UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"IMPLEMENTACIÓN DE CRIPTOGRAFÍA LIGERA BASADA EN ASCON PARA DISPOSITIVOS IOT"

Trabajo de titulación previo a la obtención del título en Ingeniería en Telecomunicaciones

Línea de investigación: Producción industrial y tecnología sostenible.

AUTOR:

Alexander Xavier Arroyo Pupiales

DIRECTOR:

Ing. Fabián Geovanny Cuzme Rodríguez. MSc

Ibarra - Ecuador (2026)

UNIVERSIDAD TÉCNICA DEL NORTE

DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	ARROYO PUPIALES ALEXANDER XAVIER

DATOS DE LA OBRA	
TÍTULO:	IMPLEMENTACIÓN DE CRIPTOGRAFÍA LIGERA BASADA EN ASCON PARA DISPOSITIVOS IOT
AUTOR (ES):	ARROYO PUPIALES ALEXANDER XAVIER
FECHA: DD/MM/AAAA	09/09/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR:	FABIÁN GEOVANNY CUZME RODRÍGUEZ

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 09 días, del mes de septiembre de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
ALEXANDER XAVIER
ARROYO PUPIALES

Alexander Xavier Arroyo Pupiales

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 09 de septiembre de 2026

ING. FABÍAN GEOVANNY CUZME RODRÍGUEZ. MSC

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) …
Validar únicamente en FirmaEC.
Firmado electrónicamente por:
FABIAN GEOVANNY CUZME RODRIGUEZ
……

ING. FABIÁN GEOVANNY CUZME RODRÍGUEZ. MSC

C.C.: 1311527012

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular "IMPLEMENTACIÓN DE CRIPTOGRAFÍA LIGERA BASADA EN ASCON PARA DISPOSITIVOS IOT" elaborado por ARROYO PUPIALES ALEXANDER XAVIER, previo a la obtención del título de INGENIERO EN TELECOMUNICACIONES, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

(f):….Validar únicamente en FirmaEC.
Firmado electrónicamente por:
FABIAN GEOVANNY
CUZME RODRIGUEZ
…….

Ing. Fabián Geovanny Cuzme Rodríguez. MSc

C.C.: 1311527012

DEDICATORIA

A mis padres, Carlos Arroyo y María Carmen Pupiales, por ser mi sostén incondicional en cada etapa de este camino. Gracias por enseñarme el valor del esfuerzo, por creer en mí cuando yo dudaba y por acompañarme con paciencia y amor. Cada logro que alcanzo lleva consigo su sacrificio silencioso. Esta tesis es fruto de sus enseñanzas, su ejemplo y su fe constante. Siempre llevaré conmigo y con mucho orgullo todo lo que me dieron: valores, fuerza y corazón.

A mis hermanos, que juntos a nuestros padres han sido un gran apoyo y fuente de inspiración para este trabajo. Agradezco su confianza en mí en cada etapa, por estar a mi lado en cada idea, cada esfuerzo y cada reto.

A Nicole mi novia, quien siempre me brindó su apoyo, por comprender los silencios de las semanas más difíciles y por hacerme recordar que una meta significativa no debe eliminar la vida que se vive mientras la buscamos. Gracias por ser mi refugio por creer en mí y recordarme que soy capaz de llegar lejos.

Alexander Xavier Arroyo Pupiales

AGRADECIMIENTO

A Dios, por ser mi guía espiritual en este largo camino, gracias a él superé lo momentos difíciles y me levanté para culminar con éxito.

Agradezco a mis padres y a mis hermanos, por ser enseñarme el valor del esfuerzo, la humildad y la perseverancia, gracias por ese amor constante y esa fe incondicional. Sin ustedes, todo esto no habría sido posible. Su amor y sacrificio han sido la luz que guio mi camino a través de este largo camino.

Agradezco a mi hermana Silvia que ha sido una gran fuente de inspiración, que me ha enseñado a nunca rendirme, gracias por siempre creer en mí, por acompañarme en cada desafío y brindarme ese apoyo cuando más lo necesitaba.

Un sincero agradecimiento al Msc. Fabián Cuzme y a mi asesor Msc. Mauricio Dominguez, cuyos conocimientos y consejos han sido de gran apoyo para el desarrollo de este proyecto. Su confianza en mi capacidad y su ayuda oportuna han sido los pilares de este trabajo. Gracias por su paciencia y su orientación en este camino universitario.

Así mismo agradezco al Msc. Jaime Michilena por dejar una huella en mi formación profesional, siempre ha estado para prestarnos sus bastos conocimientos y brindarnos su ayuda cuando más lo necesitamos. Gracias por su apoyo a lo largo de esta travesía universitaria.

A cada uno de mis compañeros que formaron parte de esta etapa e hicieron de esta travesía aún más ameno y lleno alegrías: Kim, Alfon, Lenin, Jhon, Josué, Jeffo, Dennis, David, Pame, Fernando y Diego agradezco haberlos conocido ustedes me han enseñado el valor de la amistad, gracias por todas las ocurrencias y los buenos ratos.

RESUMEN

El presente trabajo desarrolla la implementación de una solución de criptografía ligera basada en ASCON-128 para la protección de datos en dispositivos IoT con recursos limitados, para ello se integran los microcontroladores HELTEC V3, LILYGO T3 y ESP32-WROOM-32 como plataformas de evaluación y un gateway basado en Raspberry Pi como punto de procesamiento, donde se establece un esquema de comunicación que permite generar, cifrar, transmitir y posteriormente descifrar los datos obtenidos durante las pruebas mediante el protocolo MQTT. La seguridad de la comunicación se divide en dos fases, la primera consiste en establecer un canal seguro por medio de ECDH (curva elíptica de Diffie-Hellman) a través de un intercambio de sus llaves públicas, la segunda fase ya incorpora el algoritmo de cifrado ASCON que permitirá proteger los mensajes durante su transmisión. el cifrado se ejecuta en los microcontroladores mediante una implementación en C/C++, mientras que el descifrado se realiza en el gateway utilizando la implementación ASCON-C mediante una biblioteca compilada e integrada con Python. La evaluación experimental se enfoca en analizar el comportamiento temporal que presenta ASCON-128 durante las etapas de cifrado y descifrado, para ello se realizan pruebas controladas sobe los diferentes microcontroladores y se registran las mediciones correspondientes al procesamiento criptográfico, posteriormente se aplican herramientas de análisis estadístico para caracterizar la estabilidad y variabilidad de los tiempos obtenidos. Finalmente, la evaluación de la implementación de ASCON-128 permite analizar el impacto temporal de las etapas de cifrado y descifrado en las plataformas IoT evaluadas, donde el cifrado presenta tiempos promedio de 47,94 µs en la HELTEC V3, 61,07 µs en la LILYGO T3 y 61,91 µs en el ESP32-WROOM-32, mientras que el descifrado registra 57,21 µs, 54,89 µs y 58,62 µs, respectivamente, la incorporación

conjunta de estas operaciones representa una incidencia temporal aproximada de 0,41 %, 0,45 % y 0,47 %, respectivamente, respecto al tiempo total de procesamiento, estos resultados permiten evidenciar que ASCON-128 mantiene un incremento temporal reducido sin afectar la correcta recuperación de los datos durante el proceso de comunicación.

Palabras clave: ASCON-128, criptografía ligera, Internet de las cosas (IoT), ECDH, cifrado, descifrado.

ABSTRACT

This work details the implementation of a lightweight cryptography solution based on ASCON-128 to protect data on resource-constrained IoT devices. HELTEC V3, LILYGO T3, and ESP32-WROOM-32 microcontrollers serve as evaluation platforms, while a Raspberry Pi-based gateway acts as the processing point; a communication scheme is established to generate, encrypt, transmit, and subsequently decrypt test data using the MQTT protocol. Communication security is divided into two phases: the first establishes a secure channel via ECDH (Elliptic Curve Diffie-Hellman) through a public key exchange, while the second incorporates the ASCON encryption algorithm to protect messages during transmission. Encryption is executed on the microcontrollers using a C/C++ implementation, whereas decryption takes place on the gateway using the ASCON-C implementation via a library compiled and integrated with Python. The experimental evaluation focuses on analyzing the timing behavior of ASCON-128 during the encryption and decryption stages; controlled tests are conducted on the various microcontrollers to record cryptographic processing measurements, followed by the application of statistical analysis tools to characterize the stability and variability of the recorded times. Finally, the evaluation of the ASCON-128 implementation allows for an analysis of the timing impact of the encryption and decryption stages on the IoT platforms tested; encryption shows average times of 47.94 µs on the HELTEC V3, 61.07 µs on the LILYGO T3, and 61.91 µs on the ESP32-WROOM-32, while decryption records 57.21 µs, 54.89 µs, and 58.62 µs, respectively. The combined inclusion of these operations represents a timing overhead of approximately 0.41%, 0.45%, and 0.47%, respectively, relative to the total processing time; these results demonstrate that ASCON-128 maintains

a minimal increase in processing time without compromising correct data recovery during the communication process.

Keywords: ASCON-128, lightweight cryptography, Internet of Things (IoT), ECDH, encryption, decryption.

LISTA DE SIGLAS

IoT Internet of Things — Internet de las Cosas

ASCON Familia de algoritmos de criptografía ligera

ECDH Elliptic Curve Diffie-Hellman

AEAD Authenticated Encryption with Associated Data — Cifrado Autenticado con Datos Asociados

IV Initialization Vector — Vector de Inicialización

RAM Random Access Memory — Memoria de Acceso Aleatorio

SRAM Static Random Access Memory — Memoria Estática de Acceso Aleatorio

GPIO General Purpose Input/Output — Entrada/Salida de Propósito General

I**2C** Inter-Integrated Circuit

API Application Programming Interface — Interfaz de Programación de Aplicaciones

JSON JavaScript Object Notation

SQL Structured Query Language

ÍNDICE DE CONTENIDOS

CAPÍTULO I: ANTECEDENTES 17
1.1 Problema 17
1.2 Justificación 19
1.3 Objetivos 21
1.3.1 Objetivo General 21
1.3.2 Objetivos Específicos 21
2 CAPÍTULO II: FUNDAMETACIÓN TEÓRICA 22
2.1 IoT (Internet de las cosas) 22
2.1.1 Arquitectura de Referencia IoT según la ITU-T (Y.2060) 23
2.1.2 Limitaciones en Dispositivos IoT: Energía, Procesamiento y Memoria 26
2.1.3 Protocolos de Comunicación en IoT 27
2.1.3.1 6LowPAN 28
2.1.3.2 AMQP 28
2.1.3.3 COAP 29
2.1.3.4 XMPP 29
2.1.3.5 DDS 29
2.1.4 Tecnologías de comunicación 30
2.1.4.1 LoRaWAN 31
2.1.4.2 SigFox 31
2.1.4.3 NB-IoT 32
2.2 Fundamentos de la Criptografía Ligera. 33
2.2.1 Sistemas Criptográficos Primitivos 33
2.2.2 Criptografía Simétrica 34
2.2.3 Características de la Criptografía Ligera 36
2.3 Algoritmo ASCON 36
2.3.1 Funcionamiento y Estructura del Algoritmo 36
2.3.2 Características del Algoritmo 38
2.4 Comparación de ASCON con otros algoritmos ligeros 39
2.4.1 Photon-B 39
2.4.2 Elephant 40
2.4.3 ISAP 40
2.4.4 TinyJAMBU 41

3 CAPÍTULO III: DISEÑO DEL SISTEMA 42
3.1 Metodología 42
3.2 Etapa de Requerimientos 43
3.2.1 Requerimientos del Proyecto 43
3.2.2 Nomenclatura de Requerimientos 44
3.2.3 Listado de Stakeholders 44
3.2.4 Requerimientos Operacionales 45
3.2.5 Requerimientos del Sistema 46
3.2.6 Requerimientos de Hardware y Software 47
3.2.7 Elección de Hardware y Software 49
3.2.7.1 Elección de Hardware 49
3.2.7.2 Elección de Software 60
3.3 Fase de Diseño 61
3.3.1 Diagrama de bloques del Sistema 61
3.3.1.1 Diagrama de Bloque del Nodo Sensor. 63
3.3.1.2 Diagrama de Bloques de Gateway Principal 64
3.3.2 Arquitectura del Sistema IoT 65
3.3.3 Diagramas de Flujo 66
3.3.3.1 Diagrama de Flujo del Sistema IoT 66
3.3.3.2 Diagrama de Flujo del Nodo Sensor 67
3.3.3.3 Diagrama de Flujo del Gateway 69
3.3.3.4 Diagrama de Secuencia 71
3.3.4 Diagrama del Mecanismo de Seguridad 73
3.3.5 Diseño de Conexiones del Sistema 74
3.3.5.1 Alimentación 74
3.3.5.2 Esquemas de Conexiones del Sistema 75
3.3.5.3 Diagrama de Pines 77
3.3.6 Librerías 78
3.4 Fase de Implementación 80
3.4.1 Implementación del Mecanismo de Seguridad 81
3.4.1.1 Establecimiento de la clave mediante ECDH 81
3.4.1.2 Derivación de la clave ASCON 82
3.4.1.3 Protección del mensaje mediante ASCON 84
3.4.1.4 Verificación y descifrado en el Gateway 85

3.4.2 Implementación de la Comunicación 86
3.4.3 Implementación de Métricas de Evaluación 87
3.4.3.1 Medición de Consumo Energético 87
3.4.3.2 Medición del Tiempo de Procesamiento 89
4 CAPÍTULO IV: ANÁLISIS DE RESULTADOS 90
4.1 Plan de Pruebas 90
4.1.1 Prueba PR-01: Determinación del Número de Repeticiones 92
4.1.2 Prueba PR-02 Determinación del tamaño máximo de procesamiento 93
4.2 Evaluación de ASCON en los Nodos IoT 94
4.2.1 Prueba PR-03: Establecimiento del Canal Seguro y Cifrado ASCON 95
4.2.2 Prueba PR-04: Tiempo de Procesamiento de ASCON 97
4.2.2.1 Tiempo de Procesamiento sin Cifrado 97
4.2.2.2 Tiempo de cifrado en los nodos IoT 98
4.2.2.3 Tiempo de descifrado en el gateway 99
4.2.3 Comparación del Tiempo de Cifrado y Descifrado 101
4.2.3.1 Tiempo Total con Cifrado 101
4.2.3.2 Tiempo total con descifrado 103
4.2.3.3 Tiempo Total con Cifrado y Descifrado 105
4.2.4 Prueba PR-06: Consumo Energético Durante ASCON-128 107
4.2.5 Prueba PR-07: Consumo Energético Durante Descifrado 109
4.3 Análisis General de Resultados 110
4.3.1 Análisis de Tiempo de Cifrado 111
4.3.2 Análisis de Tiempo de Descifrado 112
4.3.3 Análisis de Consumo Energético Durante Cifrado. 113
CONCLUSIONES Y RECOMENDACIONES 115
REFERENCIAS 118
ANEXOS 123

ÍNDICE DE TABLAS

Tabla 1 Propiedades comparativas de los principales protocolos IoT 30
Tabla 2 Propiedades de las tecnologías LPWAN 32
Tabla 3 Parámetros para Ascón cifrado autenticado 38
Tabla 4 Nomenclatura para el análisis de requisitos. 44
Tabla 5 Lista de involucrados 45
Tabla 6 Requerimientos Operacionales 45
Tabla 7 Requerimientos de Interfaz 46
Tabla 8 Requerimientos de Software 48
Tabla 9 Comparativa nodo sensor placa ESP 50
Tabla 10 Comparativa nodo sensor placa Lora 51
Tabla 11 Características de Heltec Wi-fi Lora (Versión 3) 51
Tabla 12 Comparativa nodo sensor placa LylyGo 53
Tabla 13 Características nodo sensor placa LylyGo 53
Tabla 14 Comparativa placa de procesamiento. 54
Tabla 15 Características placa de procesamiento. 55
Tabla 16 Selección sensor de temperatura 56
Tabla 17 Características sensor DTH11 57
Tabla 18 Selección módulo de corriente 58
Tabla 19 Características sensor de voltaje 59
Tabla 20 Comparativa de lenguajes de programación 60
Tabla 21 Consumo de energía 74
Tabla 22 Pines de los nodos IoT 78
Tabla 23 Tópicos MQQT del sistema IoT 87
Tabla 24 Pruebas realizadas dentro del sistema 91
Tabla 25 Tiempo promedio cifrado ASCON 92
Tabla 26 Tamaño máximo de mensajes procesados 93
Tabla 28 Tiempo promedio de cifrado ASCON 98
Tabla 29 ***Tiempo promedio de descifrado ASCON*** 99
Tabla 30 Tiempo de procesamiento total con ASCON expresada en microsegundos . 102
Tabla 31 Tiempo de procesamiento total con ASCON expresada en segundos 103
Tabla 32 Tiempo de procesamiento con descifrado ASCON-128 104
Tabla 33 Tiempo de procesamiento con descifrado ASCON-128 en segundos 105
Tabla 34 Tiempo total de procesamiento con cifrado y descifrado ASCON-128 106
Tabla 35 Consumo energético durante cifrado ASCON 108
Tabla 36 Análisis estadístico del tiempo de ejecución 111
Tabla 37 Análisis estadístico del tiempo de cifrado 112
Tabla 38 Análisis estadístico del consumo energético durante el cifrado 113

ÍNDICE DE FIGURAS

Figura 1 Árbol de problemas. 19
Figura 2 Modelo de referencia IoT 26
Figura 6 Funcionamiento básico de la criptografía simétrica. 35
Figura 7 Estructura del cifrado Ascon. (Proceso de cifrado y Proceso de descifrado.) 37
Figura 8 Metodología en Cascada 43
Figura 9 Diagrama de bloques del sistema 62
Figura 10 Diagrama de nodo sensor 63
Figura 11 Diagrama de bloques del Gateway 64
Figura 12 Arquitectura del sistema IoT 66
Figura 13 Diagrama de flujo del sistema 67
Figura 14 Diagrama de flujo del nodo 68
Figura 15 Diagrama de flujo gateway 70
Figura 16 Diagrama de secuencia del sistema 71
Figura 17 Segunda fase envio de datos cifrado. 72
Figura 18 Diagrama del mecanismo de seguridad 73
Figura 19 Esquema de conexiones físicas 76
Figura 20 Diagrama de pines del sistema 77
Figura 21 Tiempo de cifrado 92
Figura 22 Tamaño maximo de payload procesado. 94
Figura 23 Proceso de intercambio de clave publica en el nodo. 95
Figura 24 Establecimiento de ECDH en el gateway 96
Figura 25 Tiempo promedio evaluado para ASCON 99
Figura 26 Tiempo promedio de descifrado ASCON 100
Figura 27 Energía de cifrado ASCON 108

CAPÍTULO I: ANTECEDENTES

1.1 Problema

En la actualidad, la modernización y automatización de diversas tareas cotidianas han impulsado el desarrollo de nuevas tecnologías. Uno de los esquemas emergentes más prometedores es conocido como el Internet de las Cosas (IoT, por sus siglas en inglés). Los dispositivos IoT son físicamente pequeños para lograr bajos costos de producción, lo que limita tanto la capacidad de las unidades de procesamiento como la memoria disponible. Estas limitaciones han incrementado los riesgos de seguridad para los usuarios. Los dispositivos IoT operan con información obtenida directamente del entorno. Dependiendo de su aplicación, esta información puede ser inherentemente sensible, y en ciertos casos, el volumen de datos recopilados puede ser utilizado para inferir comportamientos del usuario, comprometiendo su privacidad (Lara et al., 2020).

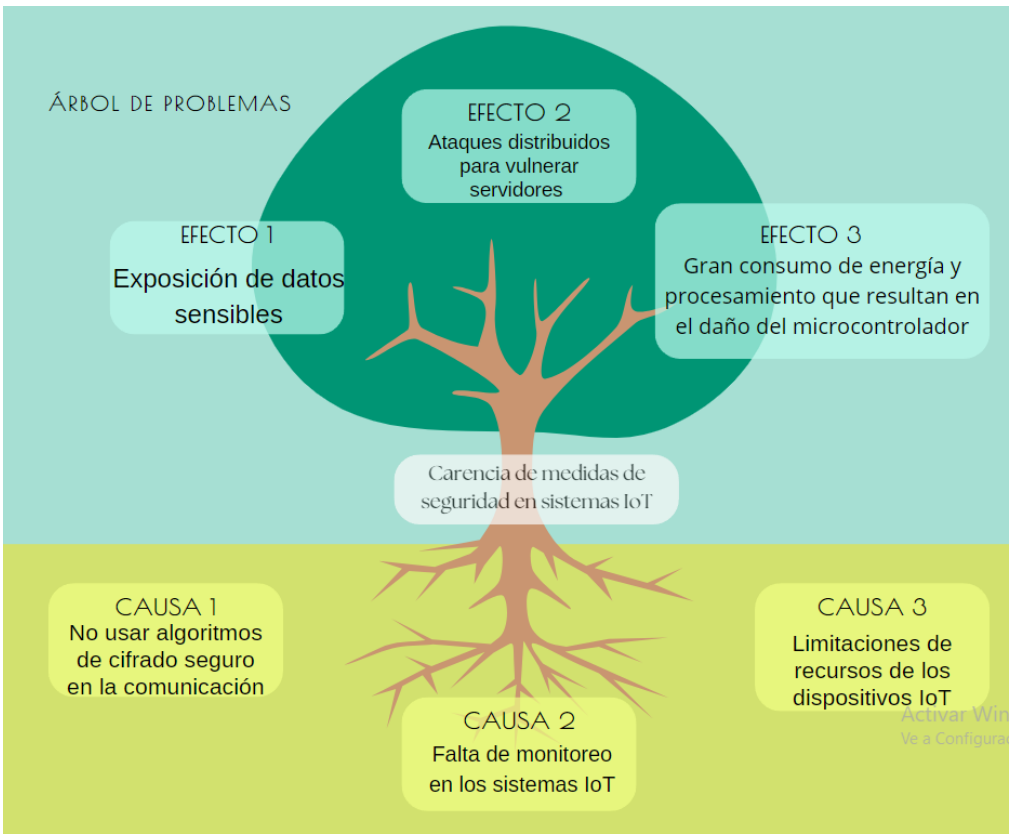
El Internet ha crecido a un ritmo vertiginoso, desde una pequeña red de investigación de unos pocos dispositivos hasta una red compleja de miles de millones de dispositivos heterogéneos en todo el mundo, que se ha adoptado rápidamente y que tiene un amplio espectro de tamaño, capacidad y características y que proporciona diferentes servicios a sus usuarios. A medida que Internet se infiltra cada vez más en la vida humana y el mundo físico, aumentan también las superficies de ataque y los posibles vectores de ataque. Por ejemplo, las vulnerabilidades de una insignificante máquina de café inteligente pueden permitir a un atacante acceder y manipular todos los dispositivos y datos de la red doméstica, incluidos sensores y dispositivos médicos esenciales para la vida, como bombas de insulina o marcapasos conectados al cuerpo humano. (Khan et al., 2020)

Rosa et al. (2023) afirman que “la mayoría de los dispositivos en IoT no utilizan algoritmos de autenticación o cifrados seguros en su comunicación. Por lo tanto, es necesario utilizar algoritmos de cifrado específicos para IoT, ya que los esquemas de cifrado tradicionales requieren recursos que no están disponibles en esos dispositivos”.

