a través del cual se programaban, se recopilaban los registros y se mandaba el comando de reset según se necesitara.

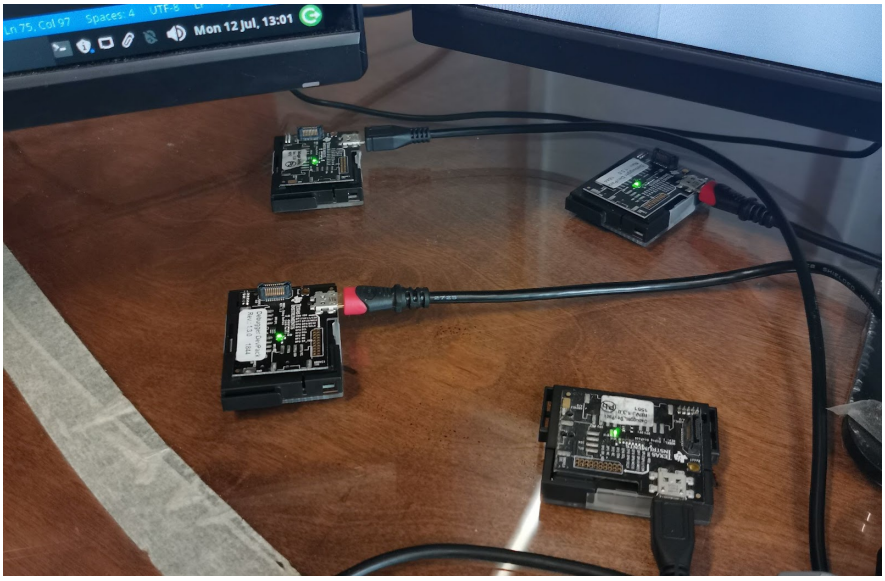


Figura 44. Implementación del escenario de malla.

- Para la implementación del escenario multisalto los nodos se posicionaron a una distancia de aproximadamente 1.25m entre ellos en forma lateral las posiciones previamente mencionadas (ver Figura 45). Al igual que en el escenario anterior, todos los nodos tienen conectados un debugger mediante un cable USB, el cual se utilizaba para programarlos, recopilar registros y reiniciarlos según fuese el caso.

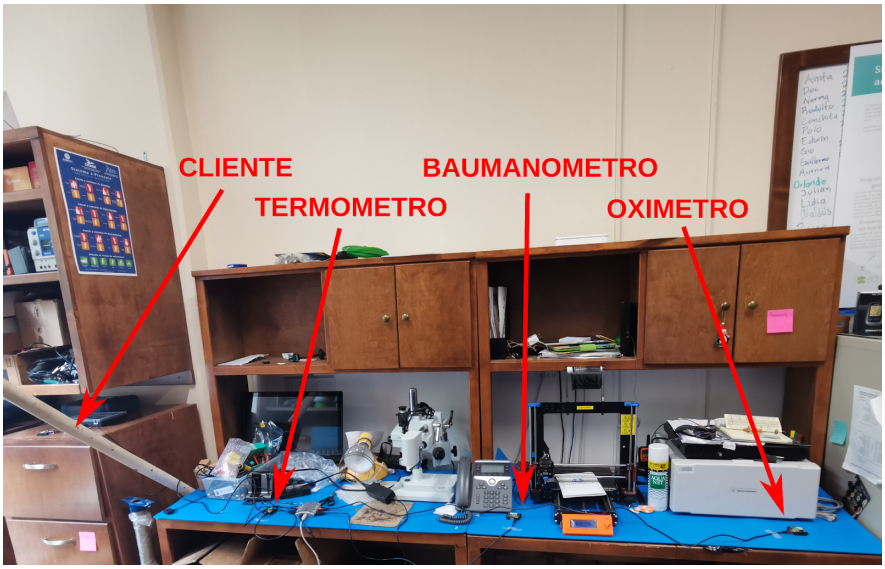


Figura 45. Implementación del escenario multisalto.

- Para la ejecución de las pruebas se desarrolló un script en python que realiza la programación de los nodos y la recopilación de los registros para poder ser analizados posteriormente. Este script siguió el proceso mostrado en la Figura 46 y consiste en lo siguiente:

  - Para cada uno de los cifradores y para cada uno de los tipos de empaquetado utilizados en el streaming de oximetría se compiló una imagen de sistema operativo correspondiente para cada uno de los nodos.
  - Posteriormente se carga en cada uno de los nodos su imagen.
  - Se envía un comando de reset lo más sincronizado posible a cada uno de los nodos.
  - Se inicia la captura de los registros de cada uno de los nodos, los cuales se almacenan en archivos separados para analizarlos posteriormente.
  - La prueba termina cuando se hayan cubierto todos las variantes tanto de cifrado como de empaquetado.
  - Este procedimiento se lleva a cabo en ambos escenarios de pruebas.

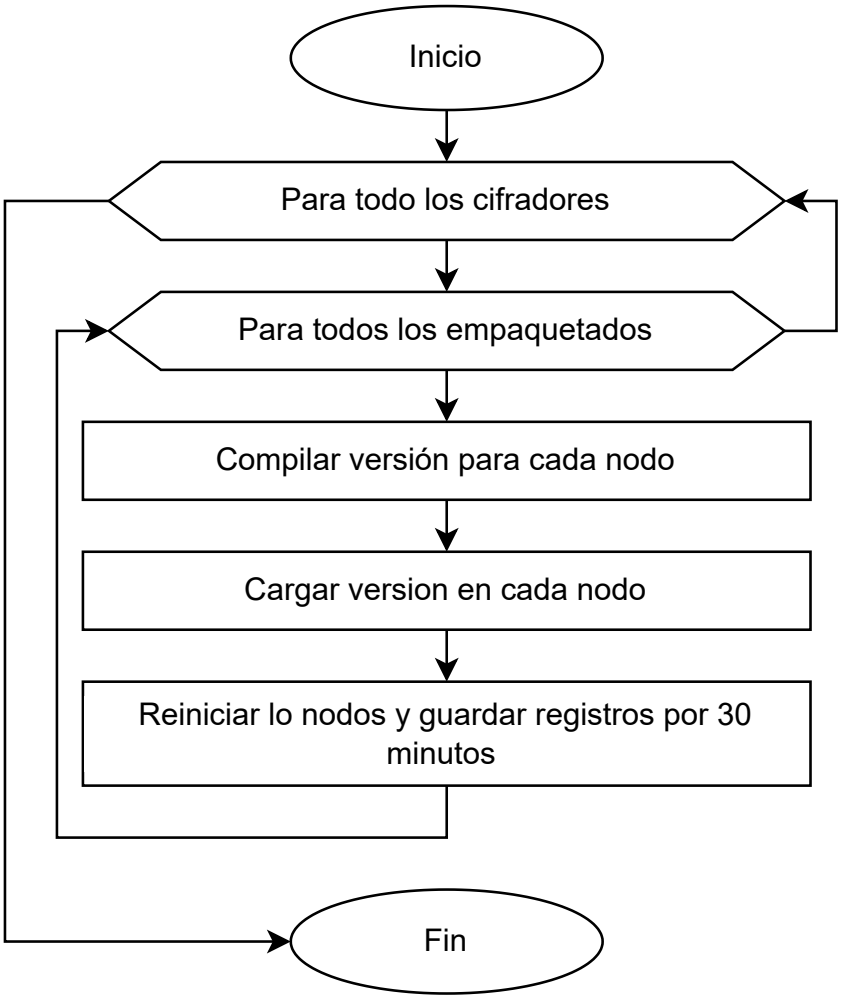


Figura 46. Procedimiento utilizado para la realización de las pruebas.

- Una vez finalizada la ejecución de las pruebas se analizaron los registros mediante scripts de python y se calcularon las métricas, la cuales se analizaran en el siguiente capítulo.

5.3.6. Prueba de consumo de energía

5.3.6.1. Prueba de consumo de energía basado en el módulo energet de contiki-ng

En este conjunto de pruebas se estima el consumo de energía en cada una de las implementaciones descritas en la sección anterior. Esto permite determinar si existe alguna diferencia al cambiar el algoritmo de cifrado ligero en el consumo de energía para esta implementación.

Para poder realizar esta prueba se utilizó el módulo energest del sistema operativo Contiki-NG. El módulo energest se puede utilizar para implementar un enfoque de estimación de energía ligero y basado en software para dispositivos IoT con recursos limitados. Al rastrear el tiempo que se encienden varios componentes de hardware, como la radio, y al conocer el consumo de energía del componente, es posible estimar el consumo de energía (Elsts, 2020b). Lo que proporciona el módulo energest es la cantidad en centésimas de segundo que ha estado encendido un determinado dispositivo o periférico desde que el sistema empezó a funcionar hasta el momento de la solicitud.

Para estimar el consumo de energía utilizando energest se siguió la metodología que se describe a continuación:

- En la implementación principal de las pruebas de sobrecarga explicadas en la sección anterior, se agregó que se imprimieran cada 10 segundos los siguientes datos:
  - El tiempo en décimas de segundo en el que el CPU ha estado encendido en modo normal.
  - El tiempo en décimas de segundo en el que el CPU ha estado encendido en modo de baja energía.
  - El tiempo en décimas de segundo en el que el radio ha estado encendido en modo de recepción.
  - El tiempo en décimas de segundo en el que el radio ha estado encendido en modo de transmisión.
- Una vez terminada la ejecución de las pruebas se obtiene una serie de datos con el formato mostrado en la Figura 47.

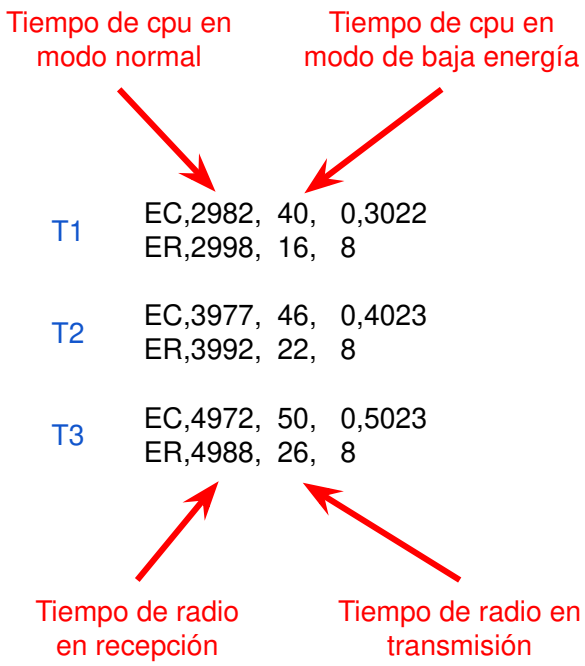


Figura 47. Procedimiento utilizado para la realización de las pruebas.

- Para calcular el consumo de la energía, lo primero que se realizó fue calcular la cantidad de tiempo en segundos que estuvo encendido cada dispositivo. Para esto se realizó una inspección de los datos arrojados por energest registrados en intervalos de 10 segundos como se muestra en la Figura 47. Por ejemplo, para calcular el tiempo que estuvo encendido el CPU en modo normal en el intervalo que va de T1 a T2 en la Figura 47 se utiliza:

$$Tiempo\ CPU\ Normal\ de\ T1\ a\ T2 = \frac{(3977 - 2982)}{100}$$

  La división entre 100 se debe a que energest utiliza como base de tiempo las centésimas de segundo.

- Después, a partir de la hoja de datos del Sensortag CC2650 se obtuvo el consumo de corriente típico que este tiene con una alimentación de 3.3v, el cual es el voltaje de alimentación que se utilizó en las pruebas, obteniendo la tabla 11.

Tabla 11. Consumo típico de corriente del Sensortag CC2650

Módulo	Consumo
CPU modo baja energía	2.7uA
CPU modo normal	550uA
Radio en modo recepción	6.1mA
Radio en modo transmisión	6.1-9.1mA

- Una vez teniendo el tiempo en segundos que cada dispositivo estuvo operando en sus diferentes modos, el consumo de energía en Joules se obtiene multiplicando el voltaje de alimentación de 3.3v por la corriente reportada en la Tabla 11 y el tiempo en segundos como se muestra en la ecuación 7.

$$Energia\ estimada\ en\ Joules = V * A * T \quad (7)$$

- Se repite esta operación para todos los registros de la prueba y para todas las pruebas mediante un script de python, y posteriormente se calcula el promedio para cada uno de los escenarios.

5.3.6.2. Prueba de consumo de batería

El módulo energest estima el consumo de energía de forma indirecta, ya que se confía en la implementación del módulo y en la hoja de datos. Para complementar los resultados, se decidió hacer una prueba de duración de batería.

Para llevar a cabo esta prueba se realizó lo siguiente:

- Caracterizar una batería compatible con el sensortag CC2650 para obtener su curva de consumo.
- Correr las pruebas con un sensor utilizando la batería caracterizada y observar la curva de descarga.
- Analizar la curva de descarga para obtener una estimación del consumo en un entorno real.

Caracterización de la batería Partiendo de que el dispositivo físico utilizado para la implementación de todas las pruebas fue el Sensortag CC2650, se seleccionó una batería que fuese compatible con él para ser analizada. En particular, la batería que fue seleccionada y caracterizada para este análisis fue una batería de iones de litio LIR2032H de tipo botón (LRA01H, 2018). Esta batería es recargable, lo que nos ofrecía la ventaja de poder utilizarla tantas veces fueran necesarias para cada una de las pruebas.

En la Figura 48 se muestra el circuito que se utilizó para la caracterización de esta batería, el cual consiste de un amperímetro conectado en serie junto a la batería y a una resistencia de 60 ohms. Este valor se eligió para que la batería tuviese un consumo estimado de 8mA, el cual es un consumo similar a la corriente de un Sensortag CC2650. También se tiene un osciloscopio que se encarga de tomar el voltaje de la batería. Estos equipos fueron conectados mediante sus respectivas interfaces USB a una computadora para poder registrar todos los valores en archivos de registro.

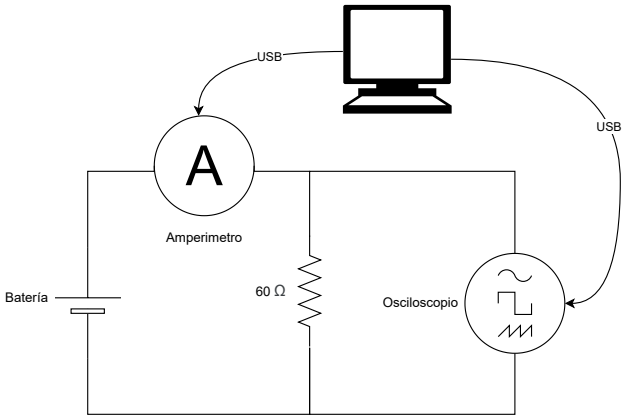


Figura 48. Circuito para la caracterización de la batería.

Las baterías se cargaron al 100 % con su propio cargador y se conectaron al circuito hasta que alcancen un voltaje nominal de 3.0v según los instrumentos. Cuando eso sucede la batería se desconecta del circuito y se procesan los datos. En la Figura 49 se muestran los resultados de esta caracterización para la batería que se utilizó en las pruebas.

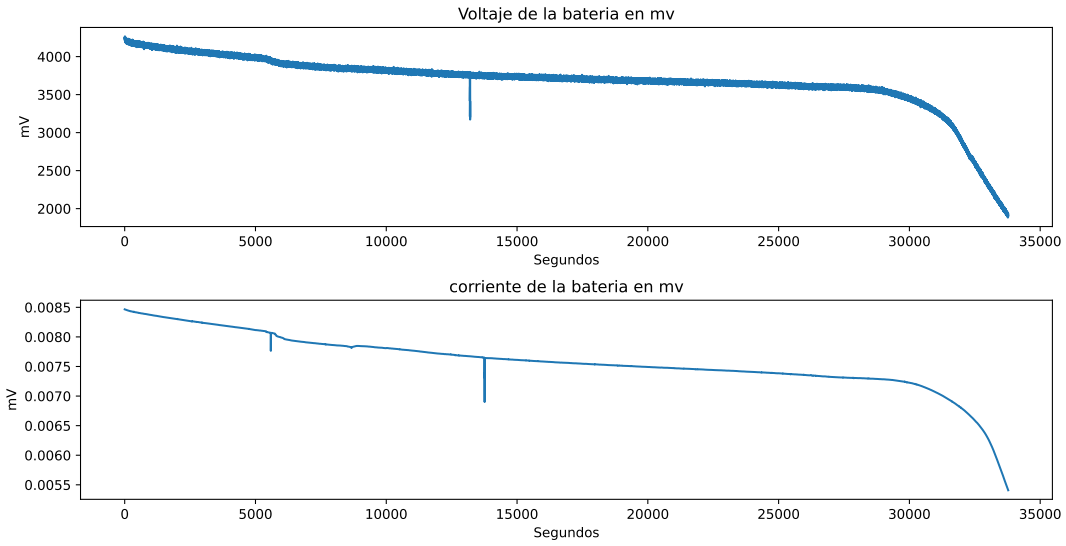


Figura 49. Curva de descarga de la batería utilizada.