Los sistemas basados en IoT necesitan primitivas criptográficas para cumplir con los requisitos de seguridad, privacidad y confianza. Sin embargo, debido a las limitaciones de recursos en los dispositivos IoT, no todas las primitivas criptográficas se pueden usar de manera integral. Es necesario utilizar algoritmos de cifrado específicos para IoT, ya que los esquemas de cifrado tradicionales requieren recursos que no están disponibles en esos dispositivos. Además, los mecanismos de autenticación e integridad livianos también son esenciales en los dispositivos IoT, especialmente en redes de sensores implementadas en lugares públicos (Khan et al., 2020; Rosa et al., 2023).

La protección de dispositivos IoT requiere un enfoque especializado en criptografía que pueda adaptarse a sus limitaciones inherentes de procesamiento y energía, sin comprometer la seguridad de los datos. La adopción de criptografía ligera y protocolos optimizados es esencial para garantizar la integridad y confidencialidad de la información en un entorno de dispositivos heterogéneos y vulnerables. En la figura 1, se presenta las causas y sus efectos respecto a la falta de medidas de seguridad en estos dispositivos.

Figura 1
Árbol de problemas.



Nota. La figura 1 presenta un árbol de problemas sobre la carencia de seguridad en ambientes IoT, detallando las causas y los efectos que pueden generarse si no se llega a implementar un sistema de seguridad.

1.2 Justificación

El aumento en la dependencia de dispositivos IoT en áreas críticas presenta nuevos retos de seguridad, particularmente debido a las restricciones de procesamiento y energía de estos aparatos, al alinearse con el Objetivo de Desarrollo Sostenible (ODS) número 9 de la ONU, que fomenta el construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación (ONU, 2015). Este proyecto busca contribuir al desarrollo de una infraestructura digital más segura y confiable mediante un modelo de seguridad adaptable tanto a entornos industriales como domésticos. Al optimizar la protección de los dispositivos conectados, el proyecto apoya la creación de infraestructuras digitales resilientes, esenciales para las industrias modernas.

Como señala (Cruz, 2023), el algoritmo de cifrado y autenticación ligera ASCON representa un avance crucial en la implementación de controles de seguridad para sistemas ciberfísicos que operan con hardware de bajo rendimiento, como placas de procesamiento limitado y almacenamiento reducido. En sistemas que manejan datos sensibles y privados, es esencial contar con un mecanismo de cifrado eficiente que no solo asegure la integridad y confidencialidad de la información, sino que también minimice el consumo de recursos, facilitando una transmisión de datos rápida y segura.

En el ámbito de IoT, muchos dispositivos interconectados de recursos limitados no están diseñados para llevar a cabo cálculos complejos que requieren un alto consumo de energía. Este es el caso de los algoritmos criptográficos convencionales, los cuales han demostrado ser altamente demandantes en redes tradicionales. La mayoría de estos algoritmos son difíciles de implementar en dispositivos IoT, ya que muchos carecen de la energía y la memoria necesarias para ejecutar procesos de criptografía convencionales. (De la Parra, 2020). En respuesta a estas limitaciones, el algoritmo de criptografía ligera ASCON ha sido creado para proporcionar cifrado y autenticación con el menor uso de recursos, brindando de esta manera una solución de seguridad eficaz y práctica.

Una de las características de los dispositivos IoT no es solo la interacción con el entorno sino también con otros dispositivos conectados. Esto implica que una vulnerabilidad menor en un dispositivo puede desencadenar un ataque a otro objeto conectado, con repercusiones significativamente mayores. La interconexión de dispositivos IoT añade, por tanto, mayor complejidad a la protección y mitigación de riesgos, ya que un punto débil puede comprometer la seguridad general del ecosistema. En este contexto, ASCON se convierte en una herramienta clave, al ofrecer un mecanismo de cifrado y autenticación ligero y eficiente, ideal para los dispositivos IoT con recursos limitados. (Oliveira, 2024)

La aplicación de criptografía ligera basada en ASCON en aparatos IoT satisface la urgencia de disponer de sistemas de seguridad ajustados a las restricciones de estos entornos. Al ofrecer una respuesta eficaz en cuanto al uso de recursos, ASCON no solo incrementa la protección individual de cada aparato, sino que también favorece la resistencia y estabilidad de estos entornos.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un prototipo IoT para la transmisión de datos segura basado en criptografía ligera ASCON basado en una red de sensores inalámbricos.

1.3.2 Objetivos Específicos

1. Analizar el algoritmo de criptografía ligera ASCON, identificando su impacto en la optimización de recursos y la seguridad en dispositivos IoT de bajo consumo.
2. Definir los requerimientos de hardware para optimizar el uso de recursos mínimos en un entorno IoT, logrando un equilibrio entre eficiencia energética, velocidad de procesamiento y robustez en la protección de datos.
3. Implementar el algoritmo ASCON en un entorno IoT, verificando la transmisión y recepción de datos, y asegurando que el sistema garantice la integridad y confidencialidad de la información intercambiada.
4. Evaluar el rendimiento del sistema mediante pruebas de seguridad y consumo de recursos, validando la eficacia de la criptografía ligera en la transmisión de datos en un entorno IoT.

2 CAPÍTULO II: FUNDAMETACIÓN TEÓRICA

Este capítulo analiza los conceptos básicos y las tecnologías asociadas con la criptografía ligera, enfatizando su uso en un entorno ambiental limitado, como Internet de las cosas (IoT). Los conceptos más importantes que nos permiten entender cómo están diseñados estos mecanismos de seguridad para proteger la integridad, la confidencialidad y la autenticidad de los datos sin comprometer el rendimiento de los sistemas en los que se introducen.

El capítulo también explorara el funcionamiento del algoritmo ASCON elegido por el NIST, presentando sus propiedades más importantes junto con la comparación con otros algoritmos criptográficos ligeros reconocidos. El propósito de este análisis es crear una base sólida para comprender los beneficios que ofrece ASCON en términos de eficiencia, seguridad y adaptabilidad en el entorno actual, como redes inteligentes, sistemas integrados y aplicaciones industriales. Finalmente, se enfatizará la importancia de integrar soluciones criptográficas apropiadas como un papel clave en el diseño de un sistema IoT seguro y sostenible.

2.1 IoT (Internet de las cosas)

Los entornos inteligentes representan un avance significativo en la digitalización global del siglo XXI. La incorporación de tecnología para dotar de "inteligencia" a diversas industrias, servicios y aplicaciones se ha expandido ampliamente, impulsada por sus beneficios económicos y sociales.

Sin embargo, debido a su creciente adopción y presencia ubicua, es fundamental comprender los riesgos y amenazas, tanto nuevas como preexistentes, a los que se enfrentan estas soluciones. Para ello, es esencial conocer estos entornos, que constituye la base sobre la cual se desarrollan estas aplicaciones y servicios.

Los entornos IoT hace referencia a la disposición de las diferentes capas o funciones que definen la arquitectura general de la tecnología IoT aplicada a los casos de uso e implementaciones, y los elementos que la conforman.

2.1.1 Arquitectura de Referencia IoT según la ITU-T (Y.2060)

El modelo de referencia propuesto por la ITU fue creado en el año 2012, bajo el nombre de la recomendación Y.2060, y luego actualizado en el 2016 a Y.4000. Generalmente, dentro de las investigaciones y trabajos, este modelo de referencia es llamado Y.4000/2060. (Chuquimarca & Maita, 2022).

Como señalan Chuquimarca y Maita (2022), el modelo Y.4000/2060 está compuesto por 4 capas. Estas son similares a las propuestas en las otras arquitecturas ya estudiadas. Sin embargo, la ITU ha sido más clara respecto al objetivo que cada capa posee en su modelo. De manera adicional y paralela, el modelo propone capacidades para la gestión y la seguridad. Las capas del modelo son: capa de Aplicación, capa de Apoyo a Servicios y Aplicaciones (SSAS) (SSAS), capa de Red y la capa de Dispositivos.

- **Capa de Aplicación**

Permite la interacción del sistema IoT con sus usuarios finales y poder así cumplir con el propósito para el cual fue diseñado. Similar al caso de la capa SSAS, el modelo no realiza ninguna mención a tecnologías admitidas en su modelo. Simplemente indica que la capa de aplicación es aquella que contiene las aplicaciones IoT. (ITU-T, 2016).

- **Capa de Apoyo a Servicios y Aplicaciones (SSAS)**

Como afirma la ITU-T (2016), la capa de soporte de servicios y aplicaciones consiste en los dos siguientes grupos de capacidades:

Capacidades de soporte genéricas: Son capacidades comunes que puede utilizarlas diferentes aplicaciones IoT, tales como procesamiento o almacenamiento de datos. Estas capacidades también pueden utilizarlas otras capacidades específicas para, por ejemplo, crear otras capacidades específicas.

Capacidades de soporte específicas: Son capacidades para atender las necesidades particulares de diversas aplicaciones. En realidad, pueden consistir en diversos grupos de capacidades precisas que ofrecen distintas funciones de apoyo a las diferentes aplicaciones IoT.

- **Capa de red.**

Según la ITU-T (2016), la capa de red consiste en los dos tipos siguientes de capacidades:

Capacidades de red: ofrecen funciones de control de la conectividad en red, tales como funciones de control de acceso y de recursos de transporte, gestión de la movilidad y autentificación, autorización y contabilidad (AAA).

Capacidades de transporte: centradas en suministrar conectividad para el transporte de información y datos específicos de servicios y aplicaciones IoT, así como el transporte de información de control y gestión relacionada con IoT.

- **Capa de dispositivo**

Se puede efectuar una clasificación lógica de las capacidades de la capa de dispositivo, en dos tipos:

Capacidades de gestión: análogamente a las redes de comunicaciones tradicionales, las capacidades de gestión IoT abarcan las clases tradicionales de fallos, configuración, contabilidad, rendimiento y seguridad (FCAPS), es decir, la gestión de fallos, de la configuración, de la contabilidad, del rendimiento y de la seguridad. Las capacidades de

gestión IoT pueden clasificarse en capacidades genéricas y específicas. Las capacidades de gestión genéricas en IoT son esencialmente las siguientes:

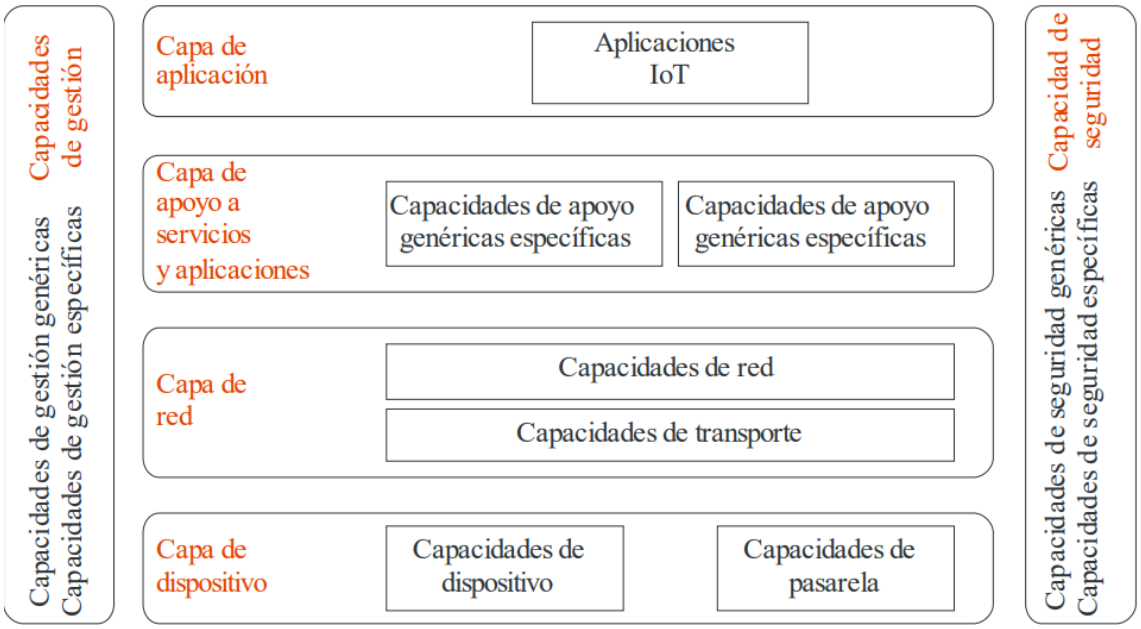
a. Gestión de dispositivos, como activación y desactivación de dispositivos remotos, diagnóstico, actualización del firmware y/o del software, gestión del estado de trabajo del dispositivo.
b. Gestión de la topología de red local.
c. Gestión del tráfico y la congestión, como la detección de las condiciones de saturación de red y la aplicación de reserva de recursos para los flujos de datos esenciales para la vida o urgentes.

Capacidades de seguridad: hay dos tipos de capacidades de seguridad: genéricas y específicas. Las capacidades de seguridad genéricas son independientes de la aplicación y son, entre otras:

a. En la capa de aplicación: autorización, autentificación, confidencialidad de datos de aplicación y protección de la integridad, protección de la privacidad, auditorías de seguridad y antivirus.
b. En la capa de red: autorización, autentificación, confidencialidad de datos de señalización y de datos de uso, y protección de la integridad de señalización.
c. En la capa de dispositivo: autentificación, autorización, validación de la integridad del dispositivo, control de acceso, confidencialidad de datos y protección de la integridad.

Las capacidades de seguridad específicas están estrechamente relacionadas con los requisitos específicos de la aplicación, por ejemplo, los requisitos de seguridad para el pago con el móvil. (ITU-T, 2016).

Figura 2
Modelo de referencia IoT



Nota. Representación de la arquitectura funcional de un sistema IoT según la recomendación Y.4000/2060 de la ITU-T. Se presentan sus cuatro capas principales y las capacidades transversales de gestión y seguridad. Tomada de ITU-T (2016)

2.1.2 Limitaciones en Dispositivos IoT: Energía, Procesamiento y Memoria

Poonam (2023) indica que uno de los principales desafíos que enfrentan los dispositivos con recursos limitados es su poder de procesamiento limitado. Por lo tanto, estos dispositivos deben equilibrar cuidadosamente sus tareas para garantizar un funcionamiento eficiente. Poca memoria, poca potencia de procesamiento y poca duración de batería son las características distintivas de los dispositivos típicos de IoT.

Restricciones de Memoria

La escasa capacidad de memoria es otro obstáculo al que se enfrentan estos dispositivos. El almacenamiento de datos, la administración de actualizaciones de firmware y la implementación de protocolos de seguridad se convierten en tareas complejas cuando la memoria es escasa. Esta limitación requiere soluciones innovadoras

como la compresión de datos, la gestión eficiente de la memoria y la priorización de datos críticos para garantizar una funcionalidad óptima. (Poonam, 2023)

Dilema de Eficiencia Energética

La eficiencia energética es una piedra angular del diseño de dispositivos con recursos limitados. Estos dispositivos a menudo dependen de baterías o fuentes de baja potencia, lo que requiere una gestión cuidadosa de la energía. Equilibrar la necesidad de transmisiones de datos frecuentes, procesar tareas y mantener la longevidad operativa presenta un desafío delicado. Se emplean técnicas innovadoras de administración de energía, como los modos de suspensión y la recolección de energía, para garantizar que estos dispositivos puedan funcionar de forma autónoma durante períodos prolongados. (Poonam, 2023)

Es fundamental resaltar estas diferencias, ya que afectan directamente el diseño e implementación de las estrategias de seguridad y privacidad requeridas en los sistemas IoT, Makhdoom et al. (2018) señalan que “los sistemas de IoT requieren algoritmos de seguridad livianos que mantengan un equilibrio entre la seguridad y el consumo de recursos, como la duración de la batería, la memoria y el uso del procesador” (p. 253).

Debido a estas limitaciones, los dispositivos IoT son más vulnerables a ataques, ya que presentan falta de actualizaciones de software, debilidades en el control de acceso y dependencia de múltiples dispositivos dentro de la red. (Makhdoom et al., 2018)

2.1.3 Protocolos de Comunicación en IoT

Los dispositivos, protocolos e infraestructuras de IoT deben proporcionar mecanismos para garantizar los servicios de Confidencialidad, Disponibilidad y Seguridad (CIA), cualquier aplicación de IoT debe proporcionar servicios basados en los siguientes requisitos para proteger la información confidencial. (Yugha & Chithra, 2020)

- **Autenticidad:** Sólo los nodos autorizados pueden participar en la comunicación entre dos nodos cualesquiera.
- **Confidencialidad:** Se evitará la fuga de información sensible a cualquier usuario no autorizado.
- **Integridad:** Al transferir la información a los dispositivos IoT, la integridad de los datos garantiza la originalidad de la información, es decir, que no sea fabricada, reescrita, copiada o reemplazada por el atacante.
- **Privacidad:** La identidad de un usuario individual debe estar protegida por el sistema IoT seguro para mantener la privacidad.
- **Disponibilidad:** Un usuario autorizado puede utilizar diversos servicios proporcionados por los protocolos de red IoT que pueden soportar la disponibilidad de los servicios.

2.1.3.1 6LowPAN

Desde el punto de vista de la seguridad, 6LowPAN sufre varios ataques que apuntan a dañar directamente la red. 6LowPAN es vulnerable a ataques de suplantación de identidad, reenvío selectivo, ataques Sybil, ataques de agujero de gusano y ataques de sumidero. Estos ataques pueden iniciarse internamente mediante código malicioso o intrusión y externamente mediante dispositivos o usuarios no autorizados. (Yugha & Chithra, 2020)

2.1.3.2 AMQP

Protocolo avanzado de colas de mensajes (AMQP) es un protocolo centrado en mensajes que proporciona comunicación de publicación-suscripción y punto a punto mediante entrega de mensajes garantizada a través de TCP/IP. En términos de seguridad, admite los protocolos de autenticación de capa segura y autenticación simple (SASL) y

seguridad de la capa de transporte (TLS) para la comunicación segura de datos. (Yugha & Chithra, 2020)

2.1.3.3 COAP

El Protocolo de Aplicación Restringida (CoAP) es un protocolo de transferencia web como HTTP (Protocolo de Transporte de Hipertexto) que proporciona compatibilidad con unidifusión y multidifusión para redes IoT. Funciona sobre la base de una solicitud-respuesta entre puntos finales en función de la arquitectura cliente-servidor. TLS es el protocolo de cifrado que se utiliza en HTTP, pero su implementación es difícil para los dispositivos IoT. Por lo tanto, CoAP se utiliza para la seguridad de la capa de transporte de datagramas (DTLS), que proporciona una comunicación segura. (Yugha & Chithra, 2020)

2.1.3.4 XMPP

El Protocolo Extensible de Mensajería y Presencia (XMPP) es un protocolo de comunicación TCP basado en el Lenguaje de Marcado Extensible (XML) que se utiliza para mensajería instantánea. Desde el punto de vista de la seguridad, XMPP proporciona mecanismos de seguridad como la autenticación mediante técnicas SASL y la comunicación segura mediante TLS, pero no proporciona ningún nivel de calidad de servicio (QoS). (Yugha & Chithra, 2020)

2.1.3.5 DDS

Servicio de distribución de datos (DDS) no tiene intermediarios y se utiliza para multidifusión con alta confiabilidad para dispositivos IoT de baja latencia. El DDS se utiliza en varias aplicaciones, como administración de redes inteligentes, aplicaciones médicas, robótica, generación de energía, simulación, pruebas, etc. DDS proporciona

mecanismos de seguridad sólidos, que incluyen autenticación, control de acceso, confidencialidad e integridad de los datos. (Yugha & Chithra, 2020)

Tabla 1
Propiedades comparativas de los principales protocolos IoT

Protocolo	Transporte	Características	Ámbito de Aplicación	Conocimiento del Contenido	Seguridad
6LoWPAN	TCP/IP	Mapea servicios IPv6 a través de WPAN de bajo consumo	D2D, D2C, M2M	Ninguno	TLS
MQTT	TCP/IP	Patrón de publicación/suscripción, flexible y simple	D2C	Ninguno	TLS
AMQP	TCP/IP	Comunicación publicación-suscripción y punto a punto	D2D, D2C, C2C	Ninguno	TLS
COAP	UDP/IP	Transferencia segura y confiable con recursos limitados	D2D	Ninguno	DTLS
XMPP	TCP/IP	Mensajería instantánea (IM) para chats, voz y video	D2D, D2C, C2C	Ninguno	TLS
DDS	TCP/UDP	Datos escalables, confiables, alto rendimiento, publicación-suscripción	D2D, D2C, C2C	Prioridades de transporte	TLS, DTLS, DDS

Nota. La tabla muestra las características principales de DDS. Tomada de(Yugha & Chithra, 2020)

2.1.4 Tecnologías de comunicación

Según Villa & Morales (2017), las tecnologías de comunicación IoT se pueden dividir en diferentes grupos, dependiendo del criterio. Uno de los criterios más útiles es la capacidad de cobertura y ancho de banda de transmisión del protocolo.