Implementación de la prueba en un entorno real Para implementar esta prueba solo se utilizó un nodo oxímetro y el nodo cliente. Se decidió que el nodo oxímetro utilizara la batería que se iba a monitorizar. El nodo cliente se conectó mediante un cable USB y se recolectaron sus registros para asegurarse de que el streaming funcionase durante la prueba. Además de esto se implementó un nodo sniffer que observaría todo el tráfico de la red utilizando el ejemplo de “sensniff” de Contiki-NG. En la Figura 50 se muestra un esquema de la configuración general de la implementación.

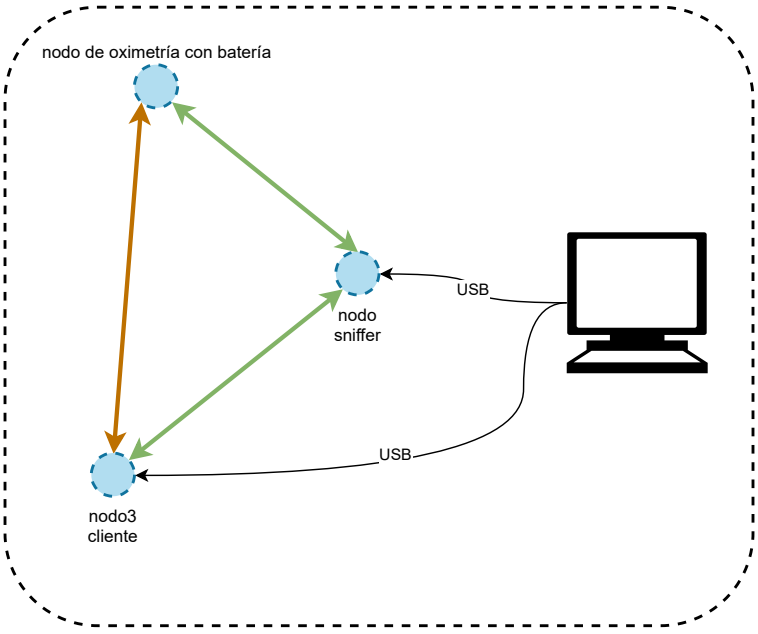


Figura 50. Esquema de la implementación de la prueba de consumo de energía.

Para poder monitorizar el voltaje de la batería se conectó un multímetro en paralelo con el Sensortag CC2650 como se muestra en la Figura 51. Mediante la interfaz USB del multímetro se capturaron valores durante un periodo de 3 horas para todos los tipos de cifradores utilizados en las pruebas de sobrecarga (no cifrado, AES-128, Xoodyak, GIFT-COFB, TinyJambu128, TinyJambu192, TinyJambu256) y para todos los empaquetados de streaming (1 muestra, 6 muestras, 9 muestras). En la Figura 52 se muestra como queda la implementación en el escenario real para esta prueba.

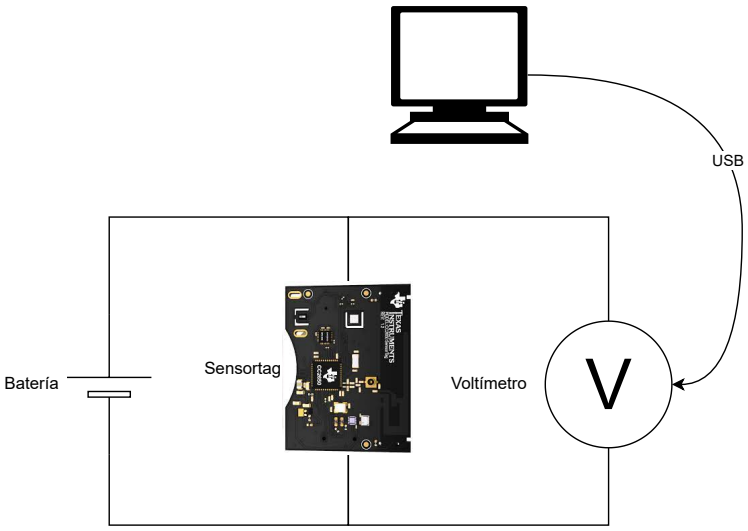


Figura 51. Circuito para el sensado de la batería.

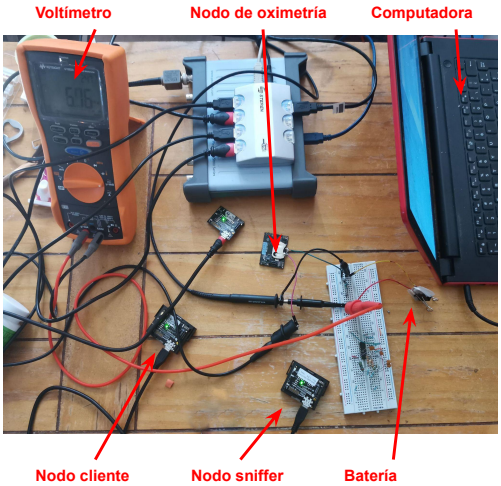


Figura 52. Implementación de la prueba de consumo de energía en un entorno real.

5.4. Sumario

Como sumario en este capitulo se describe de manera detallada cada una de las pruebas que se llevaron a cabo. Estas pruebas se dividen en dos fase principales, la primera que busca evaluar la cama de pruebas y obtener métricas preliminares y la segunda que permite evaluar la sobrecarga que se produce al implementar los diferentes algoritmos de cifrado ligero. En la primera fase es evaluara el tamaño de imagen compilada, la velocidad de cifrado y descifrado y el tiempo para establecer un enlace seguro. En la segunda fase se evaluaran en una red de malla y una red multisalto 6 variantes de los algoritmos mas un escenario sin cifrado, para obtener las siguientes metricas de red; tiempo de procesamiento, end to end delay, jitter y latencia.

Capítulo 6. Resultados y experimentación

6.1. Introducción

En este capítulo se mostrarán los resultados obtenidos al ejecutar las pruebas descritas en el capítulo anterior. Siguiendo la misma estructura que en dicho capítulo, los resultados se presentan en 2 secciones principales:

- Resultados de pruebas preliminares.
- Resultados de pruebas de desempeño.

Estos resultados se presentan a continuación.

6.2. Resultados de las pruebas preliminares

Como se explicó en el capítulo anterior, las pruebas preliminares permitieron probar el rendimiento y la usabilidad de la cama de pruebas, así como darnos una evaluación inicial de la sobrecarga causada por los algoritmos de cifrado. En esta sección se presentan los resultados de las pruebas preliminares detalladas en la sección 2 del capítulo 5.

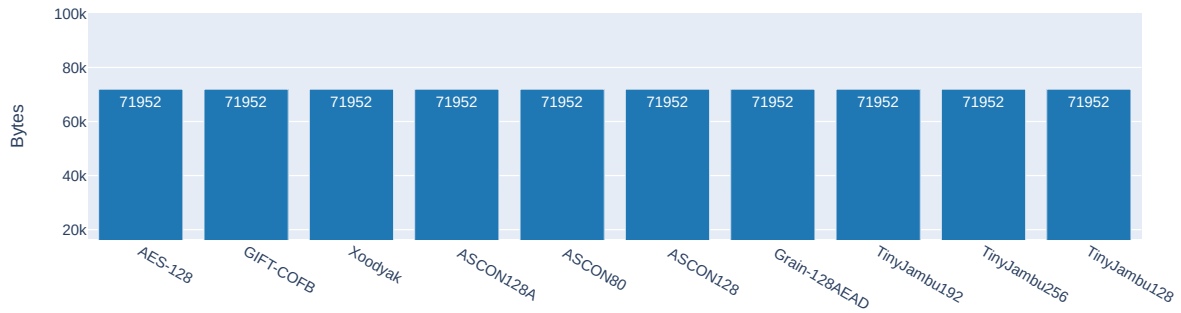
6.2.1. Tamaño de imagen compilada

La primera prueba que se desarrolló fue la estimación del uso de memoria RAM y ROM de una imagen compilada utilizando cada uno de los diferentes algoritmos de cifrado. Esto con el objetivo de observar si existe una reducción en el consumo de recursos a nivel de hardware como se esperaría al utilizar algoritmos de cifrado ligero.

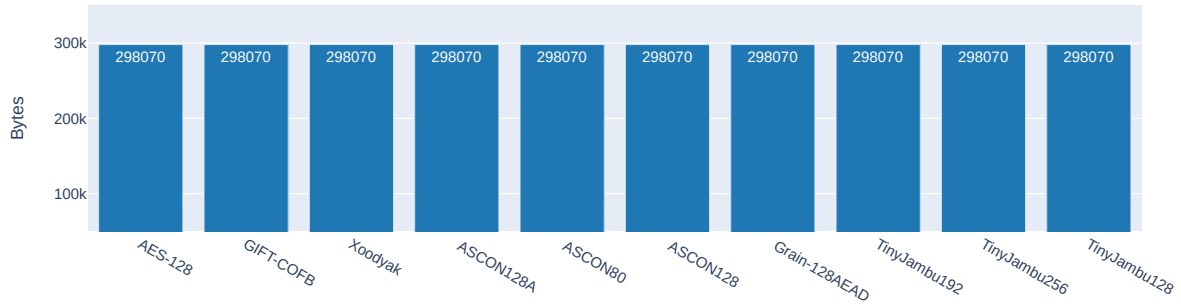
6.2.1.1. Resultados en la plataforma de simulación Cooja

En la Figura 53 se muestran los resultados de esta prueba para la plataforma Cooja, la gráfica superior corresponde al uso de memoria RAM y la gráfica inferior al uso de memoria ROM. En el eje x se posiciona cada uno de los candidatos evaluados y en el

eje y se indica la cantidad de memoria requerida en bytes. Cómo se puede observar, en esta plataforma todas las imágenes consumen el mismo espacio de memoria RAM y memoria ROM. Se determinó que esto se debe a que el mote disponible en el simulador Cooja es un dispositivo virtual y por lo tanto el proceso de compilación no crea una imagen como lo haría en un dispositivo real. Dada la importancia de evaluar el espacio necesario para el funcionamiento de los cifradores, se determinó que los resultados proporcionados por Cooja no son adecuados para esta prueba.



(a)



(b)

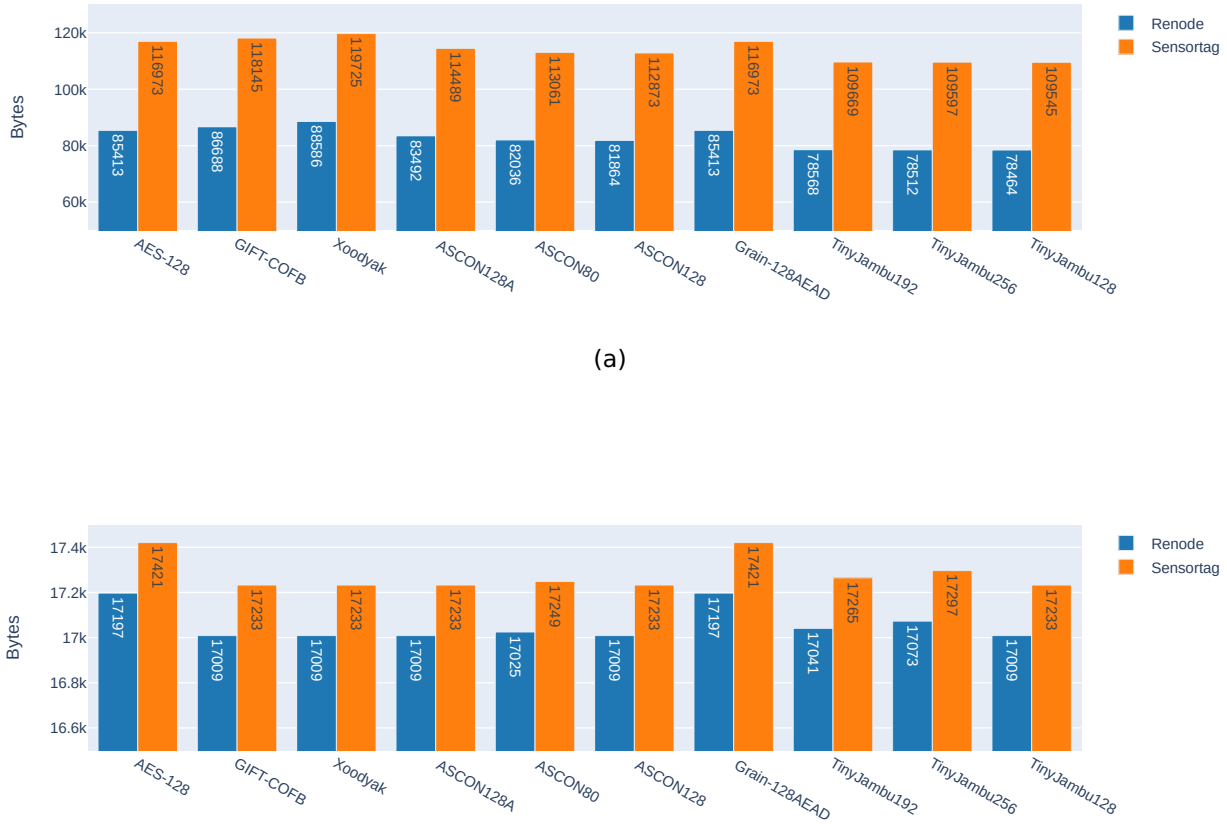
Figura 53. Uso de RAM (a) y de ROM (b) en el simulador Cooja.

6.2.1.2. Resultados en la plataforma de simulación Renode y en la plataforma física Sensortag CC2650

En la Figura 54 se muestran los resultados de la prueba tanto en la plataforma de simulación Renode, como en la plataforma física Sensortag CC2650. En la Figura 54-

a se muestra el consumo de memoria RAM para la plataforma Renode en color azul y la plataforma Sensortag CC2650 en color naranja. En la Figura 54-b se muestra el consumo de memoria ROM para ambas plataformas. En el eje x muestra cada uno de los candidatos seleccionados y en el eje y se muestra el tamaño requerido de memoria en bytes. En primera instancia se puede observar que, en contraste con los resultados arrojados por el simulador Cooja, la implementación en la plataforma Renode si muestra la variabilidad en el consumo de RAM y ROM para cada uno de los candidatos implementados. Esto se debe a que, cómo se explicó en el Capítulo 4, Renode considera en la simulación del mote CC2538 el aspecto del hardware.

En el caso de la plataforma física Sensortag CC2650, se puede observar que en general todas las implementaciones consumen más memoria en comparación con las del mote CC2538. Esto se debe a que el Sensortag CC2650 tiene una arquitectura más compleja que el CC2538, por lo que su imagen requiere más utilerías para hacerlo funcionar. Cabe hacer notar que las variaciones en el consumo de RAM y ROM arrojadas por la plataforma Renode y el Sensortag CC2650 son consistentes al cambiar de cifrador, lo cual nos permite descartar los resultados arrojados por Cooja para el resto de la discusión. En la Figura 54 se puede observar para la plataforma Sensortag CC2650 una variación entre 17.2 kBytes y 17.4 kBytes para el uso de RAM por parte de los diferentes algoritmos de cifrado. En particular, los algoritmos GIFT-COFB, Xoodyak, ASCON128, ASCON128A y TinyJambu128 son los que menos memoria consumen. Para el uso de ROM, los resultados varían entre 108.5 kBytes y 119.7 kBytes, siendo el algoritmo TinyJambu128 el que menor espacio de ROM requiere. En el caso de las imágenes para el mote CC2538 en la plataforma Renode, el uso de RAM varía entre los 17 kBytes y los 17.2 kBytes siendo los algoritmos GIFT-COFB, Xoodyak, ASCON128, ASCON128A y TinyJambu128 los que menos memoria RAM utilizan. Para el caso de la memoria ROM las implementaciones varían entre 78.4 kBytes y 88.5 kBytes, siendo de nuevo TinyJambu128 el que menor espacio de memoria requiere.



(a)

(b)

Figura 54. Espacio requerido en Bytes para el funcionamiento de los cifradores evaluados, en a) espacio en memoria RAM y b) espacio en memoria ROM para el simulador Renode y la plataforma física Sensortag.

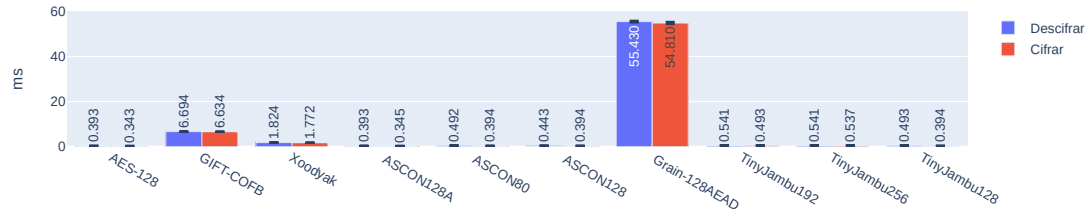
En la Figura 54 también se puede observar que el cifrado por defecto AES-128 en esta primera prueba no es el que más memoria ROM consume en ninguna de las dos plataformas. Sin embargo en el uso de memoria RAM si es de los que más consumen junto con el algoritmo de cifrado Grain-128AEAD.

6.2.2. Tiempo de cifrado/descifrado

La segunda prueba preliminar fué la de calcular el tiempo en el que se ejecuta el cifrado y el descifrado en un paquete de longitud específica. Esto nos dará una idea general de cual cifrador es el que completa la tarea de cifrado en menor tiempo, permitiendo una mayor velocidad en el proceso de aseguramiento de los datos. Además de

esto, esta prueba permitió verificar que todas las herramientas de la cama de prueba funcionaran correctamente al trabajar en conjunto.

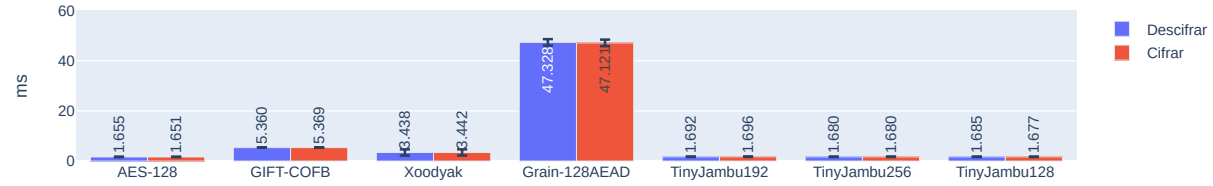
En la Figura 55 se muestran los resultados de la prueba para la plataforma de simulación Renode y en la Figura 56 para la plataforma física Sensortag CC2650 . En color rojo se muestra la media en milisegundos que se tardó en cifrar la información, mientras que en color azul se muestra la media en milisegundos que se tardó en descifrar la información. En el eje de las X se muestra cada uno de los candidatos evaluados y en el eje de las Y el tiempo en milisegundos. Para la plataforma Renode, se puede observar que el cifrador que tiene peor rendimiento es Grain-128AEAD, el cual tarda aproximadamente 55 ms en cifrar y descifrar 23 bytes. En contraste, el cifrado por defecto AES-128 tarda solo 0.39 ms en cifrar la misma información. También se puede observar que GIFT-COFB y Xoodyak tardan significativamente más tiempo que AES-128 para cifrar y descifrar los 23 bytes, mientras que la diferencia con TinyJambu no es tan grande. Finalmente, las velocidades de cifrado y descifrado con AES-128 y ASCON son equiparables.



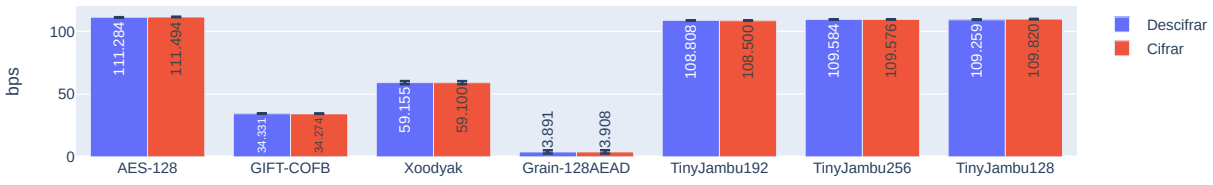
(a)

(b)

Figura 55. Velocidad de cifrado y descifrado en milisegundos (a) y en bps (b) de los algoritmos seleccionados en el simulador Renode.



(a)



(b)

Figura 56. Velocidad de cifrado y descifrado en milisegundos (a) y en bps (b) de los algoritmos seleccionados en la plataforma física Sensortag CC2650.

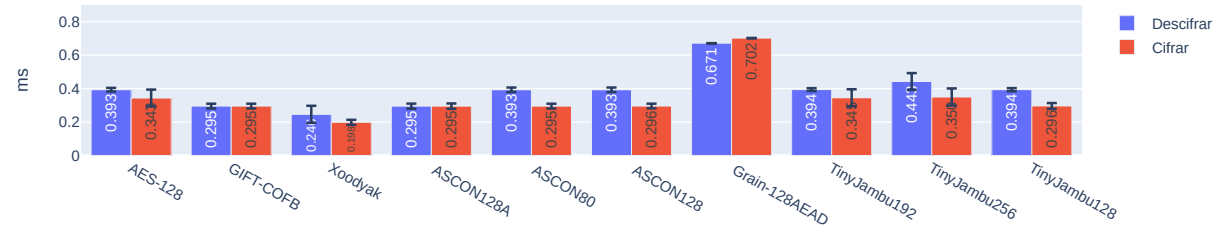
En el caso de la plataforma física Sensortag CC2650, Grain-128AEAD también es el que más se tarda en cifrar y descifrar la información utilizando 47ms aproximadamente. En contraste, AES-128 y TinyJambu tardan alrededor de 1.6 ms, siendo los más rápidos de los algoritmos evaluados. Cabe resaltar que para esta prueba en la plataforma física Sensortag CC2650 se descartaron los algoritmos ASCON debido a problemas de estabilidad con el sistema operativo al momento de ser implementados.

Al comparar la Figura 55 y la Figura 56, se puede observar un comportamiento similar de los resultados obtenidos con la plataforma física Sensortag CC2650 y la plataforma de simulación Renode. Las diferencias en los tiempos promedio calculados en ambas plataformas se deben a que se está usando el mote CC2538 en la simulación, el cual posee capacidades diferentes a la plataforma Sensortag CC2650.

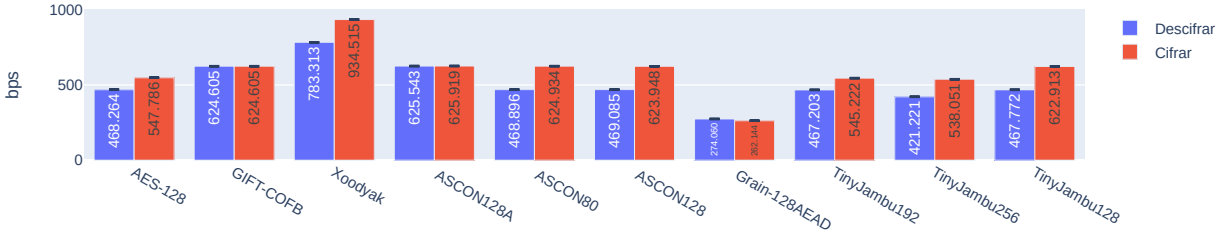
6.2.2.1. Algoritmos de cifrado optimizados

Las implementaciones de los algoritmos de cifrado que se utilizaron en la prueba anterior fueron las versiones de referencia que se enviaron para su evaluación al concurso del NIST. Sin embargo, debido a que en general estos algoritmos presentaron un rendimiento inferior que el rendimiento del cifrado por defecto (AES-128), se consideró pertinente implementar las versiones optimizadas para microprocesadores ARM de estos cifradores. Estas implementaciones optimizadas utilizan diversos métodos como tablas precalculadas y codificación en ensamblador para mejorar el rendimiento de algoritmos en arquitecturas de hardware específicas. Las versiones optimizadas de estos algoritmos fueron tomadas de fuente, que es un proyecto del NIST para probar el rendimiento de los algoritmos de cifrado del concurso en diferentes tipos de hardware.

Una vez implementadas las versiones optimizadas de los algoritmos, se obtuvieron los resultados de tiempo de cifrado/descifrado promedio mostrados en la Figura **??**.

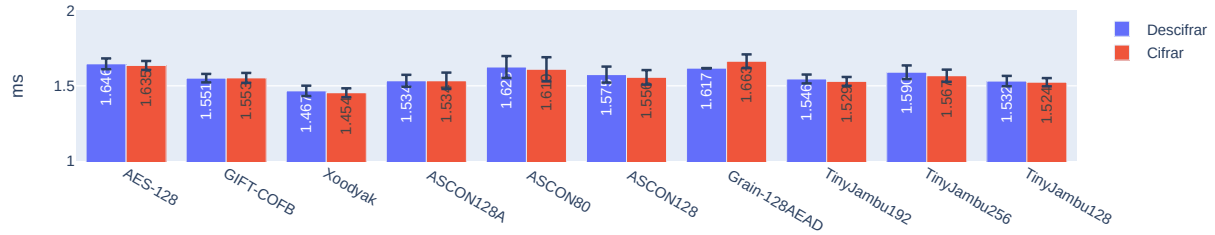


(a)

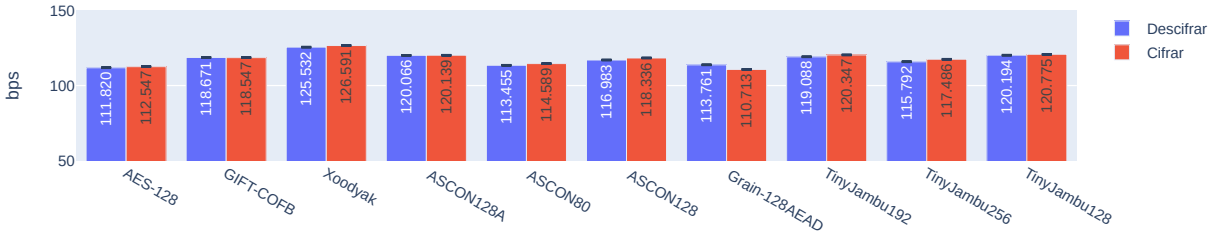


(b)

Figura 57. Tiempo de cifrado y descifrado en milisegundos (a) y en bps (b) utilizando algoritmos optimizados en el simulador Renode.



(a)



(b)

Figura 58. Tiempo de cifrado y descifrado en milisegundos (a) y en bps (b) utilizando algoritmos optimizados en la plataforma física Sensortag CC2650.

En las Figuras 57 y58 se muestran los resultados de los cifradores ligeros con la versiones optimizadas. Lo primero que se puede notar es que el rendimiento mejoró considerablemente para todos los algoritmos en ambas plataformas, incluso superando al cifrado por defecto AES-128. Para la plataforma de simulación Renode, se puede observar que Grain-128AEAD sigue siendo el algoritmo al que más tiempo le toma cifrar la información, sin embargo mejoró desde 45 ms hasta los 0.7 ms en tiempo de cifrado. El algoritmo Xoodyak se destaca como el que hace su operación de cifrado y descifrado más rápidamente con 0.19ms y 0.24ms respectivamente. Además de esto, se puede observar que todos los algoritmos de cifrado en alguna de sus variantes superan al algoritmo por defecto AES-128 (con excepción del algoritmo Grain-128AEAD).

Para el caso de la plataforma Sensortag CC2650, Grain-128AEAD y AES-128 tuvieron un desempeño equiparable. En contraste, los otros algoritmos de cifrado ligero mostraron un mejor desempeño que AES-128, siendo Xoodyak el más rápido. Otra cosa que se puede notar es que en general todos los algoritmos fueron más lentos

en la plataforma física que en la plataforma de simulación. Esto se atribuye a que el microprocesador del mote utilizado en la plataforma de simulación tiene un mejor rendimiento debido a que no fue diseñado para funcionar con muy baja potencia, como si lo está el microprocesador CC2650 utilizado en el Sensortag (YiKai Chen, 2015).

Dado que las implementaciones de los cifradores optimizados tuvieron un mejor rendimiento que las implementaciones de referencia, se decidió utilizar las implementaciones optimizadas para todas las pruebas siguientes.

6.2.3. Resultados de la velocidad del handshake

Como se explicó en el Capítulo 5, esta prueba consiste en medir el tiempo exacto en el cual se realiza un handshake del protocolo DTLS variando el algoritmo de cifrado. En particular, en esta prueba se obtuvieron 2 métricas:

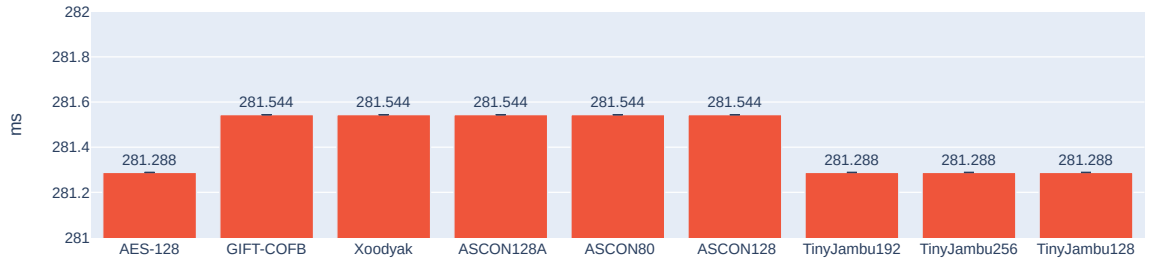
- El tiempo total en el que se llevó a cabo un handshake.
- El tiempo que le toma a un nodo desde que es encendido hasta que esté en una sesión segura con el cliente.

Para la realización de la prueba se consideraron los escenarios de uno y dos saltos con las implementaciones optimizadas para microprocesadores ARM de los algoritmos cifrado:

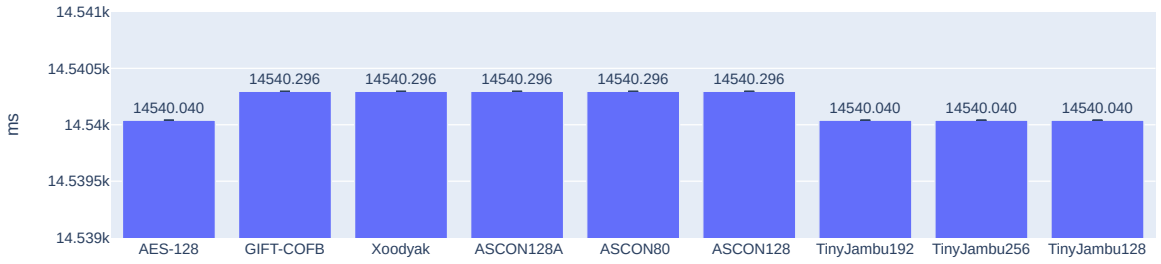
- AES-128
- GIFT-COFB
- Xoodyak
- ASCON128
- ASCON128A
- ASCON80
- Grain-128AEAD
- TinyJambu128
- TinyJambu192
- TinyJambu256

6.2.3.1. Resultados en la plataforma de simulación Cooja

En la Figura 59 se muestran los resultados de la prueba utilizando la plataforma de simulación Cooja en el escenario de un solo salto. La gráfica en color azul muestra toda la duración del enlace seguro mientras que la gráfica en color rojo muestra la duración del handshake. El eje x muestra cada uno de los candidatos evaluados y el eje y muestra el tiempo promedio en milisegundos para cada métrica.



(a)



(b)

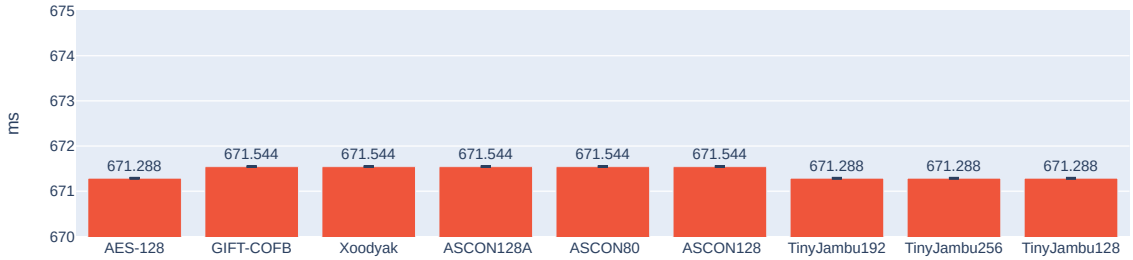
Figura 59. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en el simulador Cooja en 1 salto.