Los protocolos LPWAN (Low Power Wide Area Network), son redes caracterizadas por proveer una gran cobertura sobre anchos de banda reducidos utilizando poca energía. Los nodos LPWAN logran un alcance en el orden de los kilómetros, dependiendo del área urbana o rural donde se desplieguen.). (Aguilar, 2020). (hasta 40 km en zonas rurales y hasta 10 km en zonas urbanas). (Villa & Morales, 2017)

2.1.4.1 LoRaWAN

LoRa es una tecnología de capa física que modula las señales en la banda ISM sub-GHZ utilizando una técnica patentada de espectro ensanchado. LoRa utiliza bandas ISM sin licencia, es decir, 868 MHz en Europa, 915 MHz en América del Norte y 433 MHz en Asia. La comunicación bidireccional se proporciona mediante la modulación de espectro ensanchado por chirrido (CSS) que difunde una señal de banda estrecha sobre un ancho de banda de canal más amplio. La velocidad de datos de LoRa está entre 300 bps y 50 kbps según el factor de expansión y el ancho de banda del canal y puede lograr un alcance en zonas urbanas de hasta 5 kilómetros y en zonas suburbanas hasta los 20 kilómetros de distancia. (Mekki et al., 2018)

2.1.4.2 SigFox

Sigfox es un operador de red LPWAN que ofrece una solución de conectividad IoT de extremo a extremo basada en sus tecnologías patentadas. Sigfox implementa sus estaciones base propietarias equipadas con radios cognitivas definidas por software y las conecta a los servidores back-end mediante una red basada en IP.

Sigfox utiliza bandas ISM sin licencia, por ejemplo, 868 MHz en Europa, 915 MHz en América del Norte y 433 MHz en Asia. Al emplear la banda ultraestrecha, Sigfox utiliza el ancho de banda de frecuencia de manera eficiente y experimenta niveles de ruido muy bajos (Mekki et al., 2018), tiene un alcance en zonas urbanas de hasta 10 kilómetros

y alcanza velocidades de datos de 100 bps y la carga útil en el uplink es de 12 bytes y en el downlink es de 8 bytes. (Aguilar, 2020).

2.1.4.3 NB-IoT

Según (Aguilar, 2020) es normada por el Proyecto de Asociación de 3° Generación y opera en bandas de frecuencia licenciadas asignadas a la telefonía móvil. El alcance que logra es de hasta 1 kilómetro en zonas urbanas y utiliza la modulación QPSK. Puede desplegarse en anchos de banda de 200 kHz de forma autónoma, en bandas de guarda o dentro de una banda de una portadora LTE. Puede lograr velocidades de hasta 200 kbps, una carga útil de 1600 bytes y para su utilización se requiere la presencia de una empresa de comunicaciones móviles que brinde el servicio en la zona geográfica de interés.

Tabla 2
Propiedades de las tecnologías LPWAN

Características	**SigFox**	**LoRaWAN**	**NB-IoT**
Técnica de Modulación	BPSK	CSS	QPSK
Ancho de banda	100 Hz	250 kHz y 125 kHz	200 kHz
Frecuencia	868 MHz en Europa, 915 MHz en América del Norte, 433 MHz en Asia	868 MHz en Europa, 915 MHz en América del Norte, 433 MHz en Asia	Frecuencia LTE autorizada
Velocidad máxima	100 bps	50 kbps	200 kbp
Duplexidad	Half-duplex	Half-duplex	Half-duplex
Rango aproximado	10 km (urbano), 40 km (rural)	5 km (urbano), 20 km (rural)	1 km (urbano), 10 km (rural)

Seguridad	Muy alto	Muy alto (AES 128b)	Bajo (cifrado LTE)

Nota. La tabla menciona las especificaciones de las tecnologías LPWAN. Tomada de (Mekki et al., 2018).

2.2 Fundamentos de la Criptografía Ligera.

La criptografía ligera es una rama de la criptografía moderna que abarca algoritmos criptográficos destinados a utilizarse en dispositivos con recursos bajos o extremadamente bajos. La criptografía ligera no determina criterios estrictos para clasificar un algoritmo criptográfico como ligero, pero las características comunes de los algoritmos ligeros son los requisitos extremadamente bajos en cuanto a los recursos esenciales de los dispositivos de destino.

La Criptografía Ligera o Liviana estudia algoritmos que por sus propiedades matemáticas pueden ejecutarse en plataformas o dispositivos de recursos limitados, como lo son los que conforman IoT. Algunos algoritmos son diseñados para ser eficientes en un entorno determinado: hardware o software. Otros en cambio pueden serlo en cualquiera de los dos entornos. Todas las aplicaciones de la Criptografía tradicional encuentran su par en la Criptografía Liviana. Por ejemplo, Block Ciphers y Stream Ciphers. También existen algoritmos de Clave Pública. (Eterovic & Cipriano, 2017)

2.2.1 Sistemas Criptográficos Primitivos

La palabra criptografía viene del griego cripto (que significa «ocultar») y graphos (que significa «escribir»). Se podría traducir por: cómo escribir mensajes ocultos. En la antigüedad se utilizaba sobre todo durante las guerras, para comunicar estrategias, de manera que, aunque el mensajero fuera interceptado por el enemigo, el contenido del mensaje estaba a salvo.

Según la (Roa Buendía, 2013) criptografía consiste en tomar el documento original y aplicarle un algoritmo cuyo resultado es un nuevo documento. Ese documento está cifrado: no se puede entender nada al leerlo directamente. Podemos, tranquilamente, hacerlo llegar hasta el destinatario, que sabrá aplicar el algoritmo para recuperar el documento original. Las claves son combinaciones de símbolos (letras, números, signos de puntuación, etc.).

Por tanto, nuestra seguridad está expuesta a los ataques de fuerza bruta: probar todas las combinaciones posibles de símbolos. Para evitarlo tomaremos estas medidas:

- Utilizar claves de gran longitud (512-1024-2048-4096 bytes), de manera que el atacante necesite muchos recursos computacionales para cubrir todo el rango rápidamente.
- Cambiar regularmente la clave. De esta forma, si alguien quiere intentar cubrir todo el rango de claves, le limitamos el tiempo para hacerlo.
- Utilizar todos los tipos de caracteres posibles: una clave compuesta solo de números (diez valores posibles) es más fácil de adivinar que una con números y letras (36 valores posibles).
- No utilizar palabras fácilmente identificables: palabras de diccionario, nombres propios,etc.
- Detectar repetidos intentos fallidos en un corto intervalo de tiempo. Por ejemplo, la tarjeta del móvil se bloquea si fallamos tres veces al introducir el PIN.

2.2.2 Criptografía Simétrica

Los algoritmos de criptografía simétrica utilizan la misma clave para los dos procesos: cifrar y descifrar. Son sencillos de utilizar y, en general, resultan bastante eficientes (tardan poco tiempo en cifrar o descifrar). Por este motivo, todos los algoritmos,

desde la antigüedad hasta los años setenta, eran simétricos. Los más utilizados actualmente son DES, 3DES, AES, Blowfish e IDEA.

El funcionamiento es simple, el emisor quiere hacer llegar un documento al receptor. Toma ese documento y le aplica el algoritmo simétrico, usando la clave única, que también conoce el receptor. El resultado es un documento cifrado que ya podemos enviar tranquilamente.

Cuando el receptor recibe este documento cifrado, le aplica el mismo algoritmo con la misma clave, pero ahora en función de descifrar. Si el documento cifrado no ha sido alterado en el camino y la clave es la misma, el resultado será el documento original.

Este proceso se ilustra en la Figura 6, donde se muestra el intercambio de datos mediante cifrado simétrico.

Figura 3
Funcionamiento básico de la criptografía simétrica.



Nota. Representación del funcionamiento del cifrado simétrico, donde emisor y receptor utilizan la misma clave secreta para cifrar y descifrar el mensaje. Tomado de (Carrero, 2025)

2.2.3 Características de la Criptografía Ligera

La criptografía es una herramienta esencial para garantizar la integridad, la confidencialidad y la autenticidad. Debido a las limitaciones de recursos, los dispositivos IoT no pueden usar algoritmos criptográficos tradicionales (Hasan y otros, 2021). Las soluciones o algoritmos y características de seguridad son computacionalmente eficientes, consumen poca energía y son livianos. Por lo tanto, LWC se refiere a un tipo de criptografía que se ocupa de entornos restringidos (Mehta y otros, 2020).

Las características esenciales de un buen LWC son una menor complejidad, un diseño robusto, estándares de cifrado ricos, alto rendimiento, tiempo de ejecución corto y menos requerimiento de RAM. La implementación del programa (código y tamaños de RAM) requiere una implementación de hardware más pequeña, consume menos energía y tiene una fuerte inmunidad a ataques lineales y diferenciales.

Un primitivo criptográfico puede cumplir todos los requisitos de cinco categorías principales: cifrados de flujo ligeros, cifrados de bloque ligeros, autenticación ligera, funciones hash ligeras (HF) y criptografía de curva elíptica (ECC).

2.3 Algoritmo ASCON

ASCON fue diseñado por Christoph Dobraunig, Maria Eichlseder, Florian Mendel y Martin Schläffer de la Universidad de Tecnología de Graz, Infineon Technologies y la Universidad de Radboud. Es un método de cifrado y hash ligero. ASCON utiliza una única permutación ligera con modos de operación basados en Sponge y una permutación SPN (red de sustitución-permutación). (Buchanan & Maglaras, 2023)

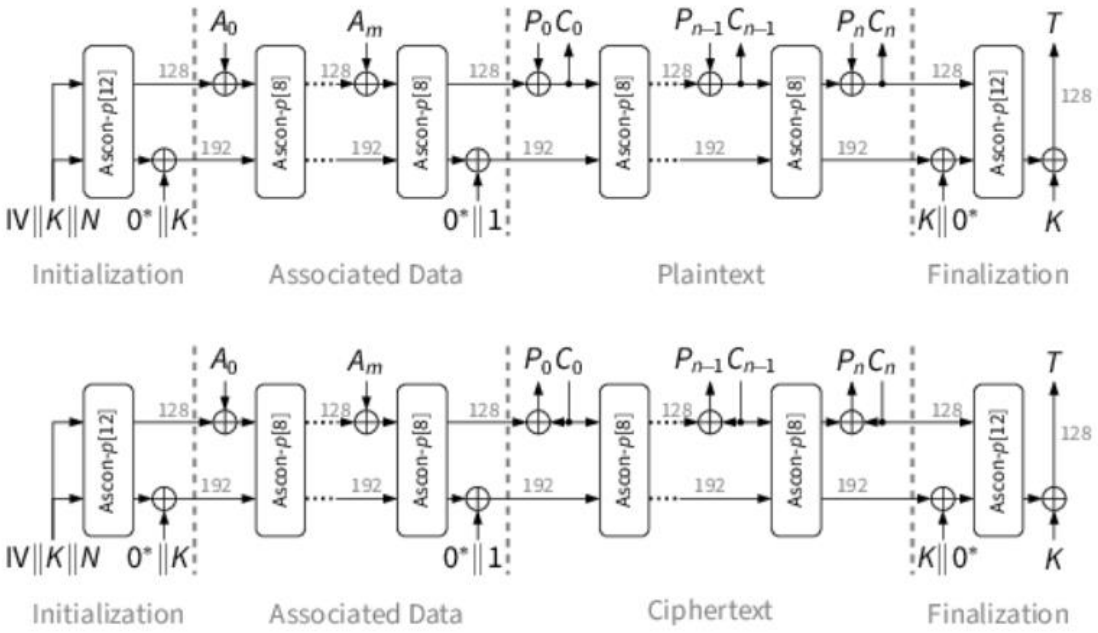
2.3.1 Funcionamiento y Estructura del Algoritmo

Citando a Dobraunig et al. (s.f.), Ascon es un algoritmo de cifrado autenticado que emplea un diseño tipo esponja (sponge), el cual opera sobre un estado interno de 320 bits. El

algoritmo ASCON-AEAD128, seleccionado por el NIST como estándar para cifrado ligero, realiza cifrado y autenticación en cuatro fases principales: inicialización, procesamiento de datos asociados, cifrado del mensaje (plaintext), y finalización. En la Figura 7 se ilustra el funcionamiento de estas fases tanto en el proceso de cifrado como el de descifrado.

Figura 4

Estructura del cifrado Ascon. (Proceso de cifrado y Proceso de descifrado.)



Nota. La figura muestra las cuatro fases principales del algoritmo Ascon-AEAD128 tanto para el proceso de cifrado (parte superior) como para el de descifrado (parte inferior). Tomado de (Dobraunig et al., s.f.).

Tras cada bloque inyectado (excepto el último bloque de texto plano), se aplica la permutación principal Ascon-p[8] al estado completo. Durante la inicialización y la finalización, se utiliza una permutación más robusta, Ascon-p[12], con más rondas.

Los siguientes parámetros resumen las especificaciones criptográficas de ASCON-AEAD128, destacando los tamaños de clave, nonce, etiqueta y el número de rondas utilizadas:

Tabla 3
Parámetros para Ascón cifrado autenticado

Cifrar	Tamaño de bit					Rondas	
	clave	nonce	tag	tasa	capacidad	p[12]	p[8]
Ascón-AEAD128	128	128	128	128	192	12	8

Nota. Se muestra los parámetros utilizados por el cifrado autenticado ASCON-AEAD128, destacando su diseño eficiente con claves, nonce y etiquetas de 128 bits. Tomado de (Dobraunig et al., s.f.).

2.3.2 Características del Algoritmo

Ascon logra alta seguridad y robustez en la práctica con una huella de área muy baja en hardware, al mismo tiempo que proporciona un buen rendimiento tanto en implementaciones de software como de hardware, particularmente para mensajes cortos. (Dobraunig et al., 2021a)

- **Cifrado y hash autenticados:** Ascon ofrece cifrado y hash autenticados con la misma permutación subyacente. Compartir un único primitivo para todos los esquemas no solo reduce los requisitos de área para las implementaciones de hardware que desean proporcionar ambos, sino que también permite restringir la base de código que se debe mantener.
- **Sencillez:** Ascon se define de forma nativa en palabras de 64 bits utilizando únicamente las funciones booleanas bit a bit y, xor, not y rot (rotación bit a bit). Esto reduce significativamente el esfuerzo de implementación del algoritmo en nuevas plataformas de destino.
- **Ligero y flexible en hardware:** Los resultados de implementación actuales muestran que Ascon ofrece excelentes características de implementación en términos de tamaño y velocidad.
- **Eficiente en Software:** Ascon está diseñado para facilitar las implementaciones de software en bits. Sus operaciones internas de 64 bits también son adecuadas

para procesadores con tamaños de palabra más pequeños y pueden aprovechar las características de paralelización y segmentación de los procesadores de alta gama.

2.4 Comparación de ASCON con otros algoritmos ligeros

A continuación se presenta una comparación entre ASCON y otros algoritmos de criptografía ligera propuestos para dispositivos con recursos limitados, entre ellos PHOTON-Beetle, Elephant, ISAP y TinyJAMBU; el análisis considera aspectos relacionados con la estructura de cada algoritmo, el uso de memoria, la eficiencia de procesamiento, el consumo de recursos y los mecanismos de seguridad empleados; esta revisión permite identificar las principales diferencias entre las alternativas estudiadas y comprender las razones por las cuales ASCON resulta adecuado para su implementación en entornos IoT.

2.4.1 Photon-B

Una de las grandes ventajas de usar un método de esponja en criptografía es que se obtiene la adición de hash al cifrado con una sobrecarga mínima en el tamaño del código y los requisitos de memoria. Si usamos un microcontrolador de 8 bits, podríamos tener solo unos cientos de bytes de ROM y un espacio similar para RAM.

Un método que se centra en crear una huella extremadamente pequeña es el método PHOTON-Beetle, que es un cifrador de bloques ligero, escrito por Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin y Kan Yasuda. Utiliza el modo Beetle, basado en esponja, con el P256 para la permutación y admite tanto cifrado autenticado (AE) como hash. PHOTON-Beetle puede optimizarse tanto para entornos con poca ROM (donde el código debe ser compacto) como para optimizar la velocidad. (Buchanan & Maglaras, 2023)

2.4.2 Elephant

Elephant es un cifrado criptográfico ligero creado por Tim Beyne, Yu Long Chen, Christoph Dobraunig y Bart Mennink. Se trata de un esquema de cifrado autenticado basado en una construcción de cifrado y luego MAC basada en nonce. Podemos proporcionar un nonce (o IV), que es el valor de sal del cifrado. Además, es compatible con el Cifrado Autenticado con Datos Asociados (AEAD), con el que podemos proporcionar datos adicionales para el cifrado. Estos datos adicionales no se envían con el cifrado ni se utilizan dentro de él, pero pueden utilizarse para autenticarlo. Una de las ventajas de Elephant es su bajo consumo, como con una permutación de 160 bits, y su paralelización. Una de las ventajas de Elephant es su bajo consumo de memoria y la posibilidad de usar una permutación de 160 bits. Este pequeño valor reduce el consumo de memoria del método. Para acelerar el proceso de cifrado, el método también puede paralelizarse. (Buchanan & Maglaras, 2023)

2.4.3 ISAP

De acuerdo con Buchanan y Maglaras (2023), ISAP es un cifrador de bloques ligero, desarrollado por Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas y Thomas Unterluggauer. Se centra en la robustez frente al análisis de potencia y los ataques de fallos, y en la presencia de un nodo para códigos de pequeño tamaño. En general, utiliza un modo basado en esponja con permutaciones de SPN. ISAP cuenta con mecanismos de protección contra ataques de fallos. ISAP utiliza un diseño de cifrado y luego MAC con dos claves para IsapMac e IsapEnc.

2.4.4 TinyJAMBU

TinyJambu fue desarrollado por Wu y Huang y es una variante de JAMBU, un esquema de cifrado autenticado por bloques basado en la permutación de claves. Según Buchanan y Maglaras (2023) el algoritmo utiliza una permutación de clave de 128 bits, un tamaño de estado de 128 bytes y un tamaño de bloque de mensaje de 32 bytes. También utiliza un registro de desplazamiento con retroalimentación no lineal para actualizar el estado.

3 CAPÍTULO III: DISEÑO DEL SISTEMA

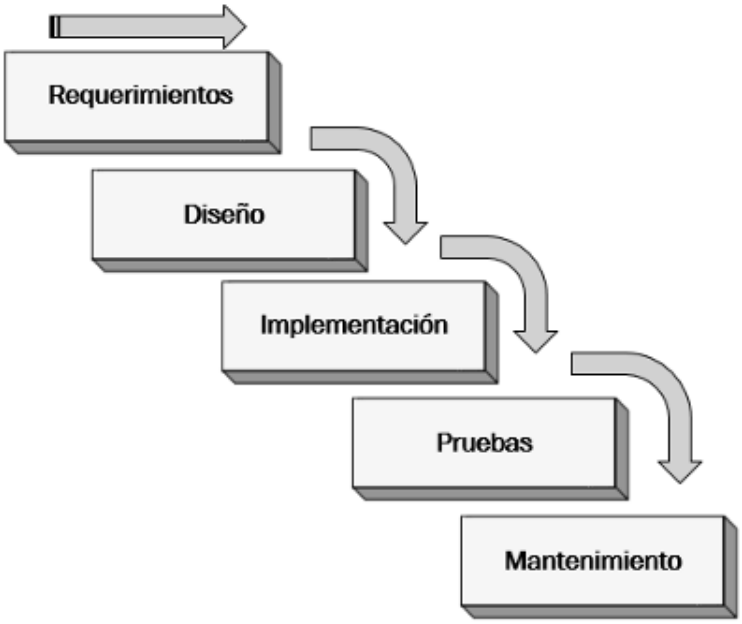
El siguiente capítulo describe el proceso de desarrollo el cual sigue un modelo en cascada caracterizado por una secuencia de fases secuenciales y bien definidas. Todos estos pasos evaluarán la viabilidad técnica y el impacto de implementar ASCON en dispositivos IoT con recursos limitados. Además, se definen indicadores clave como la latencia criptográfica, el consumo de memoria RAM y el consumo de energía, la confiabilidad de la entrega de datos y la resistencia a modificaciones no autorizadas y ataques de reproducción para evaluar el rendimiento y la seguridad del sistema. Estas mediciones permiten una evaluación detallada y objetiva de la solución implementada, asegurando que se cumplen los requisitos operativos y de seguridad marcados en las fases anteriores del proyecto.

3.1 Metodología

El modelo de la cascada, a veces llamado ciclo de vida clásico (denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en Cascada “por gravedad” hacia las siguientes fases) sugiere un enfoque sistemático y secuencial donde se ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. (Álvarez, 2015)

En la figura 8 se observa las 5 fases básicas, siendo estas, Requerimientos, Diseño, Implementación, Pruebas, y Mantenimiento. Estas etapas son esenciales para implementar el modelo; no obstante, es posible incorporar fases adicionales y subdivisiones que ayuden a detallar de forma más clara la labor a realizar.

Figura 5
Metodología en Cascada



3.2 Etapa de Requerimientos

La etapa de requerimientos establece las condiciones necesarias para orientar el desarrollo de la implementación de criptografía ligera basada en ASCON-128 dentro de la arquitectura IoT propuesta, en esta etapa se identifican las necesidades relacionadas con el funcionamiento del sistema, la seguridad de los datos y las características de los dispositivos que participan en el proceso de comunicación, considerando estos aspectos como base para definir los requerimientos que posteriormente guían la implementación y evaluación de la solución criptográfica.