Lo primero que se observa en los resultados es que existe una clara diferencia entre AES-128 y las diferentes versiones de TinyJambu en comparación con los otros cifradores. Esto puede deberse a la diferencia en overhead que producen cada uno de los cifradores sobre el texto plano tal cual se describe en capítulos anteriores. Además,

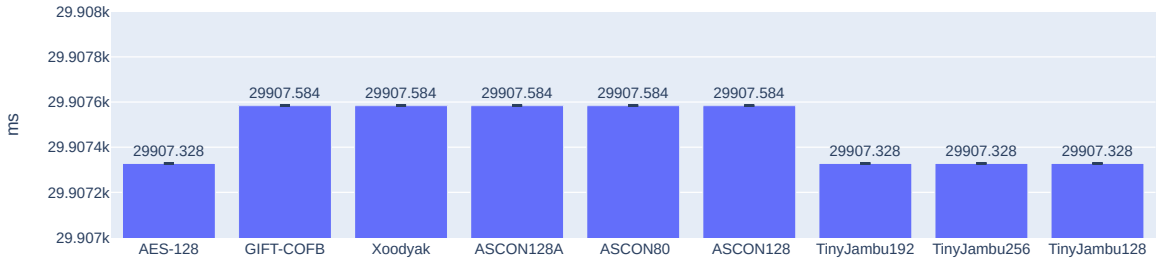
se puede observar que en esta implementación el handshake ocupa aproximadamente el 1.93 % del total del tiempo requerido para tener un enlace seguro.

Una cosa que se notó en esta implementación fue la falta de variabilidad en los tiempos arrojados por el simulador Cooja. Este comportamiento no refleja lo que se esperaría en una implementación física. Una explicación se puede encontrar en la naturaleza del nodo emulado que se estaba utilizando, el cual (como se explicó en el Capítulo 3) al ser un dispositivo virtual, no se comporta como se comportaría un dispositivo real.

En la Figura 60 se muestran los resultados de la prueba para el escenario de dos saltos. Una vez más se puede notar la falta de variabilidad en los resultados. Nuevamente, este no es un comportamiento que se observaría en un despliegue físico. Debido a los resultados obtenidos con la plataforma Cooja en esta prueba y la prueba de tamaño de imágen compilada, se determinó que *Cooja no es una plataforma adecuada para implementar las evaluaciones requeridas en el presente trabajo de investigación, por lo que ya no se incluyó en las pruebas que se llevaron a cabo más adelante*. Este es un resultado importante que contribuye a la comunidad académica para una mejor selección de la plataforma de pruebas.



(a)



(b)

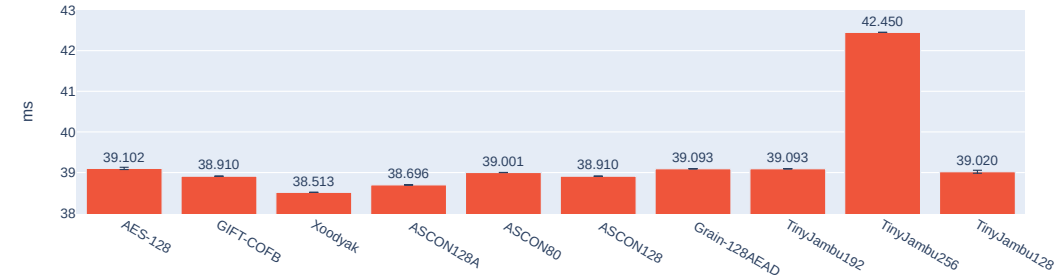
Figura 60. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en el simulador Cooja en 2 saltos.

6.2.3.2. Resultados en la plataforma de simulación Renode

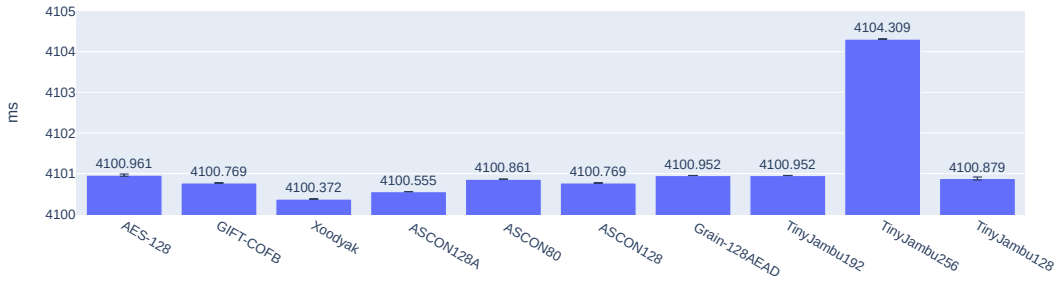
En la Figura 61 se muestran los resultados de la prueba en el escenario de un salto para la plataforma de simulación Renode. Como se puede observar, en esta plataforma sí que existen variaciones en los resultados arrojados por diferentes algoritmos de cifrado. El handshake del algoritmo TinyJambu256 es el que más tiempo se tardó en completarse con 42 ms y la implementación con el algoritmo Xoodyak el que menos tiempo le llevó completarse con 38 ms.

En la Figura 61, también se puede observar que el tiempo que toma establecer el enlace seguro está directamente relacionado con la duración del handshake. Por lo tanto, se puede suponer que todos los demás procesos que realiza la red duran

exactamente el mismo tiempo para todos los algoritmos de cifrado. Teniendo que el handshake toma aproximadamente el 10 % del tiempo para establecer todo el enlace seguro.



(a)

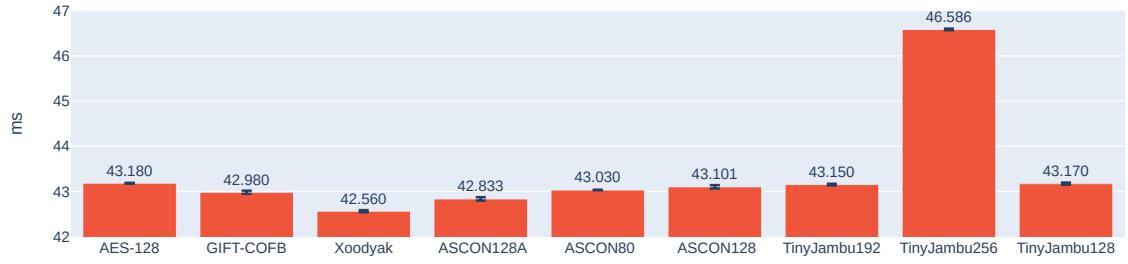


(b)

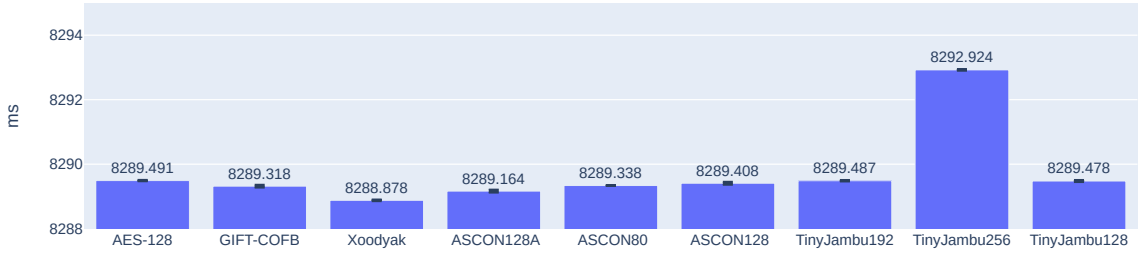
Figura 61. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en el simulador Renode en 1 salto.

En la Figura 62 se muestran los resultados de la prueba para el escenario de 2 saltos. Se puede notar que al igual que en la prueba anterior, el proceso de handshake constituye una porción de tiempo pequeña (alrededor del 5 %) del total de tiempo requerido para establecer un enlace seguro. Esto indica que lo que afecta en mayor medida al tiempo de establecimiento de enlace seguro es el número de nodos de la red y no tanto el handshake de la capa DTLS. Se vuelve a observar que el algoritmo TinyJambu256 es el que tarda más en completar el handshake. Esto puede deberse a que es el único algoritmo que ofrece una seguridad de 256bits y por lo tanto tiene que crear una clave del doble de longitud que los demás candidatos. También se vuelve a observar que la implementación del algoritmo Xoodyak es la que más rápido lleva

a cabo el handshake. En cuanto al algoritmo por defecto AES-128, se puede observar que si bien en general es más lento que los demás candidatos (a excepción de Tiny-Jambu256), esta diferencia no es tan notoria en comparación con la mayoría de los candidatos a excepción de Xoodyak.



(a)



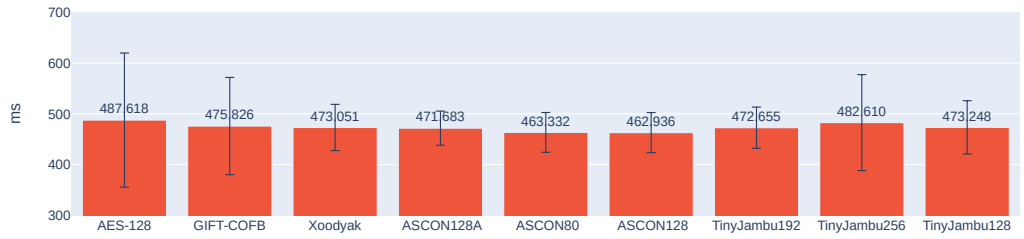
(b)

Figura 62. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en el simulador Renode en 2 saltos.

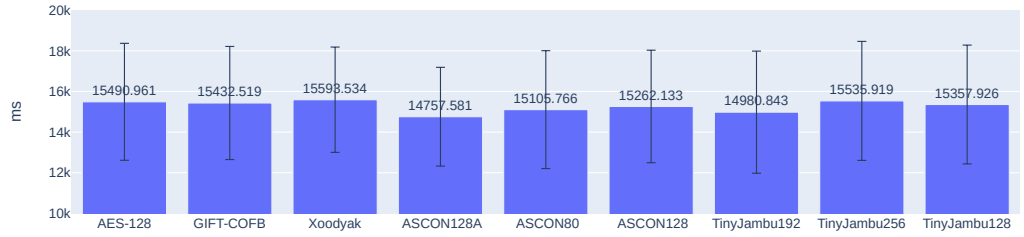
6.2.3.3. Resultados en la plataforma física Sensortag CC2650

En la Figura 63 se muestran los resultados de la prueba en el escenario de un salto para la plataforma física Sensortag CC2650. Lo primero que se observa en estos resultados es que la desviación estándar es considerablemente mayor en comparación con las implementaciones anteriores. Esto es producto de la gran variabilidad entre los resultados obtenidos en cada una de las iteraciones realizadas. Cabe notar que esta

variabilidad se puede atribuir a factores externos de la red como lo es el canal radio, lo cual es de esperar en una implementación real. Se puede observar también que el handshake en general ocupa aproximadamente el 3% de toda la conformación de la red. Otra cosa que se puede notar es que en comparación con la plataforma de simulación, tanto el tiempo del handshake como el tiempo total del enlace seguro son más lentos. Esto se debe a que el cifrado es más lento en esta plataforma y a que en el simulador el canal radio es ideal y por lo tanto todos los paquetes que salen de un nodo llegaran a su destino de manera inmediata. Cabe hacer notar que esto último no sucede en un despliegue real debido a múltiples factores que afectan al canal radio como las transmisiones e interferencias externas. Finalmente, note que si únicamente se contará con estos resultados sería difícil inferir de manera confiable con cuál algoritmo de cifrado se ejecuta más rápidamente el handshake. Esto remarca la importancia de utilizar tanto la plataforma de simulación como la plataforma física para realizar las pruebas.



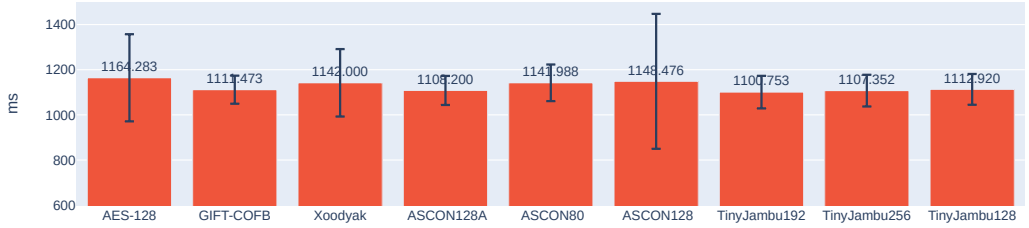
(a)



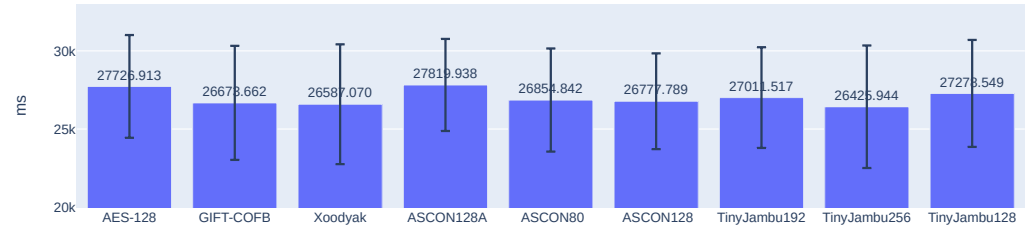
(b)

Figura 63. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en la plataforma física Sensortag en 1 salto.

En la Figura 64 se muestran los resultados de la prueba en el escenario de 2 saltos para la plataforma física Sensortag ss2650. Al igual que en los resultados de la prueba de una salto, para esta plataforma se puede notar una alta variación entre pruebas con la desviación estándar. En este caso el tiempo del enlace seguro es de aproximadamente un 170 % superior en comparación con el escenario de un salto. Por otro lado, la duración del handshake es un 230 % superior. Esto se debe a que se está utilizando un nodo como intermediario entre ambos dispositivos, el cual debe recibir el paquete de uno y transmitirlo al otro. Esto implica que en lugar de una transmisión se realizan dos transmisiones para cada paquete. También se puede notar que el handshake toma aproximadamente un 4 % de todo el enlace seguro. En este caso, a partir de los resultados de esta prueba solamente, tampoco se puede determinar qué algoritmo proporcionó el mejor tiempo para el handshake, dado que factores externos de la red aumentan la variabilidad entre las pruebas.



(a)



(b)

Figura 64. Tiempo que toma un handshake (a) y tiempo de enlace seguro (b) en la plataforma física Sensortag en 1 salto.

6.2.4. Conclusiones de pruebas preliminares

Estas pruebas fueron de mucha utilidad para corroborar que la cama de pruebas implementada funcionase correctamente. Además, nos permitió probar cada uno de los algoritmos de cifrado y corregir algunos errores en su implementación.

Además de lo anterior, se encontró que las versiones de referencia de los cifradores ligeros no eran las indicadas para una evaluación correcta de las pruebas de sobrecarga. Esto debido a su bajo rendimiento al momento de implementarlas en las plataformas base, por lo que se decidieron habilitar las versiones optimizadas para microprocesadores ARM para las demás pruebas. Por otro lado, se corroboró que el simulador Renode es apto para simulación a nivel de hardware mientras que el simulador Cooja no es compatible con estas características.

En cuanto al rendimiento general de los algoritmos de cifrado optimizado, se pudo observar por medio de la simulación con Renode que el tiempo de handshake varia al momento de cambiar de algoritmo de cifrado. En la implementación física esto no se pudo observar tan claramente, debido a la influencia de factores como el estado del canal radio y la interacción entre los dispositivos al momento de formar la red. Sin embargo, al analizar conjuntamente los resultados entregados por la plataforma Renode y la plataforma física Sensortag CC2650, se puede inferir que algoritmos como Xoodyak y GIFT-COFB ofrecen un tiempo menor de handshake que AES-128.

Finalmente, cabe remarcar la importancia de haber realizado las pruebas tanto en una plataforma de simulación como en una plataforma física, esto con el fin de obtener conclusiones relevantes para un posible despliegue en el mundo real de estos cifradores.

6.3. Resultados de las pruebas de sobrecarga generales

El objetivo principal de las pruebas de sobrecarga generales fue encontrar cómo afecta la implementación de los cifradores ligeros a una implementación realista de una red IoMT cuando se consideran métricas de red. Como se explicó en el Capítulo 5, se consideraron dos escenarios para la realización de estas pruebas:

- Escenario de red en malla.
- Escenario de red multisalto.

Las siguientes métricas de red consideradas para esta evaluación son:

- End to End delay.
- Tiempo de procesamiento.
- Jitter.
- Latencia en paquetes solicitud-respuesta.

Las cuales fueron explicadas en el Capítulo 5. Para cada escenario de evaluación se varió el empaquetado del streaming de oximetría considerando la transmisión de 1 muestra, 6 muestras y 9 muestras por paquete.

Para la evaluación se recolectaron métricas de los dos escenarios con diferentes empaquetados para oximetría utilizando:

- AES-128
- Xoodyak
- GIFT-COFB
- TinyJambu128
- TinyJambu192
- TinyJambu256

Cabe destacar que, como se explicó en el Capítulo 5, las tres variantes del algoritmo ASCON y el algoritmo Grain-128AEAD se descartaron en este conjunto de pruebas debido a que presentaron problemas de compilación en la implementación propuesta.

A continuación se muestran los resultados que se obtuvieron en estas pruebas.

6.3.1. Resultados en el escenario de malla

En la figura 65-a se muestran los resultados de End to End delay y en la Figura 65-b se muestran los resultados del Jitter del streaming del nodo de oximetría para el escenario de malla. Las barras de color azul pertenecen al empaquetado de 1 muestra, las barras en color rojo pertenecen al empaquetado de 6 muestras y las barras de color verde al empaquetado de 9 muestras. En el eje x se enlistan los algoritmos utilizados en cada implementación y en el eje y se muestran las métricas recopiladas para cada algoritmo.

Lo primero que se puede notar en esta prueba es la variación en el End-to-End delay al variar el tipo de empaquetado. Esto se debe a que al enviar una muestra del streaming de oximetría por paquete 6LoWPAN se genera una sobrecarga significativa en la red, tal cual se explicó en el Capítulo 5. Básicamente, al enviar un paquete por muestra se tendrá más actividad en las capas de acceso al medio (MAC) y la capa física (PHY). Debido a que 6LoWPAN utiliza la implementación de CSMA-CA del estándar IEEE 802.15.4, al tener más paquetes en la capa MAC habrá más contención para realizar la transmisión de cada paquete individual, lo cual se refleja directamente en el End to End delay. Otro detalle a observar en la Figura 65 es que en general todos los nodos mostraron el mismo comportamiento en cuanto a medias y desviaciones estándar del End to End delay y Jitter. Una interpretación es que se tuvieron condiciones de canal similares durante la realización de las pruebas.

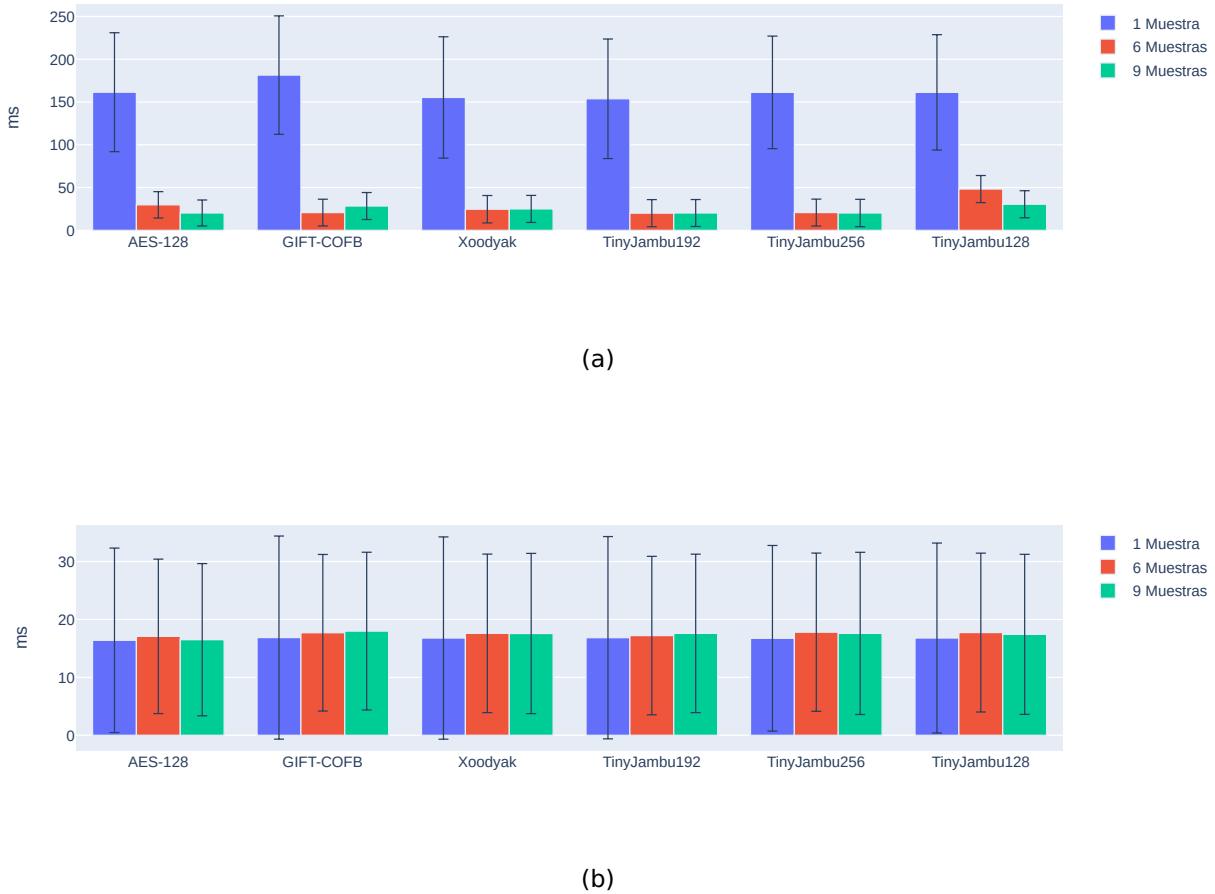


Figura 65. End to End delay (a) y Jitter (b) estimado en el nodo de oximetría para el escenario de malla.

En la Figura 66 se muestran los resultados obtenidos de la latencia para transmitir los paquetes generados por el nodo de temperatura. Lo primero que se observa es que la transmisión de los paquetes de temperatura tiene una latencia significativa. Si bien la latencia es menor cuando el streaming de oximetría usa empaquetados de 6 y 9 paquetes, la reducción en la latencia no es tan grande como el cambio observado en el End to End delay del streaming de oximetría al utilizar estos empaquetados. Considerando que:

- El streaming de oximetría produce una gran actividad en las capas MAC y PHY.
- El nodo termómetro genera paquetes a una tasa de un paquete por segundo.

La transmisión de cada paquete de oximetría encuentra una alta ocupación de

canal y una alta contención a nivel capa MAC, lo cual explica el comportamiento observado en la métrica de latencia. Esto se puede corroborar también observando que en 5 de las 7 implementaciones, cuando el nodo de oximetría transmite 1 muestra a la vez la latencia de los paquetes de temperatura es mayor.

En cuanto a los algoritmos de cifrado, podemos observar que al analizar en conjunto las métricas de End to End delay y latencia, Xoodyak se destaca del resto al ofrecer una latencia menor para la transmisión de los paquetes de temperatura aun y cuando se tiene mayor tráfico de red en el escenario de empaquetamiento de una muestra de streaming de oximetría por paquete. En contraste, para esta prueba AES-128 es el que muestra peor desempeño en la métrica de latencia aun y cuando las condiciones de operación de la red fueron similares para estas pruebas, como se evidencia al comparar el End to End delay y el Jitter obtenido para Xoodyak y AES-18. Por lo tanto, se puede concluir que en redes IoMT en donde los nodos generan variables que derivan en diferencias significativas de las tasas de transmisión de paquete de cada nodo (p.ej. 60 paquetes/segundo vs 1 paquete/segundo), el utilizar un algoritmo de cifrado más eficiente a nivel de hardware puede ayudar a disminuir el impacto en las métricas de desempeño de toda la red.

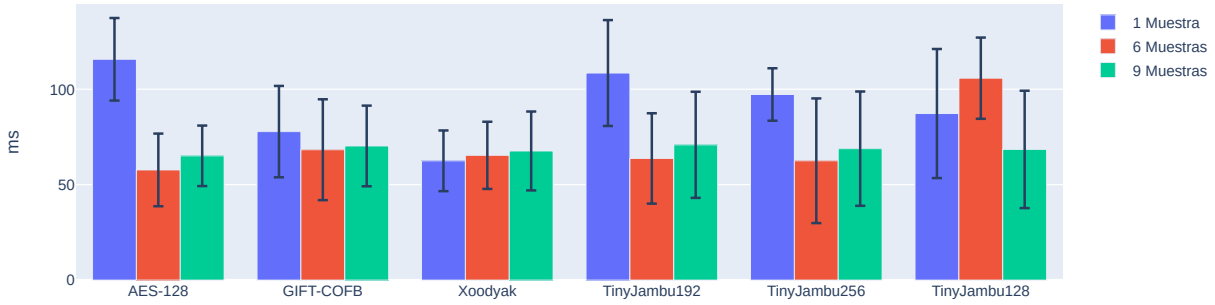


Figura 66. Latencia estimada para el nodo de temperatura en el escenario de malla.

En la Figura 67 se muestran los resultados del tiempo de procesamiento que tomó cada implementación desde que se puso la carga útil en la cola hasta que se transmitió el mensaje desde el nodo de oximetría. En este caso se ordenaron los algoritmos desde el que menos tardó hasta el que tardó más. Además, se agruparon por empa-

quetado, mostrando 1 muestra a la izquierda, 6 muestras en el medio y 9 muestras a la derecha. Lo primero que se puede observar es que el tiempo de procesamiento cuando no se aplica ningún cifrado es considerablemente menor (alrededor de un 6 %) en comparación de cuando se aplica un algoritmo de cifrado.

Lo siguiente que se observa es que para todos los tipos de empaquetados la implementación con el algoritmo Xoodyak se destaca como la que menos tardó en procesar el paquete antes de mandarlo. Esto confirma lo que se observó en la prueba de velocidad de cifrado de las pruebas preliminares. Por otro lado, se observa que los algoritmos se comportan de manera distinta al aumentar la cantidad de bytes de cada paquete. Como es de esperar el tiempo de procesado tiende a aumentar entre más aumenta la cantidad de carga útil en los paquetes. Sin embargo, algunos algoritmos aumentan a un ritmo más lento que otros. Este es el caso del algoritmo GIFT-COFB, el cual al enviar 9 muestras a la vez tiene mejor rendimiento en comparación con los demás a excepción del algoritmo Xoodyak. Otra cosa que se pudo notar fue que la implementación con el algoritmo de cifrado por defecto AES-128 es la que tardó más en procesar el paquete, lo cual es un comportamiento esperado al estar compitiendo con algoritmos de cifrado ligero.

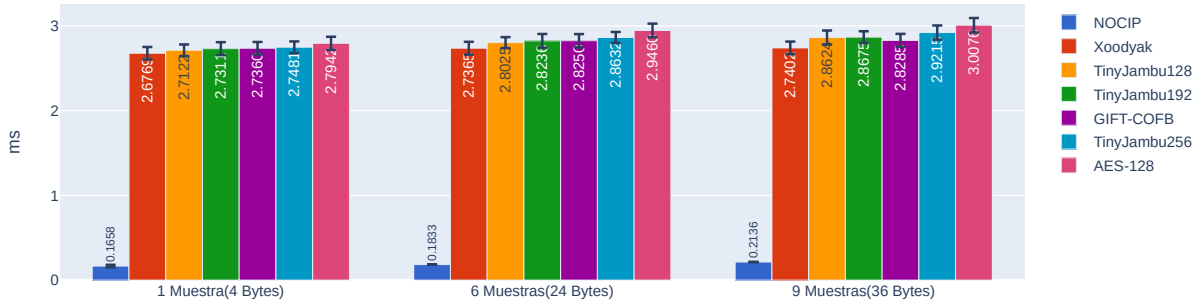


Figura 67. Tiempo de procesamiento del paquete estimado en el nodo de oximetría para el escenario de malla.

6.3.2. Resultados en el escenario multisalto

En la Figura 68 se muestran los resultados obtenidos en el End to End delay y el Jitter observados en el nodo de oximetría en un escenario multisalto. Lo primero que se puede observar es que la desviación estándar del End to End delay en comparación con el escenario de malla tiende a ser mayor. Esto hace suponer que la variación entre llegadas de paquetes tiende a ser mayor para este escenario. Dicha observación se refuerza al verificar que el valor medio del Jitter también tiende a ser mayor en comparación con el escenario de malla.

Por otro lado se vuelve a observar el fenómeno en donde al transmitir con un empaquetado de una muestra a la vez el End to End delay se dispara aproximadamente un 700 % en comparación con los empaquetados de 6 y 9 muestras a la vez. En cuanto a la diferencia entre utilizar un algoritmo de cifrado ligero u otro, solo con las métricas de End to End delay y Jitter obtenidas en esta prueba no es posible llegar a resultados concluyentes. Esto debido a que las grandes variaciones en las medias y varianzas observadas en la Figura 67 son indicativo de que se tuvieron condiciones diferentes de canal radio para cada algoritmo e iteración.

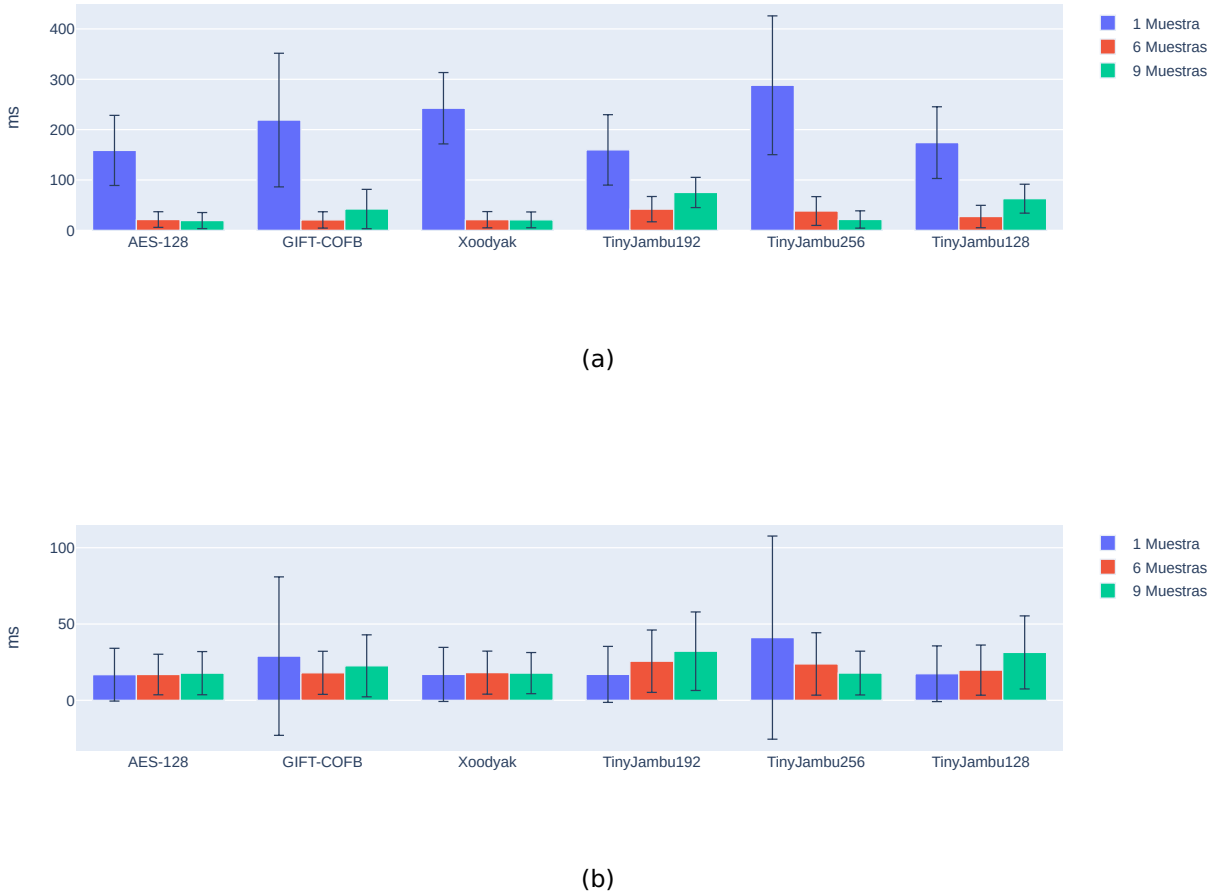


Figura 68. End to End delay (a) y Jitter (b) estimado en el nodo de oximetría para el escenario multisalto.

En la Figura 70 se muestran los resultados de la latencia estimada para el nodo de temperatura en este escenario. Para este escenario se pudo observar que la latencia de los paquetes de temperatura tendió a disminuir ligeramente en comparación con la latencia en un escenario de malla. Esto se atribuye a que el nodo de temperatura era el nodo que se encontraba más cerca del cliente, por lo que solo estaba a un salto de distancia del mismo.

En cuanto a la diferencia entre los algoritmos de cifrado, primero cabe hacer notar nuevamente que se tuvieron condiciones diferentes de canal para cada algoritmo. Esto se refleja en la variabilidad de las métricas de End to End Delay y Jitter mostradas en la Figura 68. En esta figura se puede ver que para el empaquetado de 1 muestra de streaming por paquete, AES-128 y TinyJambu192 tuvieron el menor End to End delay.

Sin embargo esto no se debe al efecto que dichos algoritmos pudieran tener en esta métrica, sino a las condiciones de canal cuando las pruebas fueron realizadas. Esto se puede constatar al analizar las métricas de End to End delay y latencia en conjunto. En particular se puede observar que a pesar de que en las pruebas de Xoodyak y GIFTCOFB no se tuvo el menor End to End delay para el streaming de oximetría con el empaquetado de 1 muestra (i.e. las condiciones del canal eran adversas en comparación con AES-128 y TinyJambo192), ambos cifradores proveen una menor latencia para la transmisión de paquetes de temperatura comparados con AES-128.