3.2.1 Requerimientos del Proyecto

La infraestructura utilizada para las pruebas de seguridad en este proyecto estará basada en un sistema IoT, cuyos requerimientos serán establecidos conforme a los lineamientos de la normativa ISO/IEC/IEEE 29148:2018. Este estándar proporciona un marco detallado para la gestión de los Requerimientos de Stakeholders (StRS), Requerimientos del Sistema (SyRS) y Requerimientos de Arquitectura (ArRS). Al

emplear la arquitectura IoT de la ITU como referencia, se tendrán en cuenta tanto las capacidades de gestión como las de seguridad, las cuales son fundamentales para asegurar que el sistema sea eficiente y seguro, especialmente en el contexto de la implementación de la criptografía ASCON.

3.2.2 Nomenclatura de Requerimientos

La Tabla 4 presenta la nomenclatura utilizada para el análisis de los requisitos del proyecto, especificando las abreviaturas y sus correspondientes descripciones, lo que permite una identificación clara y uniforme de los distintos tipos de requisitos establecidos.

Tabla 4
Nomenclatura para el análisis de requisitos.

Abreviaturas	Descripción
StSR	Requerimientos de Stakeholders
SySR	Requerimientos del Sistema
SRSH	Requerimientos de Hardware y Software

Nota. Se muestra la *nomenclatura para cada uno de los requerimientos del estándar ISO/IEC/IEEE 29148 Fuente:* (ISO/IEC/IEEE, 2018).

3.2.3 Listado de Stakeholders

La Tabla 5 identifica y define a los stakeholders clave que influirán en el desarrollo del sistema. La identificación y gestión adecuadas de estos actores son esenciales para garantizar que los requisitos del sistema se alineen con sus necesidades y expectativas para el desarrollo de este proyecto.

Tabla 5
Lista de involucrados

Listado de actores involucrados		
N°	**Nombre**	**Rol**
1	Msc. Fabián Cuzme	Director del proyecto de tesis
2	Msc. Mauricio Domínguez	Asesor del proyecto de tesis
3	Sr. Alexander Arroyo	Desarrollador del proyecto

Nota. Autoría propia

3.2.4 Requerimientos Operacionales

Se identifican los Requerimientos de Stakeholders (StRS), los cuales establecen las condiciones operacionales necesarias para el correcto funcionamiento del sistema y reflejan las necesidades y expectativas de los actores involucrados. Asimismo, se incluyen requerimientos específicos de seguridad, como el cifrado y autenticación de mensajes mediante ASCON (AEAD), la gestión de nonces para prevenir ataques de repetición y la provisión y rotación de claves criptográficas, aspectos fundamentales para garantizar la protección de los datos transmitidos en el entorno IoT propuesto, tal como se muestra en la Tabla 6.

Tabla 6
Requerimientos Operacionales

Requerimientos Operacionales StSR				
Nro	Descripción	Prioridad		
		Alta	Media	Baja
StSR1	Los dispositivos deberán ser alimentados mediante baterías.	X		
StSR2	El sistema usará tecnologías de comunicación inalámbrica (Wi-Fi).	X		
StSR3	El sistema deberá emplear un protocolo de mensajería ligero que facilite el intercambio de información entre dispositivos IoT.		X	
StSR4	Los mensajes deberán estar cifrados y autenticados con ASCON (AEAD).	X		
StSR5	Gestión de nonces/contadores para prevenir ataques de repetición (replay).	X		

StSR6	Provisión y rotación de claves simétricas por dispositivo.	X		
Requerimientos de Usuario StSR				
StSR7	El usuario visualizará los datos transmitidos en un panel web desde un navegador.		X	
StSR8	Registro y auditoría de errores de autenticación y métricas de desempeño.		X	
StSR9	El retorno a condiciones normales luego de valores no esperados deberá ser notificado por mensajería.			X

Nota. Autoría propia

3.2.5 Requerimientos del Sistema

Los Requerimientos del Sistema (SySR) definen las funcionalidades, restricciones técnicas y condiciones de operación que debe cumplir el sistema IoT para garantizar un funcionamiento seguro y eficiente. Estos requerimientos han sido establecidos conforme a la norma ISO/IEC/IEEE 29148:2018, la cual recomienda que los requisitos del sistema sean claros, verificables y trazables. En el contexto de este proyecto, los SySR se enfocan principalmente en la implementación de criptografía ligera basada en ASCON, la transmisión segura de datos, así como en la evaluación del consumo energético, procesamiento y uso de memoria en dispositivos IoT con recursos limitados.

Tabla 7
Requerimientos de Interfaz

Requerimientos de Interfz SySR				
Nro	**Descripción**	**Prioridad**		
		Alta	**Media**	**Baja**
REQUERIMIENTOS DE INTERFAZ				
SySR1	La comunicación entre los nodos IoT y el gateway deberá emplear protocolos ampliamente utilizados en aplicaciones IoT.		X	
SySR2	El sistema deberá permitir la escalabilidad para la incorporación de nuevos nodos sin modificar la arquitectura base.			X
SySR3	El sistema deberá permitir la visualización de los datos procesados por parte del usuario.		X	
SySR4	El sistema deberá disponer de mecanismos básicos de control de acceso para la visualización de la información.		X	

REQUERIMIENTOS DE USO			
SySR5	La conexión nodos-Gateway debe ser estable.	X	
SySR6	El sistema deberá notificar eventos relevantes de seguridad al usuario.		X
SySR7	Los datos generados por los nodos deberán ser transmitidos de forma inalámbrica y segura hacia el sistema central.	X	
SySR8	El sistema deberá garantizar que los datos visualizados por el usuario hayan sido previamente cifrados y autenticados mediante ASCON.	X	
REQUERIMIENTOS FISICOS			
SySR9	El sistema deberá permitir la medición y comparación del consumo energético del nodo con y sin seguridad.	X	
SySR10	El sistema deberá operar correctamente en dispositivos con recursos limitados.	X	
SySR11	El sistema deberá registrar métricas físicas relevantes (consumo, uso de memoria y tiempo de ejecución) para su posterior análisis comparativo.	X	
REQUERIMIENTOS DE ESTADO			
SySR12	El sistema deberá mantener un estado válido de claves criptográficas por nodo.	X	
SySR13	El sistema deberá detectar y gestionar estados inválidos o inconsistentes.		X
REQUERIMIENTOS DE SEGURIDAD			
SySR14	El sistema deberá operar correctamente alimentado por baterías, manteniendo un consumo energético.	X	
SySR15	El sistema deberá permitir la recopilación de métricas de rendimiento para su análisis y evaluación comparativa.	X	

3.2.6 Requerimientos de Hardware y Software

En la Tabla 8 se detallan los Requerimientos de Software y Hardware (SRSH), los cuales definen las características técnicas que deben cumplir tanto el software como las plataformas hardware del sistema. Estos requerimientos priorizan el uso de lenguajes, herramientas y librerías de código abierto, así como la compatibilidad con sensores, comunicación inalámbrica y ejecución de criptografía ligera basada en ASCON. En este proyecto se emplean entornos abiertos como Arduino IDE y lenguajes de programación

como C/C++ y Python, lo que permite garantizar la transparencia, reproducibilidad y portabilidad de la implementación, conforme a los principios de la norma ISO/IEC/IEEE 29148:2018.

Tabla 8
Requerimientos de Software

REQUERIMIENTOS DE SOFTWARE SRSH				
Nro	**Descripción**	**Prioridad**		
		Alta	**Media**	**Baja**
SRSH1	El software del sistema IoT utilizara el lenguaje de programación de código abierto.	X		
SRSH2	El software deberá implementar criptografía ligera basada en ASCON (AEAD) mediante librerías de código abierto.	X		
SRSH3	El software deberá ser modular y escalable, permitiendo la integración de nuevos nodos o servicios.		X	
SRSH4	El software deberá contar con librerías que faciliten la integración de sensores y mecanismos de comunicación inalámbrica.		X	
REQUERIMIENTOS DE HARDWARE				
SRSH5	Las plataformas hardware deberán contar con soporte y disponibilidad de librerías compatibles con aplicaciones IoT y criptografía ligera.		X	
SRSH6	Las placas deberán soportar comunicación inalámbrica para transmisión de datos en entornos IoT.	X		
SRSH7	El hardware deberá permitir la ejecución de operaciones criptográficas sin requerir aceleradores dedicados.	X		
SRSH8	El sistema deberá disponer de un componente central encargado de procesar y verificar los datos recibidos.		X	
SRSH9	El sistema deberá permitir la conexión de sensores mediante interfaces digitales.	X		
SRSH10	El sistema deberá permitir la conexión simultánea de uno o más sensores sin requerir módulos externos adicionales.		X	
SRSH11	El sistema deberá soportar la lectura periódica de los sensores sin afectar la ejecución de los procesos criptográficos.	X		
SRSH12	El sistema deberá ser compatible con sensores comúnmente utilizados en entornos IoT.			X

3.2.7 Elección de Hardware y Software

Una vez definidos los requerimientos para este trabajo según los Stakeholders, se procede a la selección del hardware y software que permitirá implementar y evaluar el sistema para un correcto funcionamiento, para ello se presenta una tabla comparativa con una valoración de 0 o 1, (donde 0 no cumple con el requerimiento y 1 sí cumple con el requerimiento), con este análisis se espera obtener que los componentes reflejen resultados altos y así cumplan con los requisitos del proyecto.

3.2.7.1 Elección de Hardware

La selección del hardware se realiza considerando dos grupos principales: las placas utilizadas como nodos sensores y la placa destinada al procesamiento. Los nodos sensores serán los encargados de recolectar datos del entorno y transmitirlos de forma inalámbrica, mientras que el gateway actuará como componente central para la recepción, procesamiento y verificación de la información. Además, se considera el uso de un sensor de temperatura y humedad para la generación de datos de telemetría durante las pruebas experimentales.

Elección de Nodo Sensor

En esta sección se realiza la selección de las placas que actúan como nodos sensores del sistema IoT, considerando los requerimientos definidos previamente y aspectos relacionados con conectividad inalámbrica, compatibilidad con sensores, alimentación mediante baterías y ejecución de operaciones criptográficas mediante software.

La ESP32-WROOM-32 fue seleccionada como nodo sensor principal debido a que presenta características adecuadas para la implementación de sistemas IoT, este microcontrolador integra conectividad inalámbrica Wi-Fi, compatibilidad con sensores IoT y soporte para alimentación mediante baterías, permitiendo su utilización en entornos

de telemetría y monitoreo. La ESP32-WROOM-32 ofrece mayor capacidad de procesamiento, mayor disponibilidad de memoria y mejor estabilidad para ejecutar de forma simultánea la lectura de sensores, la comunicación inalámbrica y las operaciones criptográficas mediante ASCON, tal como se muestra en la Tabla 9.

Tabla 9
Comparativa nodo sensor placa ESP

	Requerimientos						
Hardware	**StR2**	**StSR3**	**SySR5**	**SySR7**	**SysSR12**	**SRSH8**	**Valoración Total**
ESP8266 NodeMCU	1	1	1	1	1	1	6
Arduino Nano ESP32	1	1	0	1	1	1	5
ESP32-WROOM-32	1	1	1	1	1	1	6
Elección	**ESP32-WROOM 32**						

Como se menciona en la Tabla 10 la Heltec WiFi Kit 32 V2 obtuvo la misma valoración en la comparación realizada, se seleccionó la Heltec WiFi LoRa V3 debido a que incorpora el microcontrolador ESP32-S3, una arquitectura más reciente que ofrece mejores capacidades de procesamiento, mayor disponibilidad de memoria y mejor soporte para librerías y actualizaciones. Estas características resultan beneficiosas para la implementación de mecanismos criptográficos basados en ASCON y para la evaluación

de métricas como consumo energético, uso de memoria y tiempo de procesamiento en dispositivos IoT.

Tabla 10
Comparativa nodo sensor placa Lora

	Requerimientos						
Hardware	**StR2**	**StSR4**	**SySR5**	**SySR7**	**SRSH5**	**SRSH6**	**Valoración Total**
Heltec Wi-Fi Lora V2	1	1	1	1	1	1	6
Heltec Wi-Fi LoRa V3	1	1	1	1	1	1	6
RYLR896	1	0	0	1	1	1	4
Elección	**Heltec Wi-fi Lora V3**						

Una vez concluido que el módulo LoRa Wi-fi V3 es el que tiene una puntuación más alta en cuanto a la compatibilidad de requisitos necesarios para el funcionamiento del sistema planteado en la Tabla 11 se especifican las características de este módulo LoRa.

Tabla 11
Características de Heltec Wi-fi Lora (Versión 3)

Característica	**Descripción**	**Ventajas**
Conectividad	Wi-Fi 802.11 b/g/n, Bluetooth 4.2	Conectividad eficiente para sistemas IoT y monitoreo remoto.
Procesador	Dual-core Xtensa 32-bit LX6 microprocessor, 160 MHz/240 MHz	Bajo consumo de energía y alto rendimiento en IoT.

Característica	Descripción	Ventajas
Memoria RAM	520 KB SRAM	Suficiente para manejar tareas simples y transmitir datos en tiempo real.
Puertos GPIO	34 pines GPIO, I2C, SPI, UART, ADC, DAC	Conectividad con sensores y actuadores para el sistema de limpieza.
Alimentación	3.3V DC	Bajo consumo energético, ideal para sistemas autónomos.
Sensores Integrados	Sensor de temperatura, sensor de hall, sensor de voltaje	Medición de varios parámetros sin necesidad de sensores adicionales.

La placa LILYGO T3 fue seleccionada debido a que presenta una arquitectura más adecuada lo que permite una integración más simple y enfocada en comunicación inalámbrica, lectura de sensores, además, dispone compatibilidad con librerías utilizadas en Arduino IDE y soporte para la ejecución de algoritmos criptográficos mediante software. Estas características facilitan la implementación de ASCON y la evaluación de métricas relacionadas con el consumo energético, uso de memoria y tiempo de procesamiento en dispositivos IoT, tal como se muestra en la Tabla 12.

Tabla 12
Comparativa nodo sensor placa LylyGo

	Requerimientos						
Hardware	**StR2**	**StSR4**	**SySR5**	**SySR7**	**SRSH5**	**SRSH6**	**Valoración Total**
TTGO T-Beam	1	1	1	1	1	1	6
LILYGO T3	1	1	1	1	1	1	6
Elección	**LILYGO T3**						

En la tabla 13 se presentan las características técnicas de las placas de nodo sensor LylyGo, incluyendo aspectos como la alimentación, capacidad de procesamiento, tecnología inalámbrica y microcontrolador,

Tabla 13
Características nodo sensor placa LylyGo

Característica	Descripción	Ventajas
Conectividad	Wi-Fi 802.11 b/g/n, Bluetooth 4.2	Conectividad eficiente para sistemas IoT y monitoreo remoto.
Procesador	Dual-core Xtensa 32-bit LX6 microprocessor, 160 MHz/240 MHz	Bajo consumo de energía y alto rendimiento en IoT.
Memoria RAM	520 KB LPDDR (integrado con ESP32)	Suficiente para manejar tareas simples y transmitir datos en tiempo real.

Característica	Descripción	Ventajas
Puertos GPIO	25 pines, UART, I2C, SPI, ADC, DAC	Conectividad con sensores y actuadores para el sistema de limpieza.
Alimentación	Batería de LiPo USB (5 V) / 3.7 V	Bajo consumo energético, ideal para sistemas autónomos.
Modulo Lora	SX1276 (frecuencias de 868/915 MHz)	

Elección de placa procesamiento

La Tabla 14 muestra la comparación de las placas consideradas para desempeñar la función de gateway del sistema IoT. La evaluación se realizó con base en los requerimientos definidos previamente, permitiendo seleccionar la plataforma más adecuada para la recepción, procesamiento y verificación de los datos protegidos mediante ASCON.

Tabla 14
Comparativa placa de procesamiento.

	Requerimientos						
Hardware	**StR2**	**StSR3**	**SySR5**	**SySR7**	**SRSH5**	**SRSH8**	**Valoración Total**
RASPBERRY PI 3	1	1	1	1	1	1	6
RASPBERRY PI 4	1	1	1	1	1	1	6

	Requerimientos						
Hardware	**StR2**	**StSR3**	**SySR5**	**SySR7**	**SRSH5**	**SRSH8**	**Valoración Total**
ORANGE PI ZERO 2	1	1	1	1	1	1	6
Elección	**RASPBERRY PI 3**						

La Raspberry Pi 3 fue seleccionada como gateway principal del sistema debido a que dispone de los recursos necesarios para implementar el bróker MQTT, procesar los mensajes provenientes de los nodos IoT. Aunque la Raspberry Pi 4 y la Orange Pi Zero 2 presentan una valoración similar, la Raspberry Pi 3 ofrece un equilibrio adecuado entre capacidad de procesamiento, consumo energético, disponibilidad de librerías y compatibilidad con herramientas de código abierto como Python, C y Mosquitto MQTT.

Tabla 15
Características placa de procesamiento.

Característica	Descripción	Ventajas
Conectividad	Módulo Wi-Fi BCM43143 integrado en la placa	Permite conexión a redes inalámbricas sin módulos externos
Procesador	Broadcom BCM2837 64bit ARMv7, cuatro núcleos y frecuencia de operación de 1.2 GHz	Permite ejecutar aplicaciones embebidas, servicios de red y procesos de control con buen rendimiento para sistemas IoT

Característica	Descripción	Ventajas
Memoria RAM	1 GB de memoria RAM integrada	Facilita la ejecución simultánea de procesos, interfaces gráficas ligeras y tareas de comunicación
Puertos GPIO	Conector GPIO extendido de 40 pines	Permite la conexión directa con sensores, actuadores, módulos de comunicación y circuitos de control
Alimentación	Entrada Micro USB con fuente mejorada, hasta 2.5 A	Brinda una alimentación estable para la placa y periféricos conectados

Elección de sensor de temperatura

Para la elección del sensor se tomó como base la compatibilidad con las plataformas empleadas, la existencia de librerías y la sencillez de su integración en ambientes IoT

Tabla 16
Selección sensor de temperatura

	Requerimientos					
Hardware	**SRSH4**	**SRSH9**	**SRSH11**	**SRSH12**	**SySR10**	**Valoración Total**
DTH11	1	1	1	1	1	5
DTH22	1	1	1	1	1	5
BME280	1	1	1	1	1	5
Elección	**DTH11**					

De acuerdo con la Tabla 16, los sensores evaluados cumplen con los requerimientos establecidos, por esta razón se seleccionó el DHT11, ya que presenta una integración sencilla con las plataformas utilizadas y ofrece prestaciones suficientes para las pruebas de transmisión, considerando que la investigación no busca evaluar la precisión de los sensores, sino analizar el impacto de ASCON en los dispositivos IoT.

Se presenta las principales características del sensor DHT11, junto con las ventajas que ofrece para su integración en el prototipo IoT, donde se consideran sus parámetros de medición, condiciones eléctricas, consumo de corriente e interfaz de comunicación, aspectos que permiten verificar su compatibilidad con las plataformas seleccionadas, como se indica en la Tabla 17

Tabla 17
Características sensor DTH11

Característica	Descripción	Ventajas
Rango de humedad	Opera entre 20 %RH y 90 % RH	Cubre condiciones ambientales comunes en entornos interiores
Precisión de humedad	Presenta una precisión aproximada de ±5 % RH	Resulta suficiente para generar datos de prueba sin requerir alta exactitud
Precisión de temperatura	Presenta una precisión aproximada de ±2 °C	Ofrece mediciones adecuadas para la transmisión y evaluación del sistema
Alimentación	Funciona con una tensión de 3 V a 5,5 V en corriente continua	Facilita la conexión directa con placas basadas en ESP32

Característica	Descripción	Ventajas
Interfaz de salida	Emplea una señal digital para transmitir los datos	Simplifica la conexión y programación en las plataformas seleccionadas
Corriente de medición	Consume aproximadamente 0,5 mA y puede alcanzar hasta 2,5 mA	Reduce la influencia del sensor sobre el consumo energético total del nodo

Selección Módulo de Medición Energética.

La elección del módulo de medición energética se basa en su capacidad para registrar el voltaje, la corriente y la potencia consumida por los nodos IoT, además de su compatibilidad con las plataformas seleccionadas y las librerías disponibles.

Tabla 18
Selección módulo de corriente

	Requerimientos						
Hardware	**SySR9**	**SySR11**	**SySR15**	**SRSH4**	**SRSH9**	**SRSH10**	**Valoración Total**
INA219	1	1	1	1	1	1	6
ACS712	1	1	1	1	0	1	5
Módulo sensor 25 V	0	0	0	1	0	1	2
Elección	**INA219**						

Como se muestra en la Tabla 18, se seleccionó el módulo INA219 debido a que permite obtener mediciones de voltaje, corriente y potencia mediante una interfaz digital, característica que lo convierte en la alternativa más adecuada para evaluar el consumo energético de los nodos IoT durante las pruebas con ASCON, mientras que el ACS712 se limita principalmente a la medición de corriente y el módulo sensor de voltaje de hasta 25 V únicamente permite registrar el voltaje.

La Tabla 19 presenta las características principales del módulo INA219, considerando los parámetros que permiten verificar su compatibilidad con los nodos IoT y su capacidad para registrar las variables eléctricas empleadas en la evaluación del consumo energético.

Tabla 19
Características sensor de voltaje

Característica	Descripción	Ventajas
Chip	INA219B	Un solo circuito para medición de voltaje, corriente y potencia
Voltaje de alimentación	3 V a 5 V DC	Permite su conexión con plataformas basadas en ESP32
Rango de voltaje	0 V a 26 V	Supera el nivel de tensión de baterías del prototipo
Resolución de corriente	±0,8 mA	Facilita la detección de variaciones pequeñas en el consumo
Resistencia shunt	0,1 Ω, 1 %, 2 W	Permite detectar la caída de tensión utilizada para calcular la corriente

Característica	Descripción	Ventajas
Comunicación	Interfaz I2C	Reduce el número de conexiones necesarias mediante las líneas SDA y SCL

3.2.7.2 Elección de Software

La selección de los lenguajes de programación se realizó considerando la compatibilidad con las plataformas hardware utilizadas y los requerimientos definidos para el desarrollo del sistema IoT, la Tabla 20 describe la comparación de acuerdo a los requerimientos de hardware.