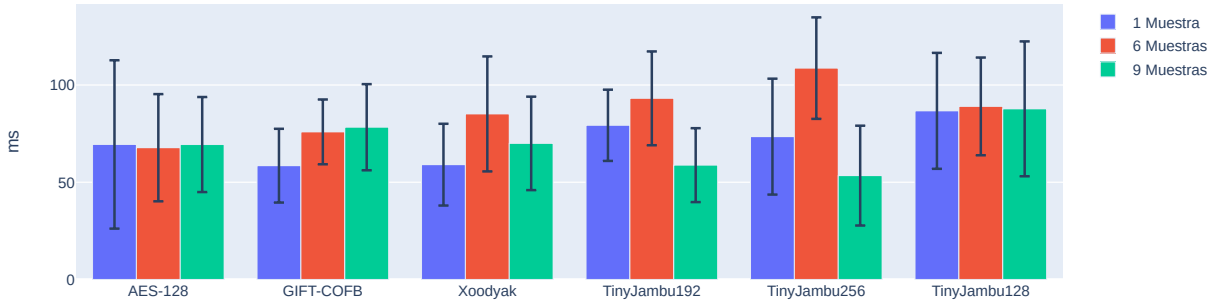


Figura 69. Latencia estimada para el nodo de temperatura en el escenario multisalto.

En la Figura 70 se muestran los resultados obtenidos del tiempo de procesamiento en el escenario multisalto. Al igual que en la gráfica presentada en el escenario de malla, se ordenaron los algoritmos desde el que menos tiempo tardó hasta el que tardó más. Además, se agruparon por empaquetado, mostrando 1 muestra a la izquierda, 6 muestras en el medio y 9 muestras a la derecha, el eje de las y está en milisegundos.

Del mismo modo que en el escenario de malla se puede observar que la implementación sin cifrado es considerablemente más rápida en comparación con las implementaciones en donde se implementó el cifrado. Por otro lado se puede observar que de nuevo la implementación con el algoritmo de cifrado Xoodyak se destaca como la que más rápido procesa los paquetes. En contraste, el algoritmo de cifrado por defecto AES-128 es el que más tardo para los empaquetados de 6 y 9 muestras a la vez y TinyJambu256 el que más tardó en paquetes de una muestra. Sin embargo cabe recalcar que el algoritmo TinyJambu256 ofrece una seguridad de 256 bits mientras

que AES-128 solo de 128 bits.

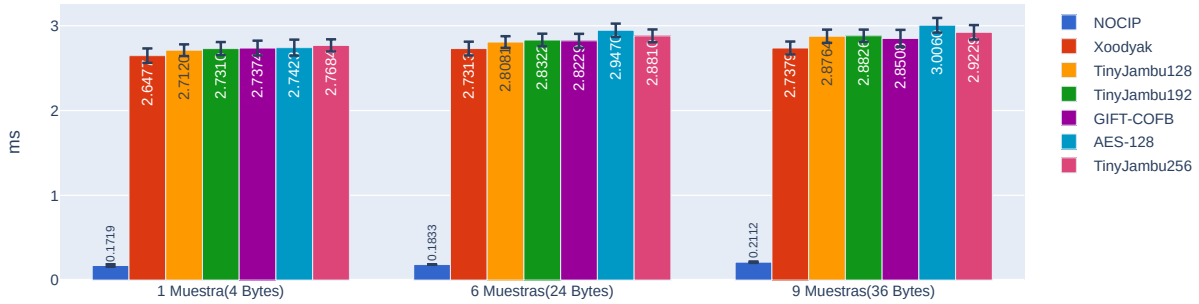


Figura 70. Tiempo de procesamiento del paquete estimado en el nodo de oximetría para el escenario multisalto.

6.3.3. Conclusiones

Con base en los resultados obtenidos en estas pruebas se encontró que el implementar seguridad en la red sí que conlleva un costo extra en el tiempo de procesamiento del paquete. Sin embargo en una perspectiva de red este retraso en el tiempo de procesamiento puede ser opacado por otros factores en el despliegue de una red real. Estos factores van desde cómo se implementan los servicios (como se observó al variar el empaquetado del streaming) hasta qué tan saturado está el canal radio. Esto se observó en las pruebas de End to End delay y Jitter del streaming de oximetría. Sin embargo, cuando se tiene una red con transmisión de datos con diferencias significativas en las tasas de generación de paquetes (p.ej. streaming de oximetría vs temperatura), la utilización de diferentes algoritmos de cifrado en la red si mostró un efecto en el desempeño de la métrica de latencia para el flujo de información con tasa de generación de paquetes más baja.

En cuanto a la evaluación de los algoritmos de cifrado se demostró que el algoritmo Xoodyak es el que destaca en la prueba de tiempo de procesamiento. Esto se refleja en las métricas obtenidas para el escenario de malla y el escenario multisalto. Además, la utilización de este algoritmo provee el mejor desempeño en la métrica de latencia para la transmisión de información de temperatura, aun y cuando el comportamiento

de la red causado por el streaming de oximetría no era el más favorable. Esto a su vez demuestra que el tiempo de procesamiento está directamente relacionado con la velocidad de cifrado del algoritmo utilizado.

6.4. Resultados de las pruebas de consumo de energía

Como se explicó en el Capítulo 5, como parte de la evaluación se realizaron las siguientes dos pruebas para estimar el consumo de energía para los diferentes cifradores:

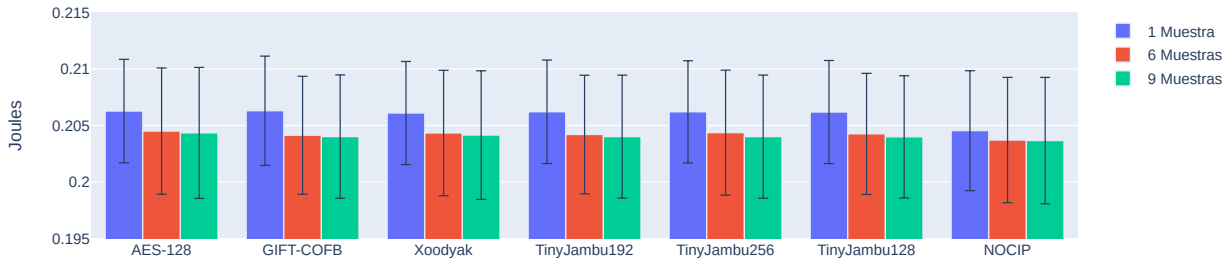
- La primera prueba consiste en una estimación indirecta del consumo, en donde con la ayuda del módulo Energest de Contiki-NG se obtiene el tiempo en centésimas de segundos que estuvo encendido cada uno de los diferentes componentes de la plataforma. Una vez obtenido este tiempo se calcula el consumo de energía en joules. Cabe resaltar que esta prueba se llevó a cabo al mismo tiempo que las pruebas de sobrecarga por lo que se realizó en ambos escenarios.
- La segunda prueba consistió en una estimación directa del consumo de energía, en donde se obtiene la curva de descarga para el voltaje de una misma batería que alimenta a un nodo de la red.

6.4.1. Resultados de consumo de energía utilizando el módulo energest

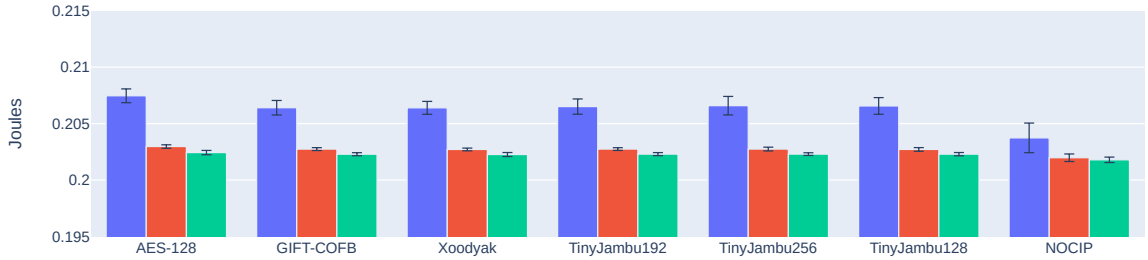
En la Figura 71 se muestran los resultados de la estimación de consumo de energía para el escenario de malla. La Figura 71-a corresponde a el nodo cliente y la Figura 71-b corresponde al nodo de oximetro. De color azul se muestran los resultados para un empaquetado de 1 muestra a la vez, en colo rojo el empaquetado de 6 muestras a la vez y el colo verde el empaquetado de 9 muestras a la vez. En el eje x está enlistado el algoritmo de cifrado de cada implementación y en el eje de las y el consumo estimado en joules.

Lo primero que se observó en estos resultados fue que utilizando el empaquetado de una muestra los nodos consumen más energía (alrededor de 1 % más) en comparación con los demás empaquetados. Esto se atribuye a que se generan más paquetes,

lo que conlleva a un coste extra de energía por cada paquete generado. Así mismo, se observa que al aumentar el número de muestras que se transmiten a la vez y disminuir el número de paquetes que se transmiten disminuye en el consumo de la energía. Sin embargo, la diferencia entre el consumo de energía entre empaquetados sigue siendo pequeño para que suponga un problema para esta implementación en particular. En parte esto se debe a que en esta implementación no se habilitaron ciclos de trabajo para el radio, el cual es el módulo que más energía consume. Sin embargo, en una implementación optimizada esto podría considerarse una diferencia importante.



(a)

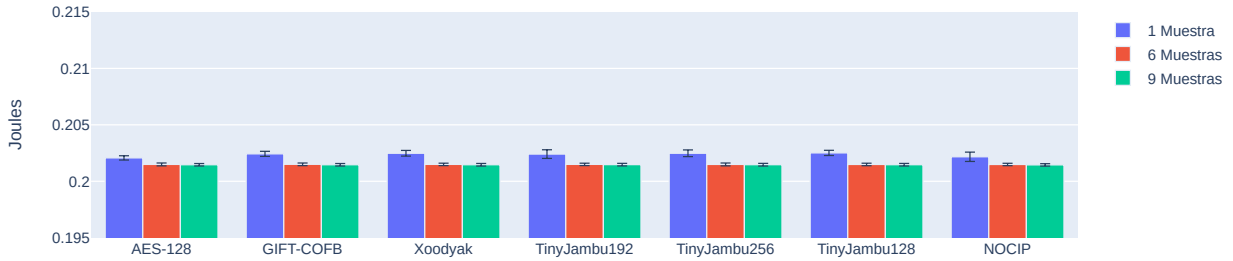


(b)

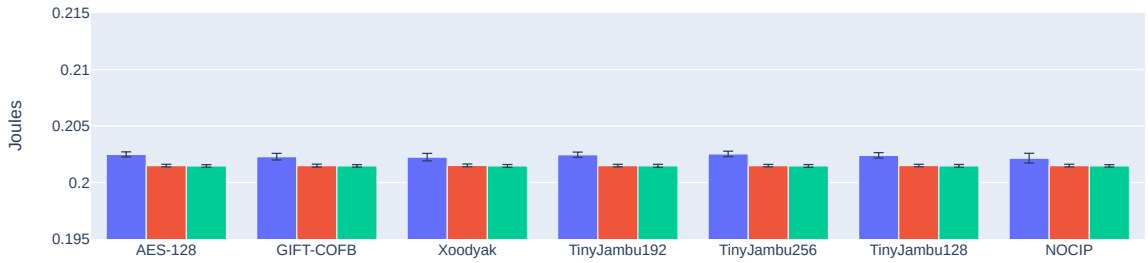
Figura 71. Consumo de energía estimado estimado en el nodo cliente(a) y el nodo de oximetría(a) para el escenario de malla utilizando el módulo energest.

En la Figura 72 se muestra en el mismo formato de gráfica el consumo de energía de los nodos termómetro y baumanómetro. Lo que se puede observar en esta gráfica es un aumento en el consumo de energía para estos nodos cuando el streaming de oxi-

metría utiliza un empaquetado de una muestra. Esto se atribuye a que dado que están en una red de malla, al menos una parte del encabezado de todos los paquetes que se transmiten por el canal radio es procesado por todos los nodos (cual corresponde al funcionamiento de las capas MAC y de red de 6LoWPAN), reflejándose directamente en su consumo de energía.



(a)

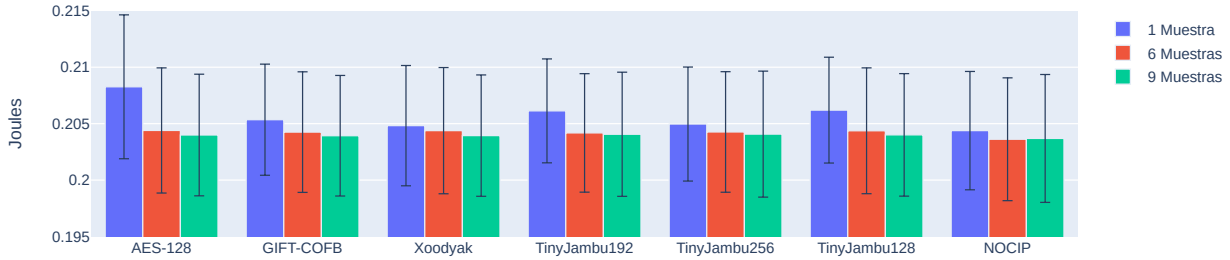


(b)

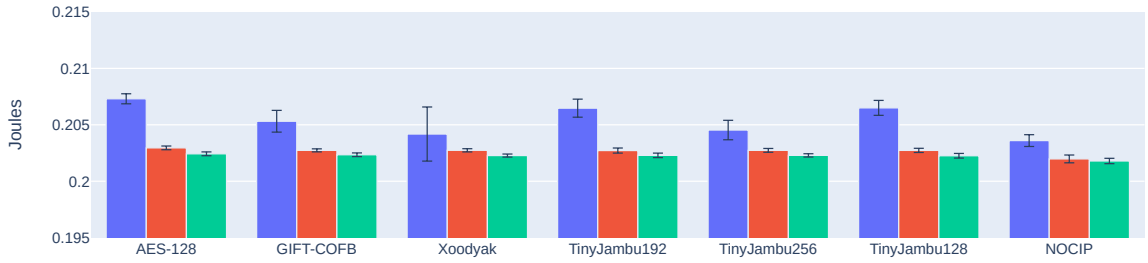
Figura 72. Consumo de energía estimado estimado en el nodo baumanómetro y el nodo de temperatura para el escenario de malla utilizando el módulo Energest.

En la Figura73 se muestra en el mismo formato de gráfica el consumo de energía que se observó en el nodo cliente y el nodo oxímetro para el escenario de multisalto. Se observa que se vuelve a repetir el fenómeno de un aumento en el consumo de energía para el empaquetado de una muestra a la vez, lo que se atribuye a las mismas razones explicadas anteriormente. Lo que sí es destacable es que, en comparación con el escenario de malla, el consumo de energía aumentó alrededor del 2 % en general.

Esto se puede atribuir al hecho de que los nodos se encuentran en una estructura de red más compleja, por lo que necesitan hacer más procesamiento para transmitir información. Sin embargo, esto no se pudo observar en la prueba de duración de procesamiento.



(a)

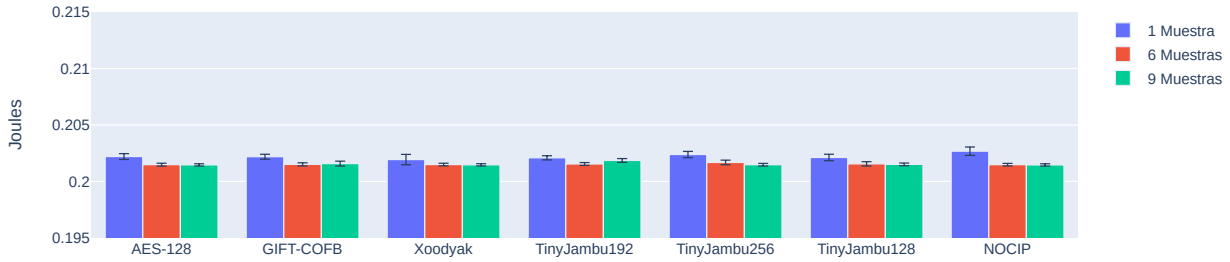


(b)

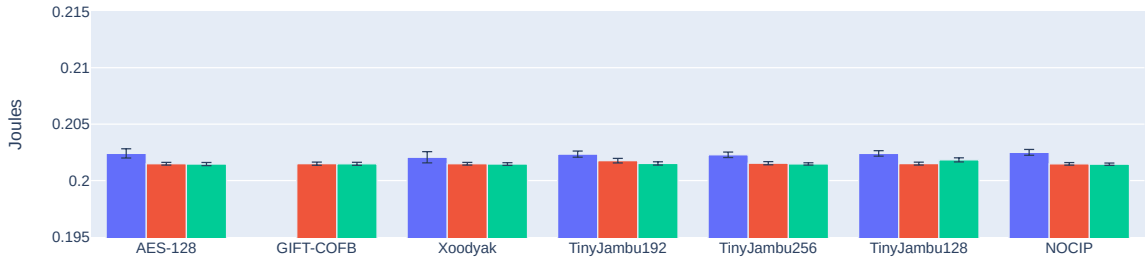
Figura 73. Consumo de energía estimado estimado en el nodo cliente(a) y el nodo de oximetría(a) para el escenario multisalto utilizando el módulo energest.