Tabla 20
Comparativa de lenguajes de programación

	Requerimientos				
Hardware	**SRSH1**	**SRSH2**	**SRSH3**	**SRSH4**	**Valoración Total**
C++ (Arduino IDE)	1	1	1	1	4
Python	1	1	1	1	4
Java	1	0	1	0	2
Elección	**C++ (Arduino IDE)**				

Luego de analizar los requerimientos de software definidos para los lenguajes de programación, se determinó que C++ y Python representan las alternativas más adecuadas para el desarrollo del sistema IoT propuesto.

C++ se emplea mediante Arduino IDE para la programación de los nodos IoT, mientras que Python se utiliza en la Raspberry Pi 3 para ejecutar las funciones del gateway relacionadas con la recepción, verificación, descifrado y procesamiento de los datos transmitidos.

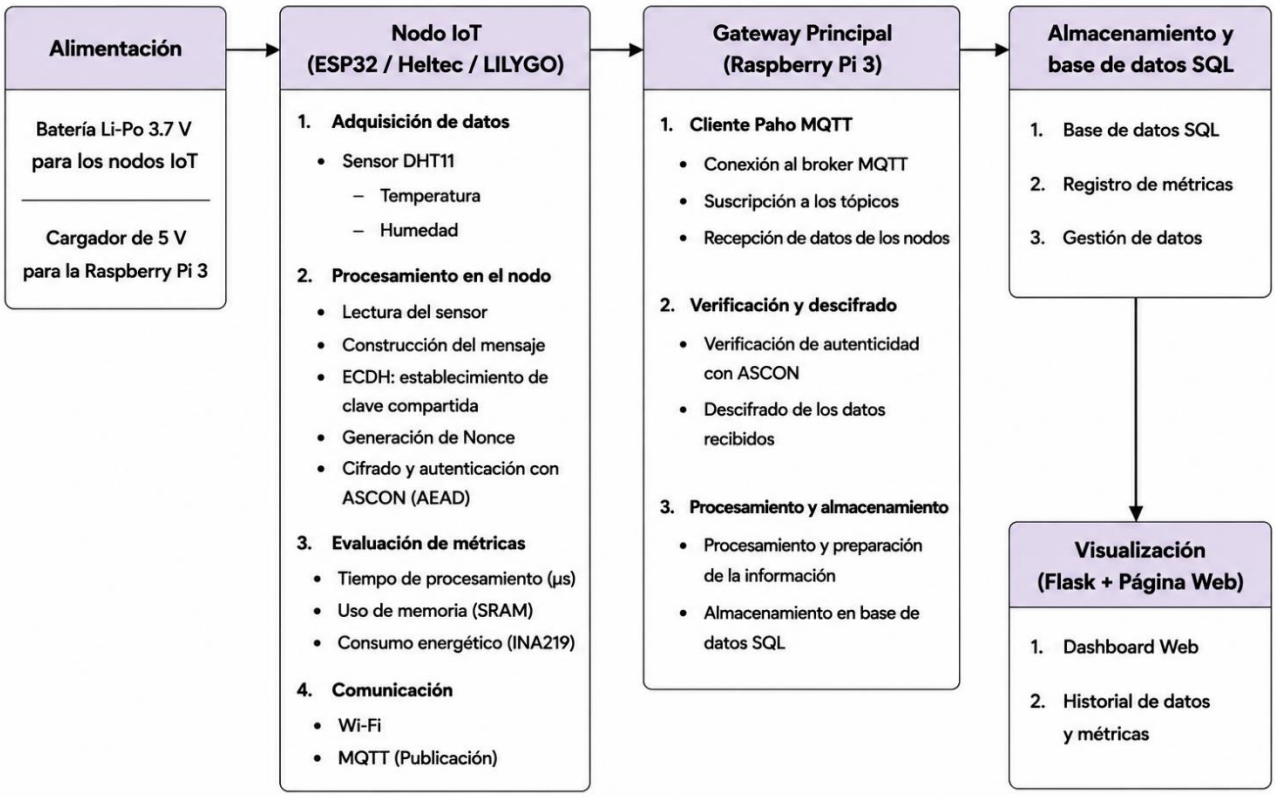
3.3 Fase de Diseño

En esta fase se describe la arquitectura del sistema, la interacción entre los nodos sensores y el gateway, el flujo de datos, el proceso criptográfico basado en ASCON y las métricas empleadas para evaluar el rendimiento, con el propósito de garantizar la transmisión segura de la información y analizar el impacto de la criptografía ligera en dispositivos IoT con recursos limitados.

3.3.1 Diagrama de bloques del Sistema

El diseño de bloques presentado en la Figura 9 muestra la estructura general del sistema IoT, desde la adquisición y procesamiento de los datos en los nodos sensores hasta su transmisión, recepción, almacenamiento y visualización. Además, representa la integración de los mecanismos de seguridad y de las métricas utilizadas para evaluar el funcionamiento del sistema.

Figura 6
Diagrama de bloques del sistema



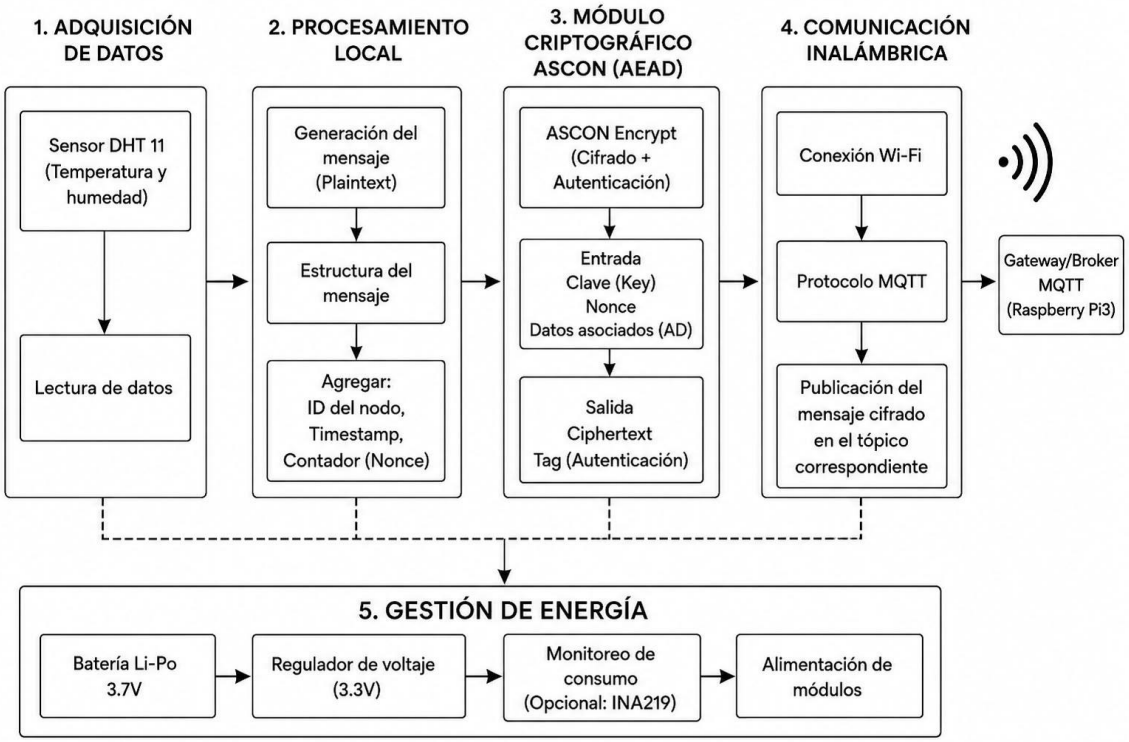
- **Bloque de alimentación:** Este bloque suministra la energía necesaria para el funcionamiento del sistema. Los nodos IoT se alimentan mediante baterías Li-Po de 3,7 V, mientras que la Raspberry Pi 3 utiliza un cargador de 5 V.
- **Bloque del nodo IoT:** Este bloque se encarga de adquirir los datos de temperatura y humedad mediante el sensor DHT11. Posteriormente, procesa la información, establece la clave compartida mediante ECDH, genera el nonce y aplica ASCON en modo AEAD para cifrar y autenticar los datos antes de su transmisión.
- **Bloque de comunicación:** Este bloque permite el envío de la información desde los nodos hacia el gateway principal mediante Wi-Fi y el protocolo MQTT.
- **Bloque del gateway principal:** La Raspberry Pi 3 recibe los mensajes mediante el cliente Paho MQTT, verifica su autenticidad, descifra los datos válidos y prepara la información para su almacenamiento.

- **Bloque de almacenamiento:** Este bloque registra en la base de datos SQL los datos obtenidos por los sensores y las métricas generadas durante las pruebas del sistema.
- **Bloque de visualización:** Este bloque presenta los datos y las métricas mediante una página web desarrollada con Flask, permitiendo consultar la información actual e histórica del sistema.

3.3.1.1 Diagrama de Bloque del Nodo Sensor.

El diagrama del nodo sensor presenta las principales etapas que intervienen en la adquisición, procesamiento, protección y transmisión de la información. En este bloque se integran la lectura de los datos, la preparación del mensaje, la aplicación de los mecanismos criptográficos y la comunicación con el gateway principal.

Figura 7
Diagrama de nodo sensor

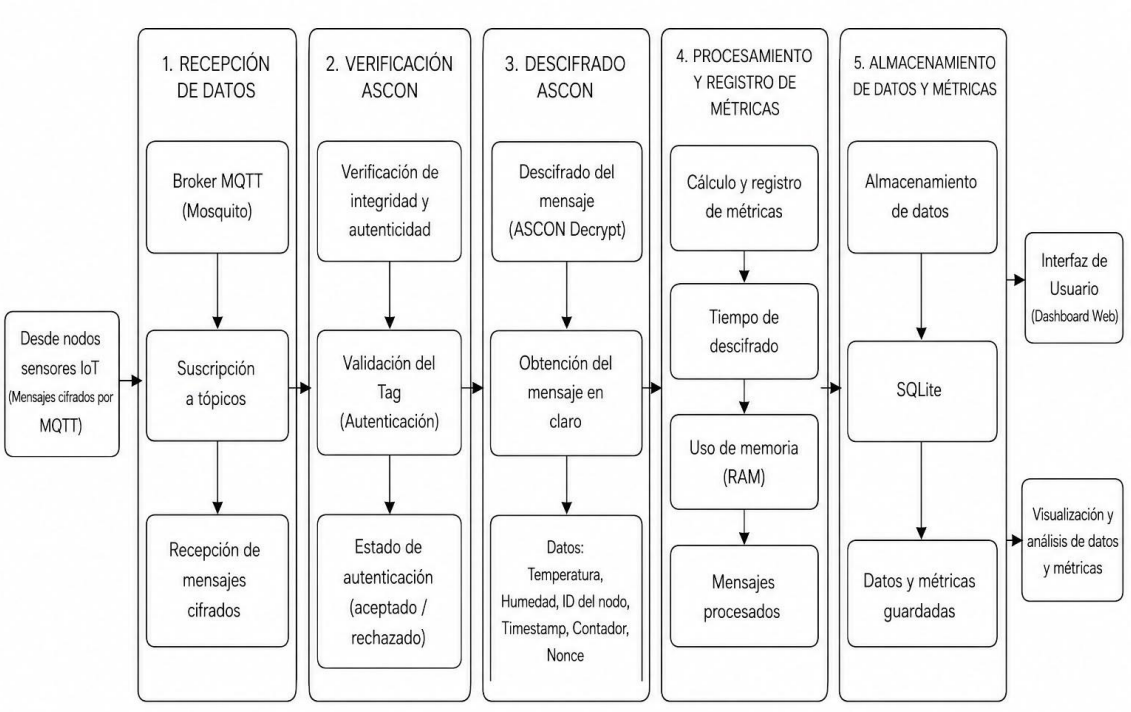


El diagrama de bloques de la Figura 10 muestra el proceso que sigue la información dentro del nodo sensor, desde la lectura de temperatura y humedad hasta su envío al gateway principal. Los datos son organizados por el microcontrolador y protegidos mediante ASCON antes de ser transmitidos por Wi-Fi utilizando MQTT. Además, el nodo funciona con una batería Li-Po, lo que permite evaluar su consumo energético durante las pruebas.

3.3.1.2 Diagrama de Bloques de Gateway Principal

El gateway constituye el punto central de coordinación entre los nodos sensores, la base de datos y la interfaz web. Su función principal es concentrar la información generada durante las pruebas y facilitar su gestión dentro del sistema IoT y es la encargada de ejecutar los servicios necesarios para el funcionamiento del prototipo y mantener la continuidad del flujo de datos.

Figura 8
Diagrama de bloques del Gateway



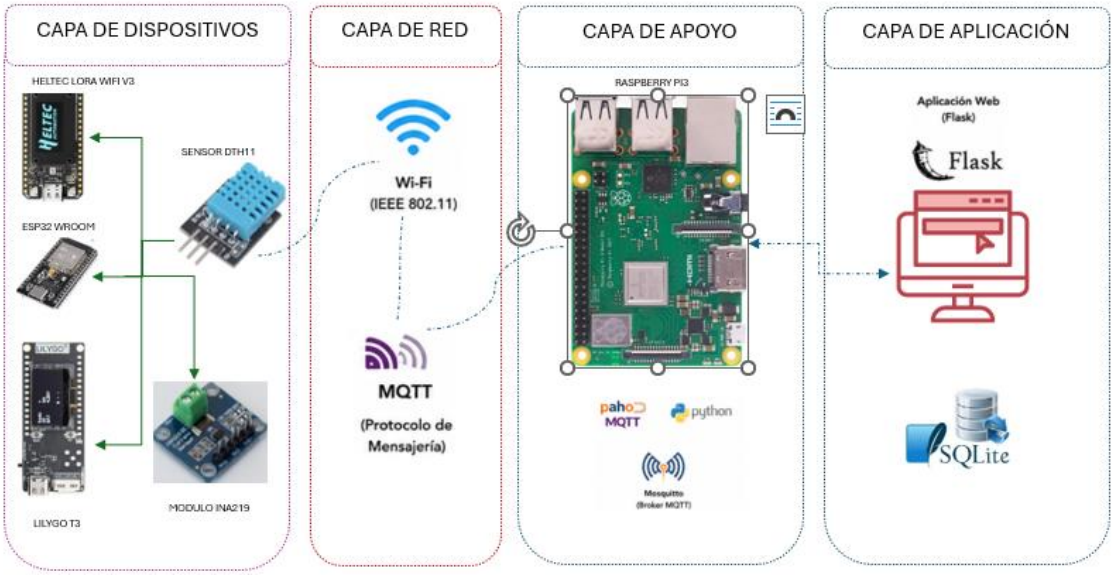
La Figura 11 presenta el diagrama de bloques del gateway principal implementado en la Raspberry Pi 3, el cual recibe los mensajes cifrados enviados por los nodos sensores mediante Wi-Fi y MQTT. Luego verifica la autenticidad e integridad de la información con ASCON y descifra únicamente los mensajes válidos. También registra métricas de procesamiento, memoria y autenticación, para finalmente almacenar los datos en la base de datos y mostrarlos en la interfaz web.

3.3.2 Arquitectura del Sistema IoT

La arquitectura del sistema IoT define la forma en que los diferentes componentes interactúan para permitir la adquisición, transmisión, procesamiento y visualización de la información. Su diseño establece la organización funcional del sistema, facilitando la integración entre los nodos sensores, el gateway, la red de comunicación y la aplicación web desarrollada para el monitoreo de los datos.

De acuerdo a la figura 12 se presenta la arquitectura funcional del sistema IoT, donde la capa de red establece la conectividad inalámbrica mediante Wi-Fi bajo el estándar IEEE 802.11 y utiliza MQTT como protocolo de mensajería, la capa de apoyo se encuentra implementada en una Raspberry Pi 3 que actúa como gateway para recibir, procesar y gestionar los mensajes provenientes de los nodos, mientras que la capa de aplicación emplea Flask para desarrollar la interfaz web y SQLite para almacenar la información generada por el sistema, conformando así un flujo continuo desde la captura de datos hasta su procesamiento, almacenamiento y visualización.

Figura 9
Arquitectura del sistema IoT



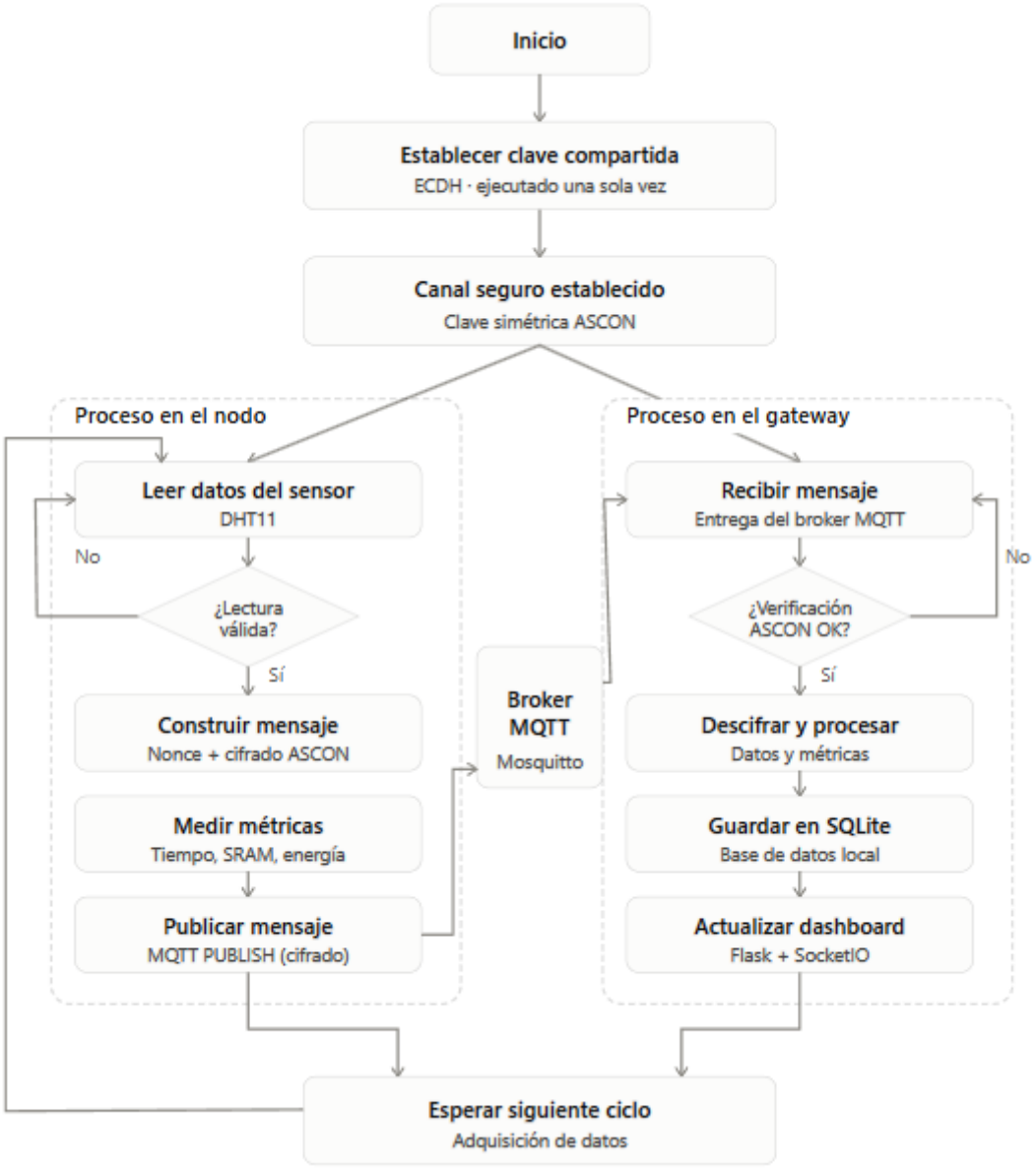
3.3.3 Diagramas de Flujo

Los diagramas de flujo representan la secuencia de operaciones del nodo sensor y del gateway principal, permitiendo comprender de forma general el funcionamiento del sistema IoT.

3.3.3.1 Diagrama de Flujo del Sistema IoT

La Figura 13 presenta el funcionamiento integrado del sistema IoT, reuniendo los procesos del nodo sensor y del gateway descritos en las Figuras 14 y 15. El flujo comienza con el establecimiento de la clave compartida mediante ECDH y continúa con el intercambio seguro de información entre ambos componentes a través de MQTT. De esta manera, el diagrama permite observar cómo se relacionan las etapas de adquisición, protección, recepción, almacenamiento y visualización dentro de un ciclo continuo de operación.

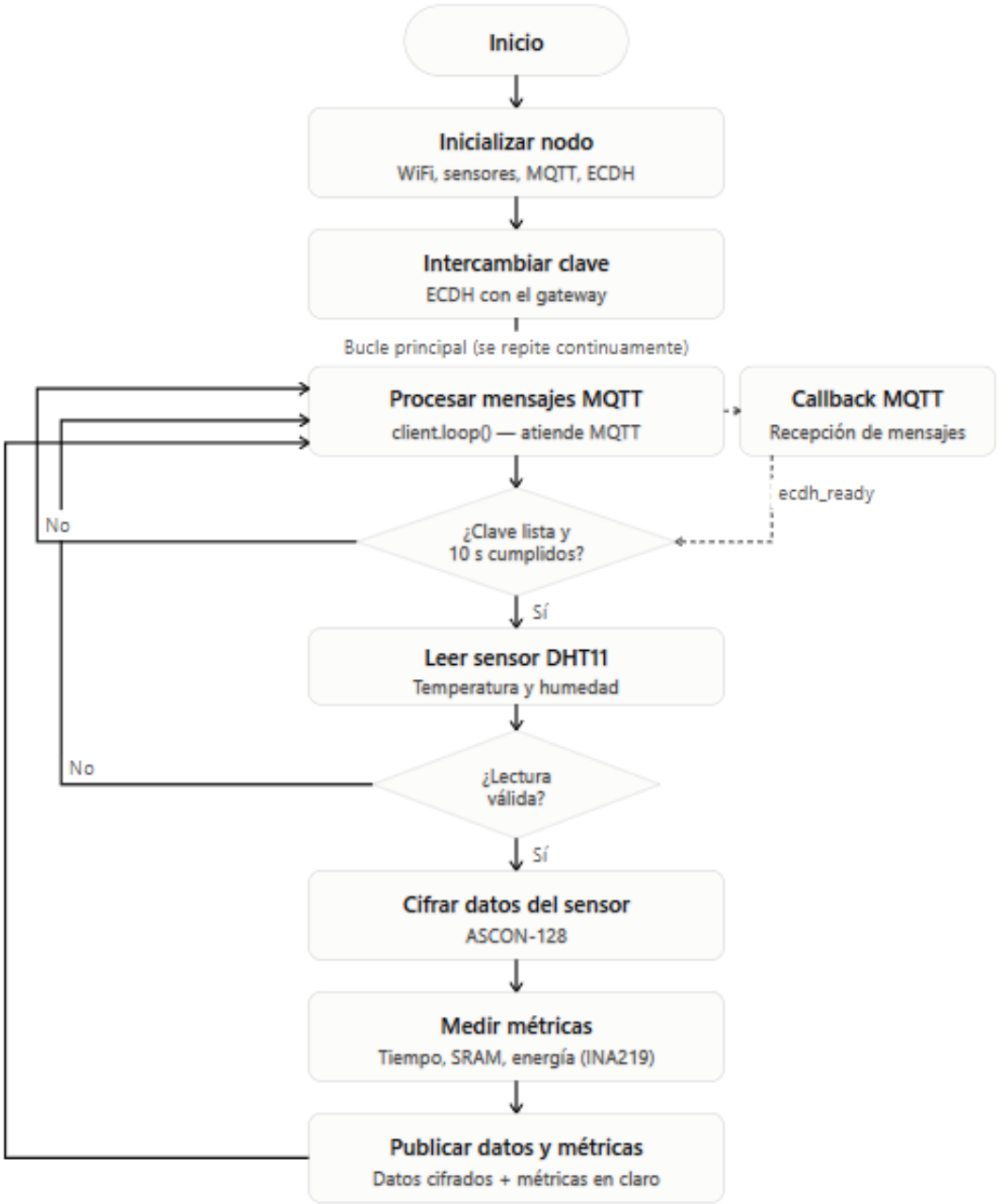
Figura 10
Diagrama de flujo del sistema



3.3.3.2 Diagrama de Flujo del Nodo Sensor

El diagrama de flujo de la Figura 14 presenta el proceso ejecutado por el nodo sensor desde la inicialización de sus componentes hasta la transmisión de la información. En esta etapa se configuran los módulos, la conexión Wi-Fi, el servicio MQTT y el mecanismo ECDH necesario para establecer la clave compartida con el gateway.

Figura 11
Diagrama de flujo del nodo



Una vez completado el intercambio de claves, el nodo entra en el bucle principal, donde verifica la conexión MQTT y realiza la lectura del sensor DHT11. Si los datos obtenidos son válidos, se construye el mensaje en formato JSON y se recopilan las métricas de consumo energético y uso de memoria.

Si el proceso se completa correctamente, se publican los datos del sensor y las métricas a través de MQTT. Finalmente, el nodo regresa al bucle principal para repetir el proceso en el siguiente ciclo de adquisición.

3.3.3.3 Diagrama de Flujo del Gateway

El gateway principal realiza la recepción y el procesamiento de la información proveniente de los nodos sensores, tal como se presenta en el diagrama de flujo de la Figura 15. Inicialmente, se configuran los servicios necesarios para el funcionamiento del sistema, incluyendo la base de datos, el cliente MQTT, Flask y SocketIO, además de la suscripción a los tópicos utilizados para la comunicación.

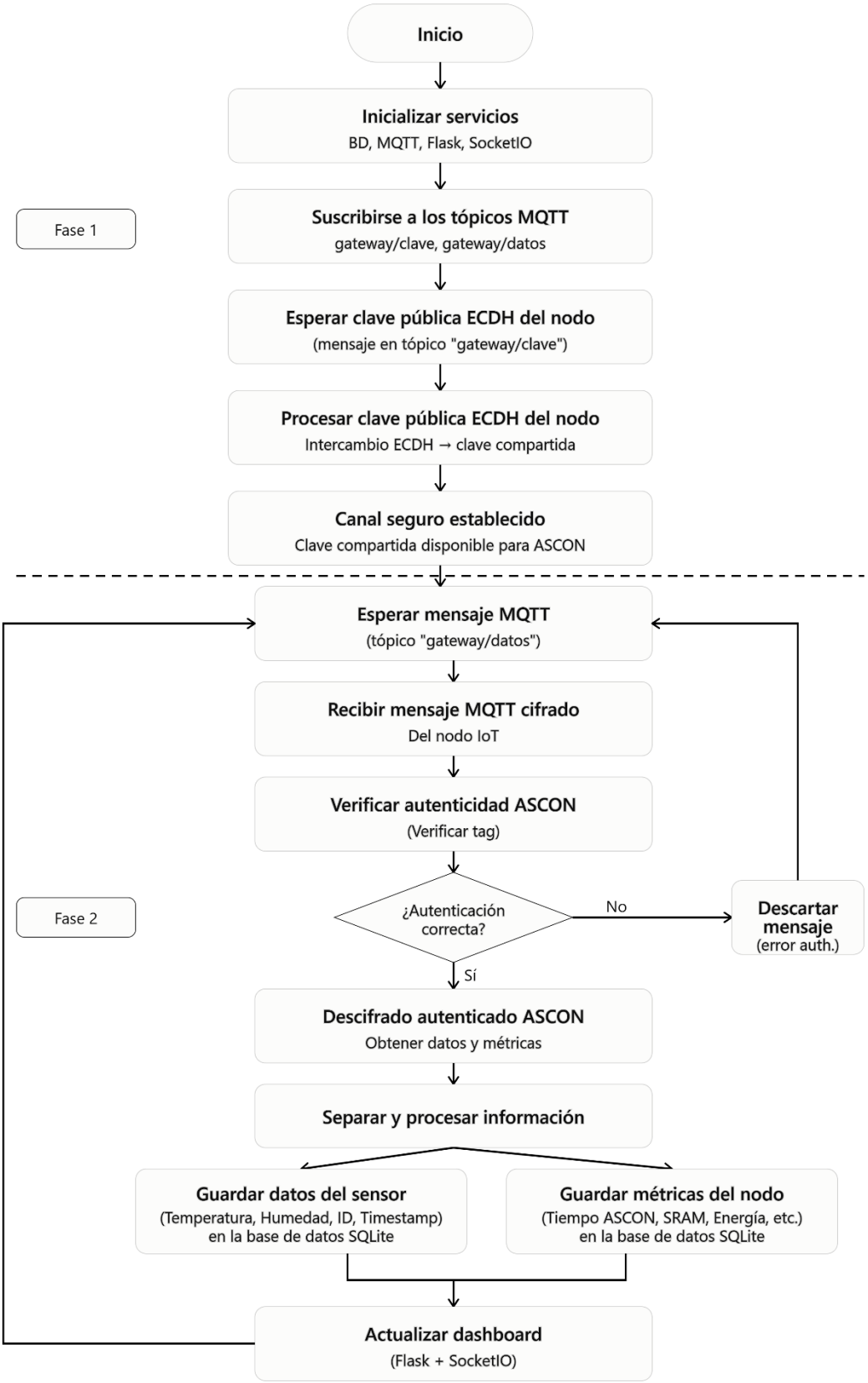
En la primera fase, el *gateway* espera la clave pública ECDH enviada por el nodo, la procesa y genera su propia clave pública para transmitirla como respuesta mediante MQTT; a partir de la clave privada local y la clave pública recibida, se calcula el secreto compartido mediante ECDH y posteriormente se aplica SHA-256 para derivar la clave de 128 bits utilizada por ASCON, permitiendo que ambos extremos dispongan del mismo material criptográfico para las operaciones posteriores.

En la segunda fase, el *gateway* permanece a la espera de nuevos mensajes MQTT y determina el proceso a ejecutar de acuerdo con el tópico recibido. Cuando el mensaje corresponde a tesis/ascon/sensor, se separa el *nonce* del contenido cifrado y se ejecuta el descifrado autenticado mediante ASCON; si la autenticación es correcta, se recuperan y almacenan los datos del sensor en la base de datos SQLite, mientras que, si la verificación falla, el mensaje es descartado para evitar el procesamiento de información no válida.

Finalmente, una vez completado el procesamiento de la información, el sistema retorna al bucle principal para continuar atendiendo nuevos mensajes, permitiendo un funcionamiento continuo del gateway dentro del sistema IoT.

Figura 12

Diagrama de flujo gateway

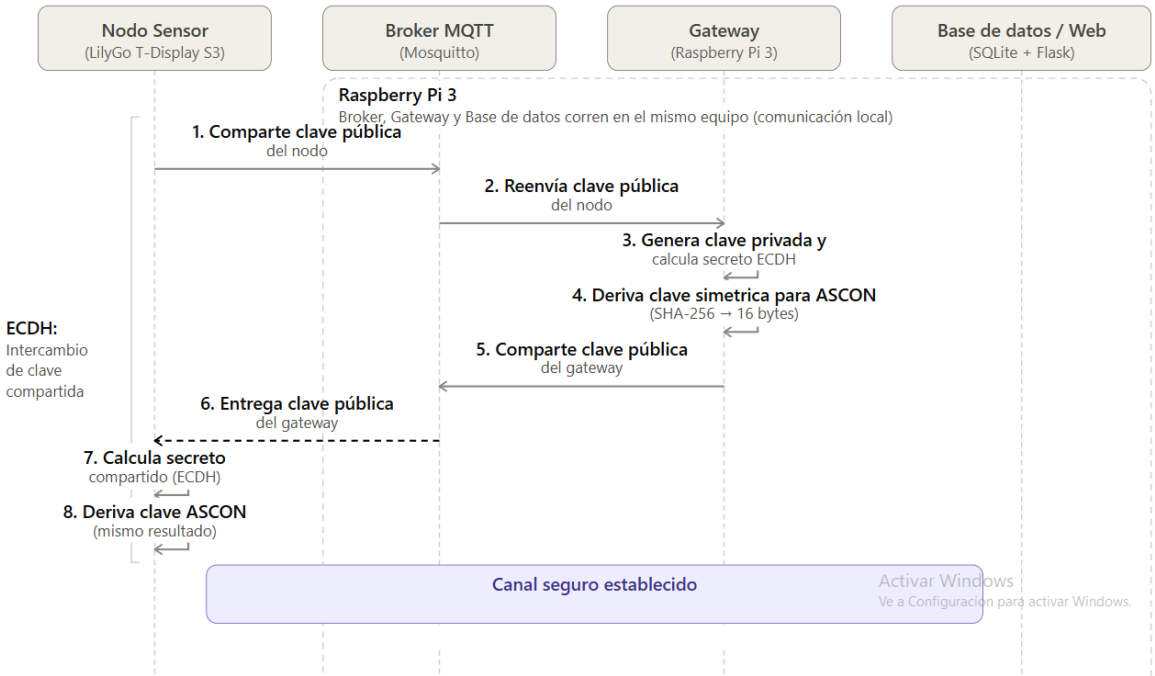


3.3.3.4 Diagrama de Secuencia

En la Figura 16 se presenta el diagrama de secuencia del sistema, el cual se organiza en dos fases para representar de manera ordenada el funcionamiento de la solución de criptografía ligera basada en ASCON-128, la primera fase corresponde al establecimiento del canal seguro mediante ECDH y la segunda comprende la generación, cifrado, transmisión, descifrado y almacenamiento de los datos obtenidos por los nodos IoT.

Figura 13

Diagrama de secuencia del sistema

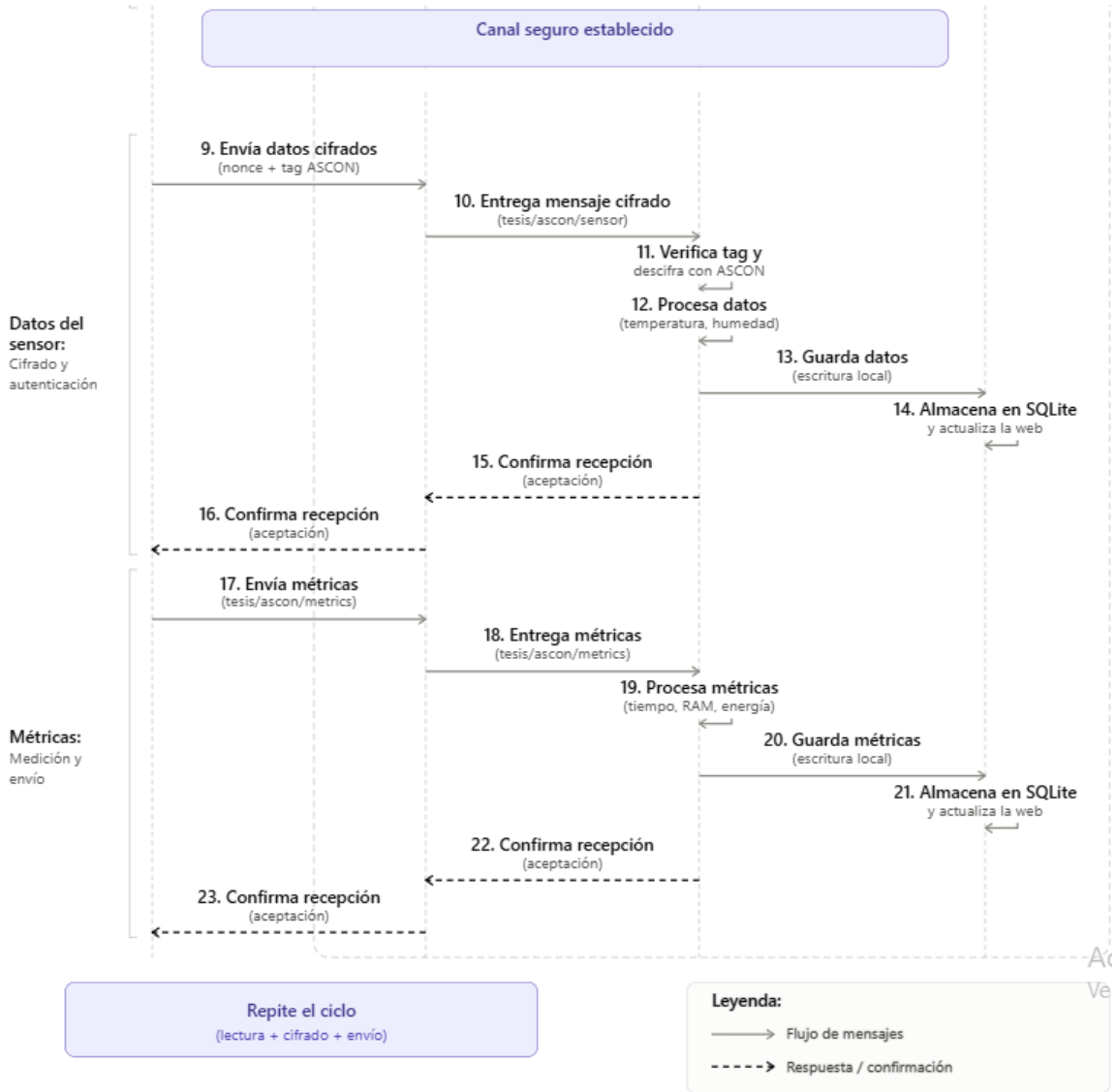


La primera fase comprende el establecimiento de la clave compartida entre el nodo IoT y el gateway mediante ECDH, el proceso inicia con el intercambio de las claves públicas entre ambos extremos y continúa con el cálculo de la clave compartida, posteriormente el gateway genera los parámetros necesarios para ASCON-128 y los comunica al nodo, una vez completado este intercambio ambos extremos disponen del material criptográfico requerido para proteger los mensajes y se establece el canal seguro para la transmisión de los datos.

La segunda fase, representada en la Figura 17, comprende el envío de los datos generados por el nodo IoT mediante MQTT, donde la información obtenida por los sensores se procesa y cifra utilizando ASCON antes de ser enviada al broker, el gateway recibe el mensaje cifrado, verifica los parámetros asociados y ejecuta el descifrado para recuperar la información original, posteriormente los datos y las métricas generadas durante el procesamiento se almacenan en la base de datos para su posterior consulta y visualización, una vez finalizado el ciclo el nodo continúa con la generación de nuevos datos y el proceso se repite bajo las condiciones establecidas.

Figura 14

Segunda fase envio de datos cifrado.

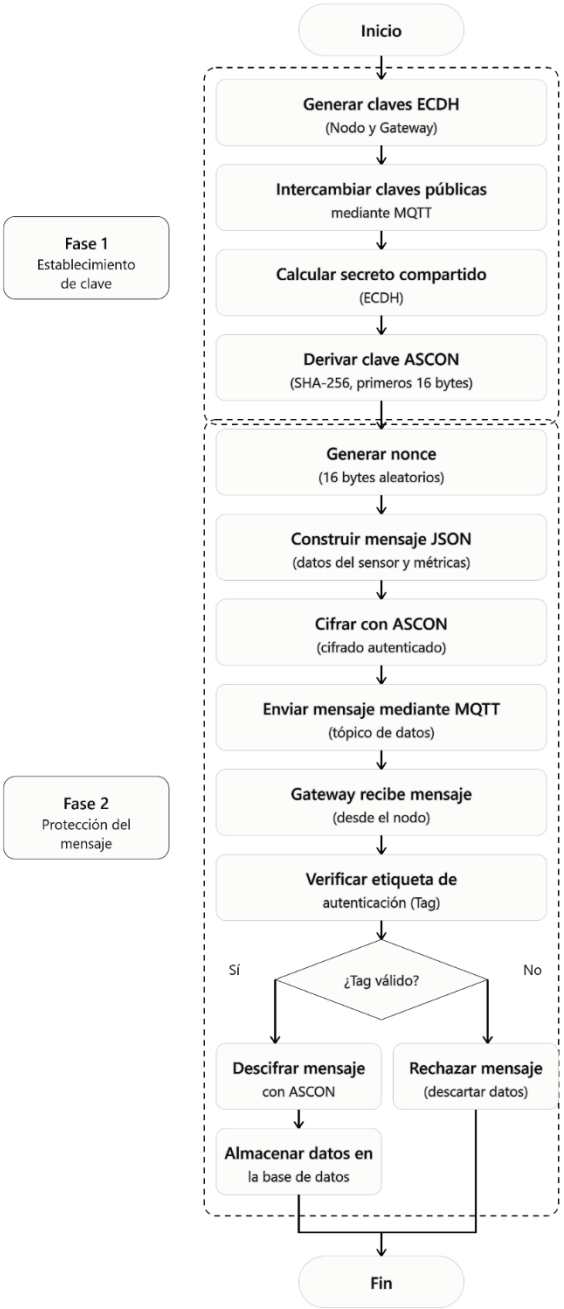


3.3.4 Diagrama del Mecanismo de Seguridad

El mecanismo de seguridad del sistema establece el proceso utilizado para proteger la información intercambiada entre el nodo sensor y el gateway. La Figura 18 representa las etapas consideradas para el establecimiento de la clave compartida mediante ECDH y la posterior protección de los datos mediante ASCON.

Figura 15

Diagrama del mecanismo de seguridad



3.3.5 Diseño de Conexiones del Sistema

Para realizar la construcción del sistema es necesario definir previamente las conexiones entre los componentes que forman parte de cada dispositivo. En este apartado se presentan los esquemas de conexión utilizados para representar la integración de las placas de desarrollo con el sensor DHT11, el módulo INA219 y la fuente de alimentación, permitiendo identificar de forma gráfica la distribución de las conexiones requeridas para el funcionamiento de los nodos.

3.3.5.1 Alimentación

Para definir la alimentación de los nodos IoT se tomaron como referencia las características eléctricas indicadas en las hojas de datos de los componentes utilizados. Estos valores permiten conocer los niveles de voltaje y consumo asociados al funcionamiento de las placas de desarrollo, tal como se observa en la Tabla 21.

Tabla 21
Consumo de energía

Dispositivo	Voltaje (V)	Consumo (mA)
Modulo INA 219	3 - 5.5V	0.7 mA
Sensor DHT11	3.5V	0.3 mA
ESP32-S3 (Heltec)	3.3V	88–91 mA RX; 283–340 mA TX
ESP32-WROOM	3.3V	112–118 mA RX; 165–239 mA TX
LilyGO T3 V1.6.1	3.3V	112–118 mA RX; 165–239 mA TX

Nota. Se especifican los valores de voltaje y consumo de referencia de los componentes utilizados en los nodos IoT, considerando los diferentes modos de operación establecidos por los fabricantes.

Los valores de consumo correspondientes a RX y TX representan las condiciones de recepción y transmisión mediante Wi-Fi, respectivamente. Estos valores se consideran

debido a que los nodos utilizan esta tecnología para establecer la comunicación mediante MQTT, por lo que el consumo puede variar de acuerdo con la actividad realizada durante el intercambio de información. Los valores indicados corresponden a condiciones establecidas por los fabricantes y se emplean únicamente como referencia para el diseño de alimentación del sistema.