En la Figura 74 se muestran los resultados del consumo de energía para los nodos de termómetro y baumanómetro para este escenario. Se observa una gran similitud en el consumo de energía en general en comparación con la prueba en el escenario de malla. Por lo tanto se puede intuir que para esta aplicación en específico la arquitectura de red no determina un consumo de energía diferente. Además de esto, se vuelve a observar el fenómeno de consumir más energía en un escenario en donde hay más actividad en la red (streaming de 1 muestra a la vez), aunque esta actividad no esté

directamente relacionada con el servicio del nodo.



(a)



(b)

Figura 74. Consumo de energía estimado estimado en el nodo baumanómetro y el nodo de temperatura para el escenario de malla utilizando el módulo Energest.

6.4.2. Resultados de consumo de potencia evaluando la curva de descarga

Para la prueba de consumo directa se buscó observar cómo cambia la curva de descarga de una batería recargable al implementar diferentes algoritmos de cifrado ligero. Debido al corto periodo de tiempo que se tenía para realizar esta prueba solo se evaluaron 2 algoritmos de cifrado ligero: Xoodyak y TinyJambu128. Además, se probó también la implementación sin cifrado para ver si existía alguna diferencia notable. Esta prueba se realizó considerando el empaquetado de streaming de oximetría de 1 muestra y 9 muestras por paquete.

En la Figura 75 se muestra la gráfica de caída de voltaje de cada una de las imple-

mentaciones evaluadas en la prueba. En el eje x se muestran los segundos transcurridos a lo largo de la prueba y en el eje y los valores en voltios que tenía la batería en cada momento. Como se puede observar a lo largo de las 3 horas que duró la prueba no se encontraron diferencias significativas en las curvas obtenidas con las diferentes configuraciones. Además de esto, se puede observar también que la implementación sin cifrado está a la par en caída de voltaje con las implementaciones que incluyen cifrado. Por lo que para esta prueba en particular no se pudo encontrar una relación clara entre el consumo de potencia y la implementación de seguridad.

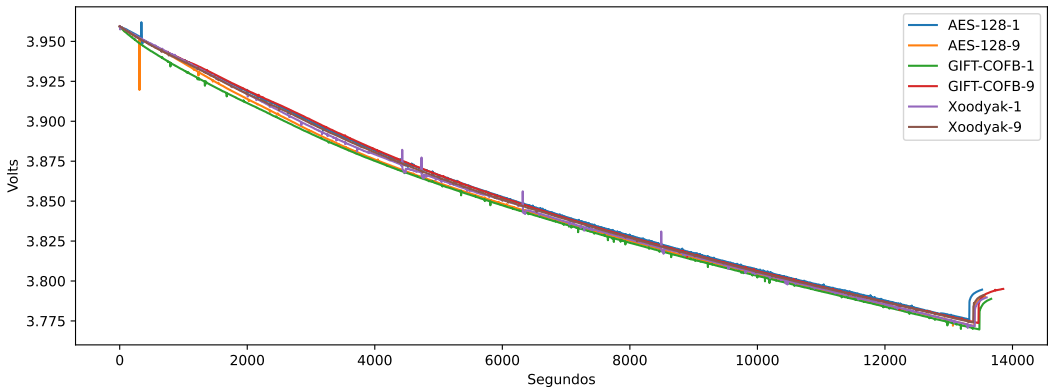


Figura 75. Caída de voltaje observado en la batería del nodo de oximetría para los empaquetados 1 y 6 usando los algoritmos evaluados.

6.5. Conclusiones

Con base en los resultados obtenidos en las pruebas de consumo de potencias se puede concluir que, aunque se espera que la implementación de seguridad suponga un aumento en el consumo energético de la red y de cada uno de los nodos dado el aumento en el procesamiento que esto supone, existen otros factores que afectan en mayor medida este aspecto. En particular consideramos que para la implementación realizada para la evaluación del consumo de energía, las posibles diferencias en el consumo de energía ofrecido por los diferente cifradores quedaron “enmascaradas” por el consumo que representa la transmisión y recepción de paquetes cuando no se utilizan ciclos de trabajo en los nodos.

Capítulo 7. Conclusiones, aportes y trabajo futuro

En este capítulo se describirán todas las conclusiones y aportaciones que se llevaron a cabo a lo largo del desarrollo de este trabajo de tesis, así como el posible trabajo futuro que se podría llegar a realizar en base al mismo.

7.1. Conclusiones

En este trabajo de investigación se realizó una evaluación sobre cómo afecta la implementación de algoritmos de cifrado ligero al comportamiento de una red IoMT. En la evaluación se seleccionaron 5 de los algoritmos que al momento de realizar este trabajo, estaban bajo evaluación del NIST para su posible adopción como estándar de cifrado ligero. Además se incluyó el cifrador AES-128 que es un estándar de facto para aplicaciones de seguridad y cifrado con DTLS.

Para el diseño de la cama de pruebas y su implementación se consideraron tecnologías IoT en estado del arte y con amplia utilización, como son 6LoWPAN y el sistema operativo Contiki-NG. Se desarrollaron dos camas de pruebas de simulación basadas en Cooja y Renode, y una cama de pruebas física basada en la plataforma Sensortag CC2650. Se encontró que la plataforma Cooja no era adecuada para realizar una evaluación realista. Por otro lado, se encontró que los resultados ofrecidos por la plataforma Renode eran consistentes con el comportamiento y resultados arrojados por la cama de pruebas física basada en el Sensortag CC2650.

Para la implementación de la cama de pruebas se seleccionó el sistema operativo para IoT Contiki-NG. Para realizar la evaluación se tuvo que modificar la implementación de TinyDTLS utilizada en Contiki-NG para poder incluir los algoritmos de cifrado ligero evaluados. Esto fue necesario debido a que TinyDTLS no incluye actualmente ninguno de los algoritmos evaluados utilizando por default el cifrador AES-128. Una vez realizadas las modificaciones de TinyDTLS, se definieron dos escenarios de evaluación que representan un despliegue típico de una red IoMT para la monitorización de oximetría, temperatura y tensión arterial en un paciente (usuario). En este escenario, el flujo de información generado por el nodo oximetro genera datos con una tasa mucho mayor al flujo de datos generado por los nodos termómetro y baumanómetro. Esto definió un flujo de información tipo streaming y dos flujos de información

solicitud-respuesta en el escenario propuesto, lo cual permitió evaluar los efectos de cifrado considerando la transmisión realista de flujos de información con diferentes tasas de datos sobre la misma red IoMT.

Para la evaluación se definieron diversas métricas enfocadas a determinar el desempeño de los algoritmos de cifrado considerando dos rubros:

- Velocidad de procesamiento y sobrecarga.
- Efectos sobre métricas de red y consumo de energía.

Al obtener y analizar las métricas arrojadas en los diferentes escenarios de evaluación planteados en esta tesis, se llegó a las siguientes conclusiones:

- **Sobre el análisis de sobrecarga de bytes.** Se observó que la implementación de uno u otro cifrador ligero ofrece diferencias significativas (entre 8 y 16 bits) en el paquete final que se transmitirá en la red. Esto ocasiona que la cantidad de carga útil máxima que se puede transmitir por paquete sea directamente afectada por la selección del cifrador.
- **Sobre el consumo en memoria de los algoritmos.** Se observó que utilizar TinyJambu128, GIFTCOFB o Xoodyak genera un ahorro de alrededor del 1% en uso de memoria RAM y ROM en comparación AES-128. El ahorro en memoria RAM y ROM proporcionado por los otros algoritmos de cifrado ligero fue menor.
- **Sobre la implementación de las versiones optimizadas de los cifradores**. Se demostró que los algoritmos de cifrado de referencia no son aptos para la utilización en dispositivos de bajas prestaciones debido a que tuvieron un desempeño muy inferior al cifrado por defecto (AES-128) en cuanto a sus tiempos de procesamiento. Lo que si bien no es el objetivo de su diseño, si puede repercutir gravemente a la eficiencia general de la red. Sin embargo, una implementación de una versión optimizada que se adapte a la arquitectura utilizada de estos cifradores si tuvo un mejor desempeño en comparación con el algoritmo de cifrado por defecto.
- **Sobre el tiempo de enlace seguro y tiempo de handshake.** Se observó que el utilizar algoritmos de cifrado ligero puede suponer una reducción de alrededor

del 1.5 % en el tiempo que dura un handshake del protocolo DTLS, lo que puede conllevar a una mejoría del tiempo que le toma a un dispositivo entrar a un entorno seguro. En este sentido Xoodyak fue el algoritmo que mejor tiempo obtuvo al realizar un handshake.

- **Sobre los resultados obtenidos de las métricas de red.** Al analizar los resultados de End to End delay, jitter y latencia obtenidos en los diferentes escenarios de evaluación (malla y multisalto) planteados, se determinó que la utilización de un cifrador u otro puede tener un impacto en el desempeño de la red. Esto posiblemente se deba a que el tiempo de procesamiento del paquete antes de la transmisión varía para cada cifrador. Es importante remarcar que si bien para el stream de oximetría no se notaron diferencias muy grandes en la métrica de End to End delay, en la métrica de latencia de paquetes de temperatura si se notaron diferencias significativas al cambiar de cifrador. Esto implica que la selección del algoritmo de cifrado, si puede tener efectos medibles en el desempeño de una red IoMT basada en 6LoWPAN cuando se transmiten datos con tasas de generación de paquetes heterogéneas. En este sentido, el algoritmo que destacó positivamente en estas pruebas fue Xoodyak, al ser el más rápido y el que mas estabilidad proporciono a la red en los escenarios de malla y multisalto. Por otro lado, mientras que el algoritmo de cifrado por defecto AES-128 fue el que peor desempeño mostró en el escenario de malla y el que más tiempo de procesamiento requirió.

- **Sobre el consumo de potencia de los nodos.** En esta implementación, el variar entre un cifrador u otro no parece mostrar diferencias significativas. Creemos que esto se debe a que hay otros módulos del dispositivo cómo el radio que tienen un impacto mucho mayor respecto al consumo de energía. Sin embargo, esto podría variar al considerar la utilización de ciclos de trabajo en los nodos de la red, lo cual no formó parte del presente trabajo de investigación.

- **Sobre la implementación de la seguridad en una red 6LoWPAN.** Esto tiene un efecto directo en la sobrecarga de la red, debido a que el tamaño de los paquetes transmitidos tiene una longitud de bytes mayor y el tiempo de procesamiento se aumenta alrededor de un 1000 %. Este aumento en el tiempo de procesamiento puede mejorar ligeramente (alrededor del 8 %) utilizando algoritmos de cifrado

ligero en comparación con los algoritmos de cifrado por defecto.

- **Sobre la selección de un candidato para el próximo estándar de cifrado ligero.** Con base en todas las métricas recabadas en esta tesis, se puede concluir que la implementación de los algoritmos de cifrado ligero en aplicaciones IoMT basadas en 6LoWPAN con nodos de capacidades limitadas (p.ej. el Sensortag CC2650) sí pueden suponer una mejora en la eficiencia general de la red. Para esta aplicación en particular, el algoritmo de cifrado Xoodyak es el que se recomienda basado en los resultados obtenidos durante la realización del presente trabajo de tesis.

Finalmente podemos decir que se ha cumplido con los objetivos de este trabajo de investigación, obteniendo una evaluación de diferentes algoritmos de cifrado ligero y el algoritmo de cifrado AES en escenarios de red en donde la seguridad se debe de garantizar desde el momento de conformación de la red.

7.2. Aportaciones

De este trabajo de tesis se destacan las siguientes aportaciones:

- Se implementaron 5 de los 10 finalistas del concurso que está llevando a cabo el NIST para el próximo estándar de cifrado ligero en el módulo de TinyDTLS utilizado por el sistema operativo IoT Contiki-NG. El código de la implementación se puede consultar en el siguiente repositorio de github para futuras investigaciones.
- Se crearon dos camas de pruebas, una en simulación y la otra en físico, las cuales pueden ser de gran utilidad para futuras investigaciones relacionadas con la evaluación de la eficiencia de la red.
- Se propuso una metodología con métricas sólidamente estructuradas para poder evaluar el desempeño de diferentes tecnologías experimentales como los cifradores ligeros en un entorno de red realista.
- Basados en la implementación realizada, se demostró que la selección del algoritmo de cifrado si puede llegar a tener efectos sobre el desempeño de una red

6LowPAN.

7.3. Trabajo futuro

Una vez cumplidos los objetivos de este trabajo de tesis, se considera que aún existen áreas de oportunidad en la evaluación de algoritmos de cifrado y de la implementación de la seguridad en general de las redes IoT. A continuación se muestran las recomendaciones relacionadas con este trabajo de investigación para el trabajo a futuro.

- Utilizando la misma metodología de pruebas, se puede realizar un análisis similar cambiando el stack de tecnologías 6LoWPAN por BLE e IEEE 802.11 para observar si los resultados son similares.
- Utilizando la misma metodología de pruebas, se puede realizar un análisis similar habilitando un protocolo de ciclos de trabajo para observar si existe una diferencia en cuanto al consumo de energía al utilizar diferentes algoritmos de cifrado.
- El exceso de bytes de encabezado en el estándar de DTLS es preocupante para las redes en las que la tasa de datos está limitada, por lo que una propuesta de optimización del estándar DTLS para dispositivos restringidos puede ser de gran utilidad.
- En este trabajo sólo se evaluaron 5 de los 10 finalistas del concurso para el estándar de cifrado ligero, evaluar los otros 5 y comparar los resultados con los de este trabajo complementaría la investigación realizada en este trabajo de tesis.
- Utilizando la misma metodología de pruebas, se puede realizar un análisis similar utilizando redes de sensores con más nodos y más saltos para evaluar cómo se comportan los algoritmos en redes cuya actividad de canal sea extremadamente alta.

Literatura citada

Aguirre, R. E. d. l. P. (2020). *Seguridad en redes inalámbricas de área corporal mediante criptografía ligera*. Tesis de maestría en ciencias, Centro de Investigación y de Estudios Avanzados del IPN.

Alamri, M. H. (2017). *Securing the constrained application protocol (CoAP) for the internet of things (IoT)*. Tesis de maestría en ciencias, Taif University.

Alsubaei, F., Abuhussein, A., y Shiva, S. (2017). Security and privacy in the internet of medical things: Taxonomy and risk assessment. En: *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, oct. IEEE, pp. 112–120.

Antmicro (2018). Renode. Consultado el 7 de febrero de 2022 de `https://renode.io/about/`.

Bansal, S. y Kumar, D. (2020). IoT Ecosystem: A Survey on devices, gateways, operating systems, middleware and communication. *International Journal of Wireless Information Networks*, **27**(3): 340–364.

Barker, E. B. y Dang, Q. H. (2015). Recommendation for key management part 3: Application-specific key management guidance. Reporte técnico, National Institute of Standards and Technology, Gaithersburg, MD.

Barolli, L. (2020). *Advances on broad-band wireless computing, communication and applications*, Vol. 97 de *Lecture Notes in Networks and Systems*. Springer International Publishing, primera edición. Cham.

Beierle, C., Jean, J., y Kolbl, S. (2019). SKINNY-AEAD and SKINNY-Hash. pp. 1–47.

Besher, K. M., Subah, Z., y Ali, M. Z. (2021). IoT sensor initiated healthcare data security. *IEEE Sensors Journal*, **21**(10): 11977–11982.

Birla, J. (2015). Performance metrics in wireless sensor network. *International Journal of Engineering Research & Technology (IJERT)*, **3**(10): 1–4.

Blondeau, C. y Nyberg, K. (2014). Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities. En: *Advances in Cryptology – EUROCRYPT 2014*. Aalto University School of Science, pp. 165–182.

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., y Vikkelsoe, C. (2007). PRESENT: An ultra-lightweight block cipher. En: *Cryptographic Hardware and Embedded Systems - CHES 2007*, Vol. 4727 LNCS. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 450–466.

Bovy, E. (2020). *Comparison of the second round candidates of the NIST lightweight cryptography competition*. Tesis de licenciatura, Radboud University Comparison.