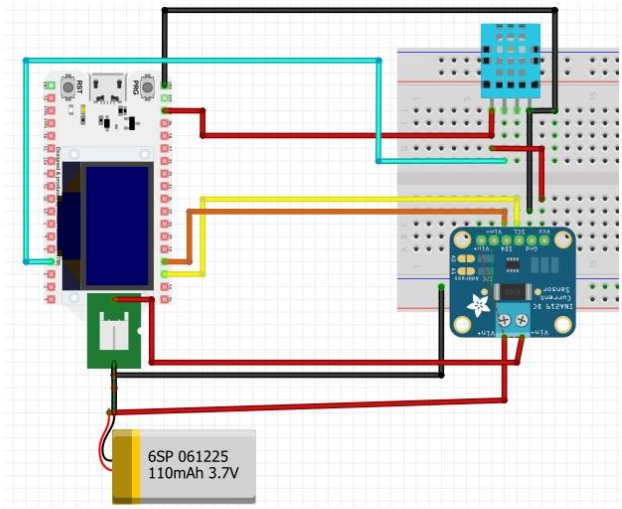
A partir de los valores de referencia presentados, para la alimentación de los nodos se seleccionaron baterías LiPo de 3,7 V y 600 mAh, debido a su compatibilidad con las placas utilizadas ya que permiten mantener una alimentación independiente durante el desarrollo de las pruebas, la capacidad seleccionada responde a las condiciones del entorno experimental debido a que el estudio no busca determinar la autonomía de los dispositivos, sino evaluar las métricas establecidas para el funcionamiento del sistema.

3.3.5.2 Esquemas de Conexiones del Sistema

Para establecer la integración física de los componentes se definieron las conexiones requeridas entre las placas de desarrollo, el sensor DHT11, el módulo INA219 y la fuente de alimentación. Los esquemas permiten identificar los pines utilizados para la adquisición de las variables ambientales, la comunicación I^2C destinada a la medición de los parámetros eléctricos y las líneas correspondientes a la alimentación de cada nodo.

Figura 16

Esquema de conexiones físicas



DISTRIBUCIÓN DE PINES Y CONEXIONES

COMPONENTE	PIN DEL MÓDULO	PIN HELTEC WiFi LoRa 32 (V3 - ESP32 S3)	FUNCIÓN	COLOR DEL CABLE
DHT11	VCC	3.3V	Alimentación del sensor	Rojo
	GND	GND	Tierra	Negro
	DATA	GPIO 4	Datos de temperatura y humedad	Cian
INA219	VCC	3.3V	Alimentación del módulo	Rojo
	GND	GND	Tierra	Negro
	SDA	GPIO 41	Comunicación I2C - SDA	Naranja
	SCL	GPIO 42	Comunicación I2C - SCL	Amarillo
BATERÍA Li-Po	(+)	VIN / BAT	Alimentación del nodo	Rojo

Nota. En la figura se observa los diferentes componentes utilizados y los respectivos pines conectados para el funcionamiento del sistema.

En la Figura 19, el nodo se encuentra conformado por la placa Heltec WiFi LoRa 32 V3, el sensor DHT11 y el módulo INA219. El DHT11 se conecta mediante una línea digital para realizar la adquisición de temperatura y humedad, mientras que el INA219 establece comunicación con la placa mediante la interfaz I²C a través de las líneas SDA y SCL, permitiendo registrar las variables eléctricas requeridas durante las pruebas.

En el esquema se observa que el módulo INA219 se incorpora en la línea de alimentación del nodo para realizar la medición de corriente y voltaje durante su funcionamiento, mientras que la batería LiPo de 3,7 V suministra la energía requerida por los componentes que forman parte del dispositivo.

Debido a que los tres nodos mantienen la misma estructura de conexión, el esquema presentado en la Figura 18 se toma como referencia general para representar la

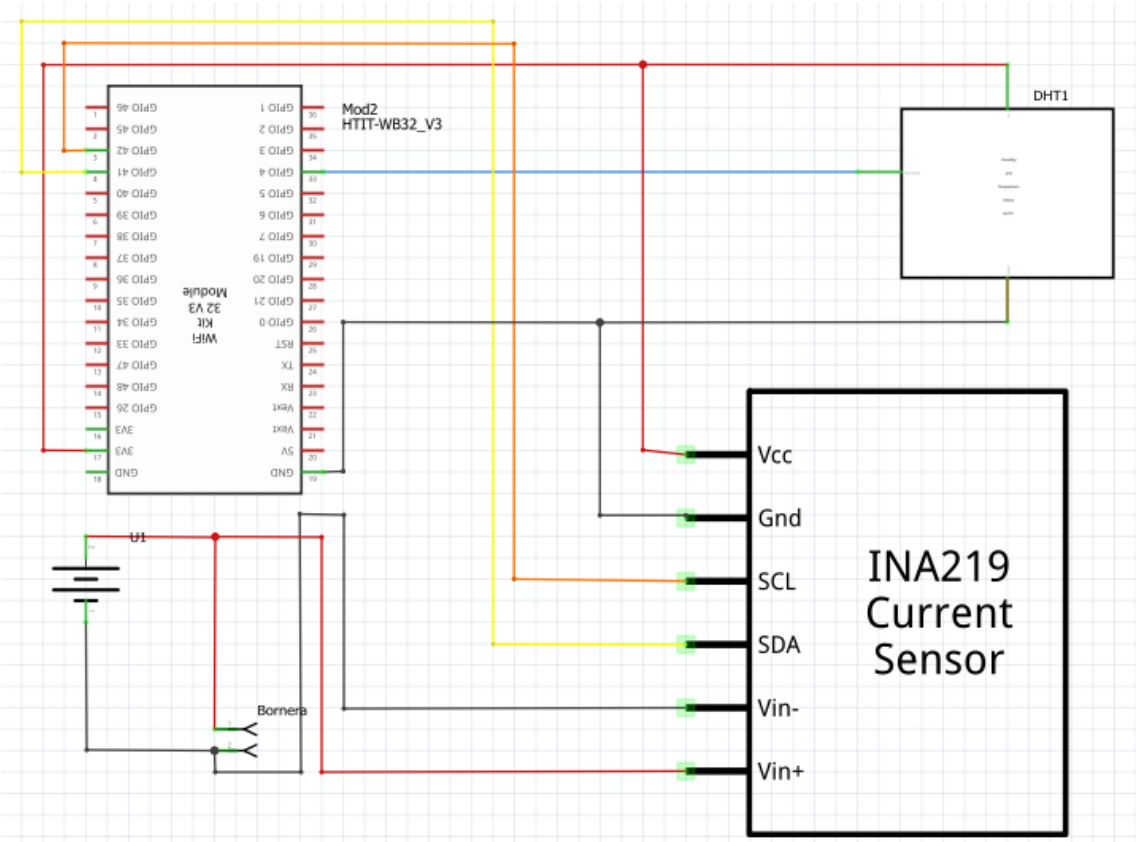
integración de sus componentes, las diferencias se encuentran principalmente en la asignación de los pines de acuerdo con cada placa de desarrollo, cuyas conexiones específicas se detallan en la Tabla 22.

3.3.5.3 Diagrama de Pines

En la Figura 20 se muestra la distribución de los pines utilizados para la conexión del sensor DHT11 que utiliza una conexión digital para la adquisición de temperatura y humedad, mientras que el módulo INA219 emplea las líneas SDA y SCL para la comunicación I²C, la asignación específica de estos pines varía según la placa utilizada en cada nodo.

Figura 17

Diagrama de pines del sistema



Nota. Se presenta como se distribuyen los pines utilizados de la placa sensor con los módulos.

Debido a que la configuración de los componentes se mantiene en los tres nodos, las principales diferencias se presentan en la asignación de los pines de acuerdo con cada

placa utilizada, mientras que las conexiones VCC y GND mantienen sus funciones de alimentación y retorno, respectivamente, tal como se representa en la figura 18, por este motivo en la Tabla 22 se detallan las conexiones utilizadas para la Heltec WiFi LoRa 32, LILYGO T3 y ESP32-WROOM.

Tabla 22
Pines de los nodos IoT

Dispositivo	Señal	LiLyGo T3	ESP32-WROOM	Heltec Lora 32
Modulo INA 219	SDA	GPIO 21	GPIO 21	GPIO 41
Modulo INA 219	SCL	GPIO 22	GPIO 22	GPIO 42
Sensor DHT11	DATA	GPIO13	GPIO 4	GPIO 4

Nota. Se presentan los pines de comunicación y de datos de cada una de las placas utilizadas.

3.3.6 Librerías

Para el desarrollo del sistema se emplearon diferentes librerías que permiten integrar las funciones necesarias para el funcionamiento de los nodos IoT y el gateway, principalmente en los procesos de seguridad, comunicación, medición y procesamiento de la información, la selección de estas librerías se realizó de acuerdo con los requerimientos de cada dispositivo

- ✓ **Mbedtls:**

Es una biblioteca que proporciona diferentes funciones criptográficas y está orientada a su implementación en sistemas embebidos debido a su reducido tamaño de código. En el sistema propuesto se utiliza principalmente para realizar el intercambio de claves mediante ECDH, generar los valores necesarios durante el proceso criptográfico y aplicar SHA-256 para obtener la clave que posteriormente se emplea en el cifrado mediante ASCON-128.

- ✓ **PubSubClient:**

Es una librería cliente orientada a la comunicación mediante MQTT, protocolo de mensajería ligero utilizado en dispositivos con recursos limitados (O'Leary, 2026). En el sistema propuesto se emplea para publicar y recibir mensajes entre los nodos IoT y el gateway, permitiendo el intercambio de claves públicas, datos cifrados y métricas del sistema.

- ✓ **Adafruit_INA219:**

Esta librería permite gestionar el funcionamiento del módulo INA219 y obtener las variables eléctricas registradas durante su operación. En el sistema propuesto se utiliza para medir el voltaje, la corriente y la potencia consumida por los nodos IoT, permitiendo obtener los datos necesarios para evaluar su comportamiento energético durante las pruebas.

- ✓ **Esp_timer:**

Esta librería proporciona funciones de temporización de alta resolución para los microcontroladores ESP32, permitiendo obtener mediciones con resolución de microsegundos. En el sistema propuesto se utiliza para registrar el tiempo antes y después de ejecutar el cifrado con ASCON-128, con el fin de calcular el tiempo de procesamiento requerido por el algoritmo

- ✓ **Esp_heap_caps:**

Esta librería permite consultar y gestionar la memoria disponible en los microcontroladores ESP32 de acuerdo con sus características de almacenamiento. En el sistema propuesto se utiliza para obtener información sobre la memoria SRAM disponible durante la ejecución de las pruebas, permitiendo determinar el uso de memoria generado por el procesamiento y cifrado de los datos mediante ASCON-128.

- ✓ **Paho.mqtt.client:**

Esta librería permite implementar la comunicación mediante el protocolo MQTT en aplicaciones desarrolladas en Python, proporcionando funciones para publicar y recibir mensajes a través de diferentes tópicos. En el sistema propuesto se utiliza en el gateway para recibir la información enviada por los nodos IoT y gestionar el intercambio de datos requerido durante la comunicación mediante MQTT.

- ✓ **Cryptography:**

Esta librería proporciona diferentes funciones criptográficas para aplicaciones desarrolladas en Python, incluyendo operaciones relacionadas con criptografía asimétrica. En el sistema propuesto se utiliza en el gateway para realizar las operaciones correspondientes al intercambio de claves mediante ECDH, permitiendo obtener el secreto compartido requerido para establecer la clave utilizada.

- ✓ **Ascon128:**

Es una librería que permite implementar el algoritmo de cifrado autenticado ASCON-128 en plataformas compatibles con Arduino. En el sistema propuesto se utiliza en los nodos IoT para cifrar los datos adquiridos y generar la etiqueta de autenticación antes de realizar su transmisión hacia el gateway.

3.4 Fase de Implementación

En esta fase se presenta la implementación de los componentes definidos durante el diseño del sistema, considerando la integración del mecanismo de seguridad en los nodos IoT y el gateway, la adquisición y transmisión de los datos, el procesamiento de la información recibida y el almacenamiento de los resultados, de manera que se establezca el funcionamiento conjunto de los elementos que conforman la solución propuesta.

3.4.1 Implementación del Mecanismo de Seguridad

El mecanismo de seguridad se implementó mediante la integración de ECDH y ASCON en el nodo sensor y el gateway principal. ECDH se empleó para establecer el secreto compartido entre ambos dispositivos, mientras que la clave obtenida se utilizó posteriormente en ASCON para proteger la información transmitida mediante MQTT.

3.4.1.1 Establecimiento de la clave mediante ECDH

La primera etapa del mecanismo de seguridad corresponde al establecimiento de una clave compartida entre el nodo sensor y el gateway mediante el algoritmo Elliptic Curve Diffie-Hellman (ECDH). Este proceso permite que ambos dispositivos obtengan el mismo secreto compartido sin transmitir directamente dicha información por el canal MQTT, reduciendo el riesgo de exposición durante el intercambio inicial.

En el nodo sensor se realiza inicialmente la configuración del contexto criptográfico mediante la biblioteca Mbed TLS, empleando la curva elíptica SECP256R1. Para ejecutar el intercambio ECDH, el nodo genera una pareja de claves conformada por una clave privada y una clave pública. La clave privada permanece almacenada dentro del dispositivo, mientras que la clave pública es utilizada para realizar el intercambio con el gateway.

La generación de la clave pública del nodo se realiza mediante la función mbedtls_ecdh_make_public(), como se muestra en el siguiente fragmento de código:

```
ret = mbedtls_ecdh_make_public(
    &ecdh,
    &node_public_key_len,
    node_public_key,
    sizeof(node_public_key),
    mbedtls_ctr_drbg_random,
    &ctr_drbg
);
```

Esta función permite obtener la clave pública asociada a la clave privada generada dentro del contexto ECDH. El resultado se almacena en la variable `node_public_key`, mientras que `node_public_key_len` conserva la longitud de la información generada.

Por parte del gateway, una vez recibida la clave pública del nodo sensor, se genera una clave privada local utilizando la misma curva criptográfica definida para el intercambio ECDH.

```
gateway_private_key = ec.generate_private_key(
    ec.SECP256R1()
)
```

La clave privada generada permanece dentro del gateway y no es transmitida por la red. A partir de esta clave privada y de la clave pública recibida desde el nodo sensor, el gateway calcula el secreto compartido mediante la operación ECDH:

```
shared_secret = gateway_private_key.exchange(
    ec.ECDH(),
    node_public_key
)
```

El resultado obtenido corresponde al secreto compartido utilizado por ambos dispositivos. De manera equivalente, el nodo sensor realiza el mismo cálculo utilizando su clave privada y la clave pública del gateway, obteniendo un valor idéntico sin necesidad de intercambiarlo directamente.

3.4.1.2 Derivación de la clave ASCON

Una vez obtenido el secreto compartido mediante el protocolo ECDH, tanto el nodo sensor como el gateway realizan el proceso de derivación de la clave que será utilizada por el algoritmo ASCON. Para ello, ambos dispositivos aplican la función hash SHA-256 sobre el secreto compartido y utilizan los primeros 16 bytes del resultado para generar una clave de 128 bits. En el nodo sensor, la biblioteca Mbed TLS emplea la

función `mbedtls_sha256()` para calcular el resumen criptográfico del secreto compartido generado previamente mediante ECDH.

```
uint8_t hash[32];
mbedtls_sha256(
    secret_bytes,
    secret_len,
    hash,
    0
);
memcpy(key, hash, sizeof(key));
```

La variable `secret_bytes` contiene el secreto compartido obtenido durante la negociación ECDH, mientras que el arreglo hash almacena el resultado de 32 bytes generado por SHA-256. Posteriormente, mediante la función `memcpy()`, se copian los primeros 16 bytes del resumen criptográfico hacia la variable key, la cual será utilizada posteriormente como clave de cifrado para ASCON.

De manera equivalente, el gateway realiza la misma operación utilizando la función hash SHA-256 sobre el secreto compartido obtenido durante el intercambio ECDH

```
hash_key = hashlib.sha256(shared_secret).digest()
ascon_key = hash_key[:16]
```

En este caso, la función `hashlib.sha256`() genera el resumen criptográfico del secreto compartido y la operación `hash_key[:16]` selecciona los primeros 16 bytes del resultado para obtener la clave simétrica utilizada por ASCON. Debido a que ambos dispositivos poseen el mismo secreto compartido ECDH y aplican el mismo proceso de derivación, la clave obtenida en el nodo sensor y en el gateway es equivalente, por lo cual esta clave será utilizada posteriormente durante el proceso de cifrado y autenticación de los mensajes MQTT, garantizando la confidencialidad e integridad de los datos enviados desde los nodos sensores hacia el gateway.

3.4.1.3 Protección del mensaje mediante ASCON

Una vez obtenida la clave criptográfica derivada mediante ECDH, el nodo sensor ejecuta el proceso de protección de la información utilizando el algoritmo ASCON. Este algoritmo permite realizar cifrado autenticado, proporcionando confidencialidad mediante la transformación del mensaje original en texto cifrado y autenticidad mediante la generación de una etiqueta de verificación o tag.

Antes de ejecutar el cifrado, el nodo sensor prepara los datos obtenidos desde los sensores en un formato estructurado para su transmisión mediante MQTT. Posteriormente, la información es procesada utilizando la clave ASCON derivada previamente, un nonce generado aleatoriamente y el contenido original del mensaje.

```
int ascon_encrypt(
const uint8_t* key_in,
const uint8_t* iv_
in, const uint8_t* plaintext_in,
size_t plen,
uint8_t* ciphertext_out
)
```

La función `ascon_encrypt()` recibe la clave criptográfica, el vector de inicialización y el mensaje original como parámetros de entrada. Inicialmente, se configura el algoritmo ASCON mediante la carga de la clave simétrica:

```
ascon.setKey(key_in, 16);
```

Posteriormente, se establece el nonce utilizado durante la operación criptográfica:

```
ascon.setIV(iv_in, 16);
```

El nonce es generado previamente mediante la función `generate_iv()`, donde se obtiene un valor aleatorio de 16 bytes utilizando el generador interno del ESP32. Este valor permite que cada proceso de cifrado sea diferente, incluso cuando se utiliza la misma clave y contenido del mensaje.

Una vez configurados estos parámetros, se ejecuta el cifrado de los datos mediante:

```
ascon.encrypt(
    ciphertext_out,
    plaintext_in,
    plen
);
```

La etiqueta es agregada al mensaje cifrado y permite que el gateway verifique la integridad y autenticidad de la información recibida.

Por lo tanto, el mensaje transmitido mediante MQTT está compuesto por los datos cifrados y el tag de autenticación generado por ASCON, permitiendo que únicamente el gateway que posee la misma clave simétrica pueda verificar y recuperar la información original.

El código fuente completo utilizado para la implementación de ASCON-128 en los nodos IoT, incluyendo las funciones de cifrado, generación del nonce y obtención de las métricas durante las pruebas, se presenta en el Anexo F.

3.4.1.4 Verificación y descifrado en el Gateway

Una vez recibido mediante MQTT el mensaje protegido con ASCON, el gateway ejecuta el proceso de verificación y descifrado utilizando la clave ASCON previamente derivada mediante ECDH. Esta etapa permite recuperar la información transmitida por el nodo sensor y comprobar que los datos recibidos no fueron modificados durante el proceso de comunicación.

El proceso de descifrado requiere la misma clave utilizada durante el cifrado, además del nonce y la etiqueta de autenticación generada por ASCON. Estos parámetros permiten que el gateway valide la autenticidad del mensaje antes de recuperar la información original. A continuación, se presenta el proceso de descifrado:

```
iv = raw[:16]
ciphertext_tag = raw[16:]
plaintext = ascon.decrypt(
    ASCON_KEY[16],
    iv,
    b"",
    ciphertext_tag,
    variant="Ascon-128"
)
if plaintext is None:
    return None
```

Inicialmente, el gateway separa el nonce recibido y el contenido cifrado del mensaje. Posteriormente, mediante la función `ascon.decrypt`(), se proporciona la clave ASCON, el nonce y el mensaje protegido para realizar la verificación de autenticidad y el proceso de descifrado.

Si la etiqueta de autenticación es válida, ASCON permite recuperar el mensaje original transmitido por el nodo sensor. En caso contrario, la función indica que la autenticación ha fallado y el mensaje es descartado para evitar el procesamiento de información no válida.

El código fuente completo implementado en la Raspberry Pi 3 para el establecimiento del secreto compartido mediante ECDH, la derivación de la clave ASCON, la recepción de mensajes mediante MQTT, el descifrado mediante ASCON-C y el procesamiento de la información se presenta en el Anexo G.

3.4.2 Implementación de la Comunicación

La comunicación entre los nodos sensores y el gateway se realiza mediante el protocolo MQTT sobre una red Wi-Fi; este mecanismo permite transmitir el intercambio de claves públicas empleados en ECDH, los mensajes protegidos mediante ASCON y las métricas del sistema bajo un modelo de publicación y suscripción.

Tabla 23
Tópicos MQQT del sistema IoT

Tópico MQQT	Publicador	Suscriptor	Descripción
tesis/ascon/key/node	Nodo sensor	Gateway	Envía la clave pública del nodo para iniciar el establecimiento del secreto compartido mediante ECDH.
tesis/ascon/key/gateway /<node>	Gateway	Nodo sensor	Envía la clave pública del Gateway como respuesta al nodo para completar el intercambio de claves ECDH.
tesis/ascon/sensor	Nodo sensor	Gateway	Transmite los datos del protegidos mediante ASCON
tesis/ascon/metrics	Nodo sensor	Gateway	Envía las métricas de evaluación del nodo, como tiempo de cifrado, memoria y consumo energético.

En la Tabla 23 se muestra cómo el protocolo MQTT emplea los tópicos para intercambiar las claves públicas necesarias para el establecimiento del secreto compartido, así como para transmitir los datos protegidos y las métricas generadas por los nodos sensores, además que permite separar cada tipo de información en un tópico específico, facilitando el procesamiento y la administración de los mensajes.

3.4.3 Implementación de Métricas de Evaluación

Para evaluar el desempeño del sistema propuesto, se definen las métricas que serán utilizadas durante la etapa experimental: estas métricas permiten analizar el comportamiento del algoritmo ASCON en los dispositivos IoT desde diferentes aspectos de rendimiento.