Calik, C., Hasan, M., y Kang, J. (2020). Benchmarking round 2 candidates on microcontrollers NIST lightweight cryptography workshop. Reporte técnico, NIST.

Campbell, R. (2019). Evaluation of post-quantum distributed ledger cryptography. *The Journal of the British Blockchain Association*, **2**(1): 1–8.

ÇAMUR, Z. (2020). *A study of lightweight cryptography*. Tesis de maestría en ciencias, Middle east technical university.

Cárdenas, R. A. H. (2019). *Evaluación del impacto del ciclo de trabajo del transceptor de radio al proceso de formación de redes inalámbricas de sensores multisalto en el contexto de la Internet de las Cosas (IoT)*. Tesis de maestría en ciencias, Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California.

Carrillo, A. M. (2017). *Evaluación del desempeño en la transmisión de señales biomédicas en un ambiente inalámbrico en redes 6LoWPAN*. Tesis de maestría en ciencias, Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California.

Casillas, M., Villarreal-Reyes, S., González, A. L., Martinez, E., y Perez-Ramos, A. (2015). Design guidelines for wireless sensor network architectures in mHealth mobile patient monitoring scenarios. pp. 401–428.

Casillas, M. O. (2012). *Diseño de una red híbrida de dos saltos para aplicaciones de redes de sensores en telemonitoreo y telemedicina*. Tesis de maestría en ciencias, CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA Programa.

Choi, G., Kim, D., y Yeom, I. (2016). Efficient streaming over CoAP. En: *2016 International Conference on Information Networking (ICOIN)*, jan. IEEE, Vol. 2016-March, pp. 476–478.

Contiki-ng (2018). Contiki-NG: The OS for next generation IoT devices. Consultado el 7 de febrero de 2022 de `https://github.com/contiki-ng/contiki-ng`.

Corak, B. H., Okay, F. Y., Guzel, M., Murt, S., y Ozdemir, S. (2018). Comparative analysis of IoT communication protocols. En: *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, jun. IEEE, pp. 1–6.

Cruz, L. A. R. (2017). *Análisis del impacto del número de usuarios y tasa de datos ofrecida en el traspaso entre resumideros de una WBAN/WPAN enfocada a aplicaciones de sistemas del cuidado de la salud*. Tesis de maestría en ciencias, Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California.

Daemen, J., Hoffert, S., Peeters, M., Assche, G. V., y Keer, R. V. (2019). Xoodyak, a lightweight cryptographic scheme.

Deering, S. y Hinden, R. (1998). RFC 2460 - Internet protocol, version 6 (IPv6) specification. Reporte técnico, RFC Editor.

Despaux, F. (2018). *Modeling and evaluation of the end-to-end delay in wireless sensor networks*. Tesis de doctorado, l'Universit´e de Lorraine.

Dhanda, S. S., Singh, B., y Jindal, P. (2020). Lightweight cryptography: A solution to secure IoT. *Wireless Personal Communications*, **112**(3): 1947–1980.

Dierks, T. y Rescorla, E. (2008). RFC 5246 - The transport layer security (TLS) protocol version 1.2. Reporte técnico, RFC Editor.

Dizdarević, J., Carpio, F., Jukan, A., y Masip-Bruin, X. (2019). A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys*, **51**(6): 1–29.

Dobraunig, C., Eichlseder, M., Mendel, F., y Schläffer, M. (2019). Ascon. Consultado el 7 de febrero de 2022 de https://ascon.iaik.tugraz.at.

Dworkin, M. J. (2007). Recommendation for block cipher modes of operation : The CCM mode for authentication and confidentiality. Reporte técnico, National Institute of Standards and Technology, Gaithersburg, MD.

Eclipse (2016). eclipse/tinydtls: Eclipse tinydtls. Consultado el 7 de febrero de 2022 de https://github.com/eclipse/tinydtls.

Elsts, A. (2020a). Documentation: Timers . Consultado el 13 de febrero de 2022 de https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Timers.

Elsts, A. (2020b). Documentation: Energest . Consultado el 13 de febrero de 2022 de https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Energest.

FIPS (2001). Advanced encryption standard (AES). Reporte técnico, National Institute of Standards and Technology, Gaithersburg, MD.

Gallenmüller, S., Schöffmann, D., Scholz, D., Geyer, F., y Carle, G. (2019). DTLS Performance - How expensive is security? *arXiv*.

Gaur, P. y Tahiliani, M. P. (2015). Operating systems for IoT devices: A critical survey. En: *2015 IEEE Region 10 Symposium*, may. IEEE, pp. 33–36.

Greco, L., Percannella, G., Ritrovato, P., Tortorella, F., y Vento, M. (2020). Trends in IoT based solutions for health care: Moving AI to the edge. *Pattern Recognition Letters*, **135**: 346–353.

Hagen, S. (2002). *IPv6 Essentials*. O'Reilly & Associates, Inc., primera edición. USA.

Hartke, K. (2015). RFC 7641 - Observing resources in the constrained application protocol (CoAP). Reporte técnico, RFC Editor.

Hatzivasilis, G., Soultatos, O., Ioannidis, S., Verikoukis, C., Demetriou, G., y Tsatsoulis, C. (2019). Review of security and privacy for the internet of medical things (IoMT). En: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, may. IEEE, pp. 457–464.

Hayajneh, T., Mohd, B., Imran, M., Almashaqbeh, G., y Vasilakos, A. (2016). Secure authentication for remote patient monitoring with wireless medical sensor networks. *Sensors*, **16**(4): 424.

Hell, M., Johansson, T., Meier, W., Sönnerup, J., y Yoshida, H. (2019). Grain-128AEAD-A lightweight AEAD stream cipher cover sheet corresponding submitter: Backup point of contact.

Hermida, R. C., Ayala, D. E., Fernández, J. R., Mojón, A., y Calvo, C. (2007). Influence of measurement duration and frequency on ambulatory blood pressure monitoring. *Revista Española de Cardiología (English Edition)*, **60**(2): 131–138.

Hernandez, F. J. S. (2018). *A comparison of lightweight ciphers meeting nist lightweight cryptography requirements to the advanced encryption standard*. Tesis de maestría en ciencias, California State Polytechnic University.

Hong, D., Lee, J.-K., Kim, D.-C., Kwon, D., Ryu, K. H., y Lee, D.-G. (2014). LEA: A 128-Bit block cipher for fast encryption on common processors. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8267 LNCS. pp. 3–27.

Huang, H. W. y Tao (2019). TinyJAMBU: A family of lightweight authenticated encryption algorithms. *In: Lightweight Cryptography Standardization Process round 2 submission, NIST*, (September).

Hui, J. W. y Culler, D. E. (2008). Extending IP to low-power, wireless personal area networks. *IEEE Internet Computing*, **12**(4): 37–45.

IEEE Computer Society (2020). *IEEE standard for low-rate wireless networks*, Vol. 2020. Primera edición. pp. 1–800.

Isical (2019). GIFT-COFB Authenticated encryption scheme. Consultado el 7 de febrero de 2020 de `https://www.isical.ac.in/$\sim$lightweight/COFB/`.

Jolly, B. (2020). Watts better than mAh for IoT devices? En: *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, aug. IEEE, Vol. 2020-Augus, pp. 245–248.

Karn, P. y Simpson, W. (1999). RFC 2522 - Photuris: session-key management protocol. Reporte técnico, RFC Editor.

Khan, M. F., Felemban, E. A., Qaisar, S., y Ali, S. (2013). Performance analysis on packet delivery ratio and End-to-End delay of different network topologies in wireless sensor networks (WSNs). En: *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, dec. IEEE, pp. 324–329.

Krovetz, T. y Rogaway, P. (2011). The software performance of authenticated-encryption modes. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 6733 LNCS. pp. 306–327.

LAN/MAN Standards Committee (2020). *IEEE Standard for low-rate wireless networks. amendment 1*, Vol. 2020. Primera edición. pp. 1–174.

Larios, D. F., Mora-Merchan, J. M., Personal, E., Barbancho, J., y León, C. (2013). Implementing a distributed WSN based on IPv6 for ambient monitoring. *International Journal of Distributed Sensor Networks*, **9**(6): 328747.

Lee, C. (2014). Biclique cryptanalysis of PRESENT-80 and PRESENT-128. *The Journal of Supercomputing*, **70**(1): 95–103.

Lenstra, A. K. (2007). Key length. En: *Brute Force*. Springer New York, New York, NY, pp. 23–35.

Liu, K., Ma, Q., Liu, H., Cao, Z., y Liu, Y. (2013). End-to-End delay measurement in wireless sensor networks without synchronization. En: *2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, oct. IEEE, pp. 583–591.

LRA01H (2018). Lithium-ion rechargeable coin cell.

Luo, L. (2018). Data acquisition and analysis of smart campus based on wireless sensor. *Wireless Personal Communications*, **102**(4): 2897–2911.

McGrew, D. (2008). RFC 5116 - An Interface and algorithms for authenticated encryption. Reporte técnico, RFC Editor.

McGrew, D. y Bailey, D. (2012). RFC 6655 - AES-CCM Cipher suites for transport layer security (TLS). Reporte técnico, RFC Editor.

Mischie, S. (2017). A MATLAB graphical interface to evaluate the CC2650 sensor tag. *22nd IMEKO TC4 International Symposium and 20th International Workshop on ADC Modelling and Testing 2017: Supporting World Development Through Electrical and Electronic Measurements*, **2017-Septe**: 391–396.

Misic, J. y Misic, V. B. (2010). Bridge performance in a multitier wireless network for healthcare monitoring. *IEEE Wireless Communications*, **17**(1): 90–95.

Mohd, B. J., Hayajneh, T., y Vasilakos, A. V. (2015). A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues. *Journal of Network and Computer Applications*, **58**: 73–93.

Morales-Sandoval, M. y Diaz-Perez, A. (2015). DET-ABE: A Java API for data confidentiality and fine-grained access control from attribute based encryption. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9311. pp. 104–119.

NIST (2009). About NIST. Consultado el 7 de febrero de 2020 de `https://www.nist.gov/about-nist`.

NIST (2018). Submission requirements and evaluation criteria for the lightweight cryptography standardization process. pp. 1–17.

Oikonomou, G. (2018). Tutorial: RAM and ROM usage. Consultado el 13 de febrero de 2022 de `https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-RAM-and-ROM-usage`.

Ojo, M. O., Giordano, S., Procissi, G., y Seitanidis, I. N. (2018). A Review of low-end, middle-end, and high-end Iot devices. *IEEE Access*, **6**: 70528–70554.

Olowu, T. O., Sundararajan, A., Moghaddami, M., Sarwat, A. I., Unigwe, O., Okekunle, D., Kiprakis, A., Latif, A., Gawlik, W., y Palensky, P. P. (2014). *Computer security: Art and science*, Vol. 2017. Primera edición. pp. 1–67.

Olsson, J. (2014). 6LoWPAN demystified. *Texas Instruments*, p. 13.

Parra, R. D. (2020). Tabla resumen de implementaciones de algoritmos criptográficos ligeros. pp. 2–4.

Peyrin, T. y Sim, S. M. (2019). GIFT-COFB Chapter 1. *NIST*.

Pramanik, P. K. D., Sinhababu, N., Mukherjee, B., Padmanaban, S., Maity, A., Upadhyaya, B. K., Holm-Nielsen, J. B., y Choudhury, P. (2019). Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage. *IEEE Access*, **7**: 182113–182172.

Quasim, M. T., Khan, M. A., Abdullah, M., Meraj, M., Singh, S. P., y Johri, P. (2019). Internet of things for smart healthcare: A hardware perspective. En: *2019 First International Conference of Intelligent Computing and Engineering (ICOICE)*, dec. IEEE, pp. 1–5.

Regalado Jalca, J. J., Romero Castro, V. F., Azúa Menéndez, M. D. J., Murillo Quimiz, L. R., Parrales Anzúles, G. R., Campozano Pilay, Y. H., y Pin Pin, Á. L. (2018). *Redes de computadoras*. Editorial Científica 3Ciencias.

Rescorla, E. y Modadugu, N. (2012). RFC 6347 - Datagram transport layer security version 1.2. Reporte técnico, RFC Editor.

Rezvani, B., Coleman, F., Sachin, S., y Diehl, W. (2019). Hardware implementations of NIST lightweight cryptographic candidates: A first look. *Eprint 2019-824*.

Rghioui, A., Bouhorma, M., y Benslimane, A. (2013). Analytical study of security aspects in 6LoWPAN networks. En: *2013 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, mar. IEEE, pp. 1–5.

Rogaway, P. (2002). Authenticated-encryption with associated-data. En: *Proceedings of the 9th ACM conference on Computer and communications security - CCS '02*, New York, New York, USA. ACM Press, p. 98.

Rubí, J. N. S. y Gondim, P. R. d. L. (2019). IoMT Platform for pervasive healthcare data aggregation, processing, and sharing based on OneM2M and OpenEHR. *Sensors*, **19**(19): 4283.

Sanchez-Gallegos, D. D., Galaviz-Mosqueda, A., Gonzalez-Compean, J. L., Villarreal-Reyes, S., Perez-Ramos, A. E., Carrizales-Espinoza, D., y Carretero, J. (2020). On the continuous processing of health data in Edge-Fog-Cloud computing by using micro/-nanoservice composition. *IEEE Access*, **8**: 120255–120281.

Sethi, P. y Sarangi, S. R. (2017). Internet of things: Architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, **2017**: 1–25.

Shelby, Z., Hartke, K., y Bormann, C. (2014). RFC 7252 - The constrained application protocol (CoAP). Reporte técnico, RFC Editor.

Simon Duquennoy (2018). Tutorial: Running Contiki-NG in Cooja · contiki-ng/contiki-ng Wiki. Consultado el 7 de febrero de 2022 de `https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-Running-Contiki\begingroup\let\relax\relax\endgroup[Pleaseinsert\PrerenderUnicode{âĂŘ}intopreamble]NG-in-Cooja`.

Singh, S., Sharma, P. K., Moon, S. Y., y Park, J. H. (2017). Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, **0**(0): 1–18.

Sönmez Turan, M. (2019). On the NIST lightweight cryptography standardization. *ECC 2019: 23rd Workshop on Elliptic Curve Cryptography December 2, 2019*.

Stallings, W. (1996). IPv6: the new Internet protocol. *IEEE Communications Magazine*, **34**(7): 96–108.

Stefan, K. y Schneider, T. (2019). Gimli. *NIST*, pp. 1–48.

Tabassum, A., Erbad, A., y Guizani, M. (2019). A survey on recent approaches in intrusion detection system in IoTs. En: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, jun. IEEE, pp. 1190–1197.

Texas Instruments (2015). CC2650 - SimpleLink™ multistandard wireless MCU.

Thomson, C. (2016). Cooja simulator manual. (C): 2015–2016.

Thomson, S., Narten, T., y Jinmei, T. (2007). RFC 4862 - IPv6 stateless address autoconfiguration. Reporte técnico, RFC Editor.

TI.com (2016). CC2650STK Development kit | TI.com. Consultado el 7 de febrero de 2022 de https://www.ti.com/tool/CC2650STK.

Turan, M. y Bassham, L. (2019). Status report on the first round of the NIST lightweight cryptography standardization process. pp. 1–13.

Vignesh, K. (2017). *Performance analysis of end-to-end DTLS and IPsec-based communication in IoT environments*. Tesis de maestría en ciencias, Blekinge Institute of Technology.

Vishnu, S., Ramson, S. J., y Jegan, R. (2020). Internet of medical things (IoMT) - An overview. En: *2020 5th International Conference on Devices, Circuits and Systems (ICDCS)*, mar. IEEE, pp. 101–104.

Vucinic, M., Tourancheau, B., Watteyne, T., Rousseau, F., Duda, A., Guizzetti, R., y Damon, L. (2015). DTLS performance in duty-cycled networks. En: *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, aug. IEEE, Vol. 2015-Decem, pp. 1333–1338.

Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., P. Levis, K. Pister, R. Struik, JP. Vasseur, y R. Alexander (2012). RFC 6550 - RPL: IPv6 routing protocol for low-power and lossy networks. Reporte técnico, RFC Editor.

YiKai Chen (2015). Difference between CC2650 and CC2538? - Zigbee & Thread forum - Zigbee & Thread - TI E2E support forums. Consultado el 18 de febrero de 2022 de https://e2e.ti.com/support/wireless-connectivity/zigbee-thread-group/zigbee-and-thread/f/zigbee-thread-forum/419264/difference-between-cc2650-and-cc2538.

Yoon, J.-Y. y Kim, B. (2012). Lab-on-a-Chip pathogen sensors for food safety. *Sensors*, **12**(8): 10713–10741.

Zikria, Y. B., Kim, S. W., Hahm, O., Afzal, M. K., y Aalsalem, M. Y. (2019). Internet of Things (IoT) Operating systems management: opportunities, challenges, and solution. *Sensors*, **19**(8): 1793.