3.4.3.1 Medición de Consumo Energético

La medición del consumo energético se implementa mediante el módulo INA219, encargado de registrar las variables eléctricas asociadas al funcionamiento del nodo

durante las pruebas. Los valores obtenidos permiten disponer de información sobre el voltaje, la corriente, la potencia.

```
void iniciarINA219() {
  if (!ina219.begin()) {
    Serial.println("ERROR: No se detecta INA219");
    ina_ok = false;
    return;
  }
  ina219.setCalibration_32V_2A();
  ina_ok = true;
  unsigned long now = millis();
  ina_start_ms = now;
  ina_last_ms = now;
  lastPowerSample = now;
  Serial.println("INA219 iniciado correctamente");
}
void leerINA219(unsigned long now) {
  if (!ina_ok) {
    return;
  }
  ina_shunt_mv = ina219.getShuntVoltage_mV();
  ina_bus_v = ina219.getBusVoltage_V();
  ina_current_ma = ina219.getCurrent_mA();
  ina_power_mw = ina219.getPower_mW();
  ina_load_v = ina_bus_v + (ina_shunt_mv / 1000.0);
  ina_current_abs_ma = fabs(ina_current_ma);
  float power_abs_mw = fabs(ina_power_mw);

  if (ina_last_ms > 0 && now >= ina_last_ms) {
    double dt_h = (double)(now - ina_last_ms) / 3600000.0;
    acc_mAh += ina_current_abs_ma * dt_h;
    acc_mWh += power_abs_mw * dt_h;
  }
  ina_last_ms = now;
  double elapsed_h = (
    (double)(now - ina_start_ms) / 3600000.0
  );
  if (elapsed_h > 0.0) {
    avg_current_mA = acc_mAh / elapsed_h;
    if (avg_current_mA > 0.5) {
      est_runtime_h = (
        BATTERY_CAPACITY_MAH / avg_current_mA
      );
    } else {
      est_runtime_h = 0.0;
    }
  }
}
```

De acuerdo con el código descrito anteriormente, el módulo INA219 se inicializa mediante comunicación I²C y posteriormente obtiene las mediciones de voltaje, corriente y potencia del nodo, permitiendo registrar el comportamiento energético durante la ejecución del sistema.

3.4.3.2 Medición del Tiempo de Procesamiento

La medición del tiempo de procesamiento se realiza mediante una prueba aislada del algoritmo ASCON, con el propósito de medir el tiempo requerido para realizar el cifrado de los datos.

```
uint64_t medirASCONPromedio(
  size_t plen,
  int repeticiones,
  int* ciphertext_len_final
) {
  uint64_t suma_us = 0;
  int ciphertext_len = -1;
  for (int i = 0; i < repeticiones; i++) {
    generate_iv();
    int64_t inicio_us = esp_timer_get_time();
    ciphertext_len = ascon_encrypt(
      key,
      iv,
      plaintext,
      plen,
      ciphertext
    );
    int64_t fin_us = esp_timer_get_time();
    if (ciphertext_len <= 0) {
      *ciphertext_len_final = ciphertext_len;
      return suma_us;
    }
    suma_us += (uint64_t)(fin_us - inicio_us);
    ascon_control ^= ciphertext[0];
  }
  *ciphertext_len_final = ciphertext_len;
  return suma_us;
}
```

El tiempo se registra en microsegundos mediante esp_timer_get_time() con el fin de obtener valores más precisos. Estos datos se acumulan durante 1000 ejecuciones para determinar el tiempo promedio de procesamiento. La cantidad de repeticiones utilizada fue establecida mediante pruebas preliminares realizadas de forma progresiva hasta alcanzar las 1000 ejecuciones, cuyo procedimiento se presenta en la sección 4.2.2

4 CAPÍTULO IV: ANÁLISIS DE RESULTADOS

En este capítulo se presentan las pruebas realizadas para evaluar el funcionamiento del mecanismo de seguridad implementado en los nodos IoT, la evaluación se centra en el comportamiento del algoritmo ASCON sobre las diferentes plataformas de hardware utilizada.

Para ello, se establecieron condiciones de evaluación comunes entre los dispositivos, comenzando con pruebas preliminares para determinar el número de repeticiones y el tamaño máximo de datos procesados, posteriormente se realizaron las pruebas experimentales de las métricas definidas mediante varias rondas de medición, cuyos resultados permiten comparar el desempeño de las plataformas, analizar el impacto de ASCON sobre los recursos disponibles y determinar su viabilidad en los dispositivos IoT utilizados.

4.1 Plan de Pruebas

De acuerdo a la Tabla 24 se establece un conjunto de pruebas orientadas a analizar el comportamiento del algoritmo en los diferentes nodos IoT, donde se generan condiciones de evaluación comunes entre los dispositivos; iniciando con pruebas preliminares para determinar el número de repeticiones y el tamaño máximo de datos procesados, posteriormente se ejecutan las pruebas experimentales de las métricas definidas mediante varias rondas de medición, cuyos resultados permiten comparar el desempeño de las plataformas, analizar el impacto de ASCON sobre los recursos disponibles y determinar su viabilidad en los dispositivos IoT utilizados.

Tabla 24
Pruebas realizadas dentro del sistema

PLAN DE PRUEBAS		
ID	**Objetivo**	**Resultado**
PR-01	Determinar el número adecuado de repeticiones de ASCON-128.	Establecer una cantidad de repeticiones que permita obtener mediciones estables.
PR-02	Determinar el tamaño máximo de datos que puede procesar cada microcontrolador.	Identificar el límite de datos soportado antes de presentar una restricción de memoria.
PR-03	Verificar el establecimiento de canal seguro mediante ECDH y cifrado ASCON.	Comprobar la generación de la clave y la protección de los datos entre el nodo y el gateway.
PR-04	Evaluar el tiempo de procesamiento de ASCON-128.	Obtener el tiempo promedio de procesamiento en cada plataforma.
PR-05	Evaluar el uso de memoria SRAM durante el proceso seguro.	Determinar el uso de memoria SRAM en cada plataforma.
PR-06	Estimar el consumo energético asociado al procesamiento criptográfico.	Determinar la energía estimada durante el procesamiento en cada plataforma.
PR-07	Evaluar el incremento en el tamaño de los datos protegidos.	Determinar la sobrecarga de datos generada por el mecanismo de seguridad.

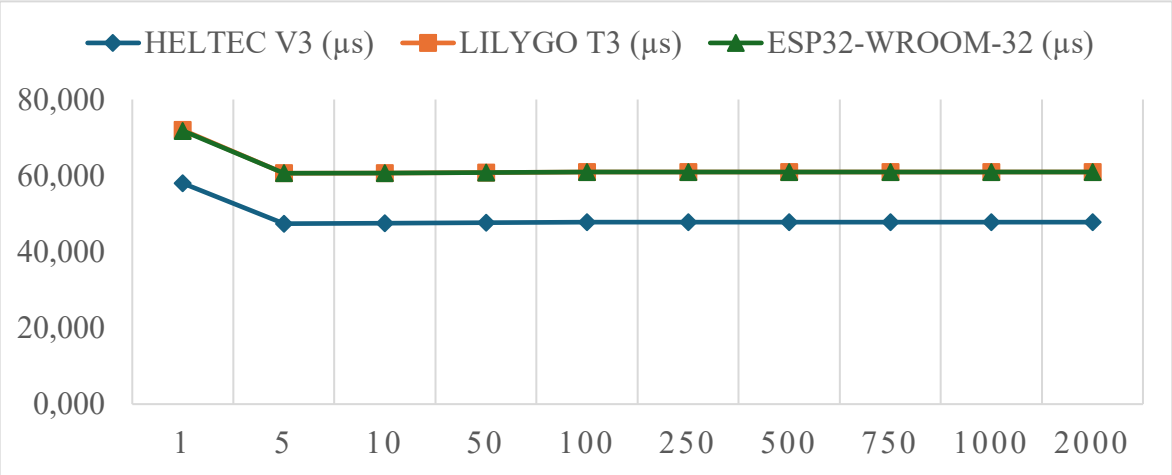
4.1.1 Prueba PR-01: Determinación del Número de Repeticiones

Para establecer el número de repeticiones utilizado en las pruebas experimentales, ASCON se evaluó de forma aislada en las tres plataformas bajo las mismas condiciones de prueba. Para cada cantidad de repeticiones se realiza cinco mediciones independientes y se obtiene el tiempo promedio (µs), las pruebas y los registros obtenidos durante este procedimiento se presentan en el Anexo A, mientras que los resultados se presentan en la Tabla 25.

Tabla 25
Tiempo promedio cifrado ASCON

	Numero de Repeticiones									
Hardware	**1**	**5**	**10**	**50**	**100**	**250**	**500**	**750**	**1000**	**2000**
HELTEC V3	58.00	47.40	47.50	47.68	47.76	47.79	47.79	47.79	47.79	47.79
LILY GO T3	72.00	60.64	60.70	60.88	60.97	60.95	60.97	60.97	60.97	60.97
ESP32-WROOM	71.80	60.64	60.70	60.84	60.94	60.93	60.95	60.96	60.96	60.96

Figura 18
Tiempo de cifrado



Los resultados presentados en la Tabla 25 y representados en la Figura 21 muestran que las mediciones realizadas con pocas ejecuciones presentan una mayor variación, mientras que al incrementar las repeticiones los tiempos promedio tienden a estabilizarse. A partir de los valores obtenidos se seleccionaron 1000 repeticiones debido a que el incremento hasta 2000 no genera cambios relevantes en el tiempo promedio de las tres plataformas. De esta manera, el tiempo de cifrado utilizado en las pruebas posteriores corresponde al promedio calculado a partir de 1000 ejecuciones consecutivas del algoritmo.

4.1.2 Prueba PR-02 Determinación del tamaño máximo de procesamiento

Para determinar el tamaño máximo de datos procesados por cada nodo, se ejecutó ASCON-128 de forma aislada bajo las mismas condiciones de prueba, utilizando una secuencia creciente de tamaños expresados en bits, la evaluación permitió identificar el mayor tamaño procesado correctamente por cada plataforma y verificar el comportamiento del dispositivo al alcanzar un requerimiento superior de memoria tal como se observa en la Tabla 26.

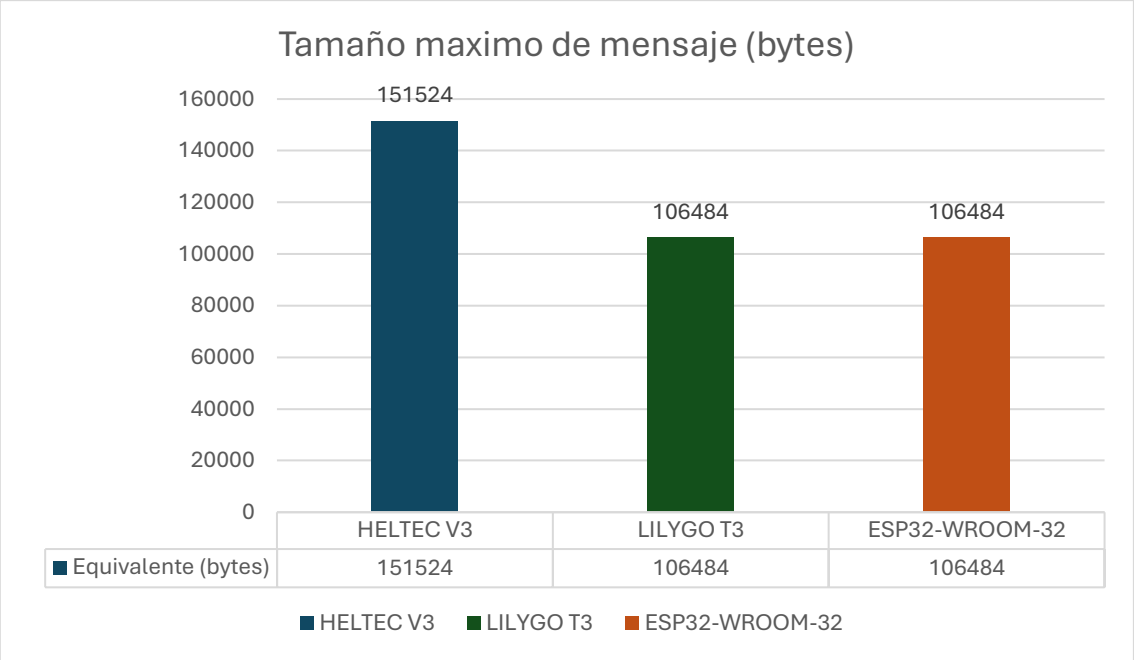
Tabla 26
Tamaño máximo de mensajes procesados

Hardware	Último tamaño válido (bits)	Equivalente (bytes)	Estado
HELTEC V3	1212192	151524	Procesado
LILYGO T3	851872	106484	Procesado
ESP32-WROOM-32	851872	106484	Procesado

Los resultados presentados en la Figura 22 evidencian diferencias en la capacidad de procesamiento de la carga útil entre las plataformas evaluadas; la HELTEC V3 alcanza el mayor tamaño procesado correctamente con 151524 bytes, mientras que la LILYGO T3 y la ESP32-WROOM-32 alcanzan 106484 bytes; esta diferencia se relaciona con la

disponibilidad de memoria durante la ejecución de ASCON, debido a que el procesamiento de mensajes de mayor tamaño requiere reservar *buffers* proporcionales a la carga útil evaluada; en las plataformas que alcanzan 106484 bytes se presenta una condición de memoria insuficiente al procesar el siguiente tamaño de la secuencia, mientras que la HELTEC V3 completa correctamente todos los tamaños establecidos en la prueba.

Figura 19
Tamaño maximo de payload procesado.



4.2 Evaluación de ASCON en los Nodos IoT

En esta etapa se evalúa el funcionamiento del mecanismo de seguridad implementado en los nodos IoT, considerando el establecimiento de clave compartida mediante ECDH y la aplicación del algoritmo de cifrado autenticado ASCON-128. Las pruebas se realizn bajo la arquitectura completa del sistema, donde los nodos sensores generan la información, establecen el canal seguro con el gateway y transmiten mediante MQTT los datos protegidos con ASCON. La evaluación analiza el comportamiento del

mecanismo criptográfico en las diferentes plataformas de hardware mediante el tiempo de procesamiento, consumo energético y tamaño de los datos protegidos.

4.2.1 Prueba PR-03: Establecimiento del Canal Seguro y Cifrado ASCON

Para el desarrollo de la prueba se verifica inicialmente el establecimiento del secreto compartido entre el nodo IoT y el *gateway* mediante ECDH; posteriormente, el valor obtenido se emplea para derivar la clave utilizada por ASCON-128 durante el cifrado y la autenticación de los datos transmitidos mediante MQTT.

El proceso comienza cuando el nodo IoT genera su par de claves ECDH y publica su clave pública mediante el tópico MQTT correspondiente; el *gateway* recibe esta información, genera su propia clave pública y la envía como respuesta al nodo, permitiendo que ambos dispositivos dispongan de la información necesaria para calcular de manera independiente el mismo secreto compartido,

En la implementación desarrollada, el nodo publica su clave pública en el tópico correspondiente y espera la respuesta del gateway. Una vez recibida la clave pública del gateway, el nodo deriva la clave utilizadad por ASCON y habilita el proceso de cifrado; de forma equivalente, el gateway procesa la clave pública recibida desde cada nodo y verifica posteriormente la autenticidad de los mensajes protegidos mediante ASCON.

Figura 20

Proceso de intercambio de clave publica en el nodo.

```
Suscrito a: tesis/ascon/key/gateway/ESP32_WROOM
ECDH inicializado correctamente
Nueva sesion ECDH generada
Clave publica nodo enviada: OK
MQTT recibido en topic: tesis/ascon/key/gateway/ESP32_WROOM
Clave publica gateway recibida
Clave ASCON derivada correctamente por ECDH
```

Después de recibir la respuesta del gateway, el nodo procesa su clave pública y calcula el secreto compartido mediante ECDH; la evidencia presentada en la Figura 23 muestra la recepción de la clave pública del nodo y su procesamiento correcto, lo que permite posteriormente derivar la clave utilizada por ASCON-128.

Figura 21
Establecimiento de ECDH en el gateway



En el gateway, la recepción y procesamiento de la clave pública del nodo permite calcular el mismo secreto compartido; la Figura 24 muestra el mensaje ECDH OK para nodo: ESP32_WROOM, el cual confirma que el proceso ECDH se completa correctamente para dicho dispositivo, mientras que el mensaje correspondiente al envío de la clave pública del gateway evidencia la respuesta enviada al nodo mediante MQTT.

Una vez que ambos dispositivos calculan el mismo secreto compartido, se deriva la clave simétrica empleada por ASCON-128; de esta manera, el nodo utiliza dicha clave para cifrar y autenticar la información del sensor antes de su transmisión, mientras que el gateway emplea la clave equivalente para verificar la autenticidad del mensaje y recuperar los datos originales

Finalmente, durante la prueba el nodo genera la información del sensor, ejecuta el cifrado mediante ASCON y transmite el paquete protegido hacia el gateway mediante MQTT; en el receptor se verifica la etiqueta de autenticación y se realiza el descifrado del mensaje, confirmando la integración correcta entre ECDH y ASCON dentro del sistema implementado.

Las evidencias correspondientes al establecimiento del secreto compartido mediante ECDH y a la integración de ASCON-128 dentro del sistema IoT se presentan en el Anexo C.

4.2.2 Prueba PR-04: Tiempo de Procesamiento de ASCON

Para evaluar el desempeño del algoritmo ASCON en las plataformas IoT implementadas, se analiza el tiempo requerido durante la ejecución de los procesos de cifrado y descifrado, considerando el cifrado realizado en los nodos IoT y el descifrado efectuado en el gateway, donde las mediciones se realizaron bajo las condiciones experimentales establecidas para cada proceso con el propósito de caracterizar el comportamiento temporal de ASCON dentro del sistema implementado.

El tiempo de procesamiento se determina mediante la función `esp_timer_get_time()`, tomando como instante inicial el comienzo de la operación criptográfica y como instante final su culminación; la prueba se ejecuta durante diez rondas por cada plataforma para obtener mediciones individuales y un valor promedio representativo del comportamiento temporal

4.2.2.1 Tiempo de Procesamiento sin Cifrado

Para establecer una referencia del comportamiento temporal de las plataformas IoT sin la incorporación de ASCON, se realiza una prueba bajo un escenario sin cifrado, manteniendo las operaciones de adquisición de datos, construcción del mensaje y transmisión mediante MQTT; esta medición permite determinar el tiempo base de procesamiento de cada nodo antes de incorporar las operaciones criptográficas; los registros individuales obtenidos durante las rondas de medición se presentan en el Anexo D.

4.2.2.2 Tiempo de cifrado en los nodos IoT

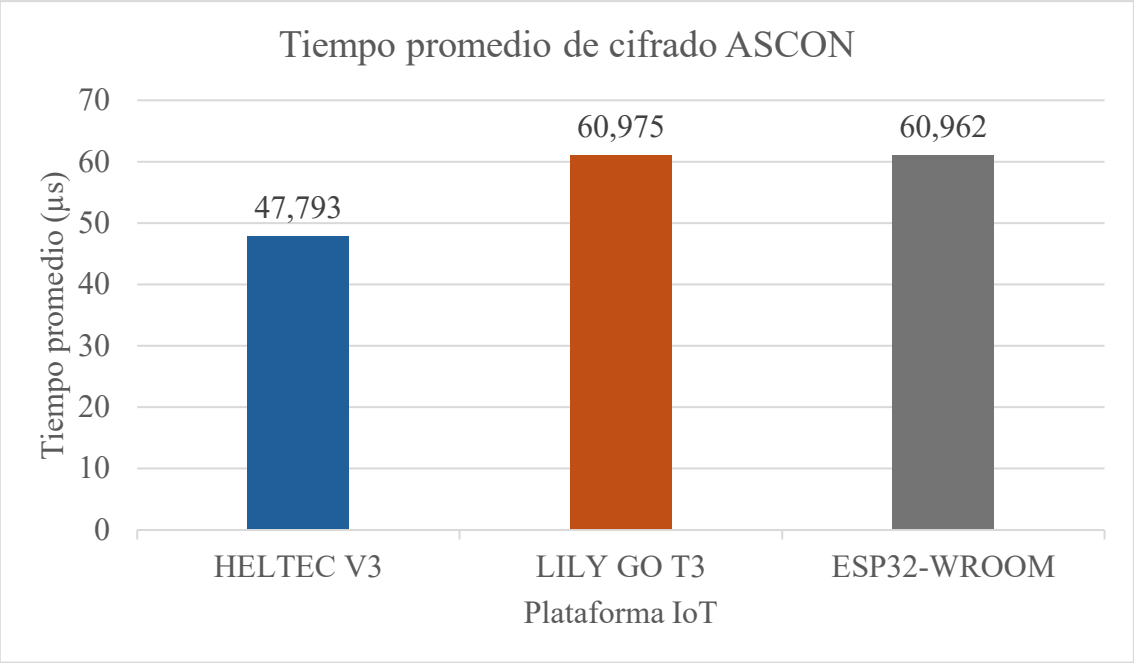
A partir del procedimiento experimental definido previamente, se realizan diez rondas de medición para determinar el comportamiento temporal del cifrado ASCON-128 en las plataformas HELTEC V3, LILYGO T3 y ESP32-WROOM-32; los registros correspondientes a las ejecuciones realizadas en cada plataforma se presentan en el Anexo B, mientras que los valores promedio obtenidos se muestran en la Tabla 27.

Tabla 27
Tiempo promedio de cifrado ASCON

Hardware	Repeticiones	Tiempo promedio
HELTEC V3	1000	47.79
LILY GO T3	1000	60.97
ESP32-WROOM	1000	60.96

Los resultados de la Figura 25 muestran diferencias en el tiempo de ejecución del algoritmo ASCON entre las plataformas evaluadas, la HELTEC V3 presenta el menor tiempo promedio de cifrado con un valor de 47,793 µs, mientras que la LILYGO T3 y la ESP32-WROOM-32 registran valores cercanos a 61 µs, esta variación se relaciona con las características de procesamiento de cada microcontrolador y con la forma en que las operaciones criptográficas son ejecutadas, permitiendo caracterizar el comportamiento temporal de ASCON en los nodos IoT evaluados.

Figura 22
Tiempo promedio evaluado para ASCON



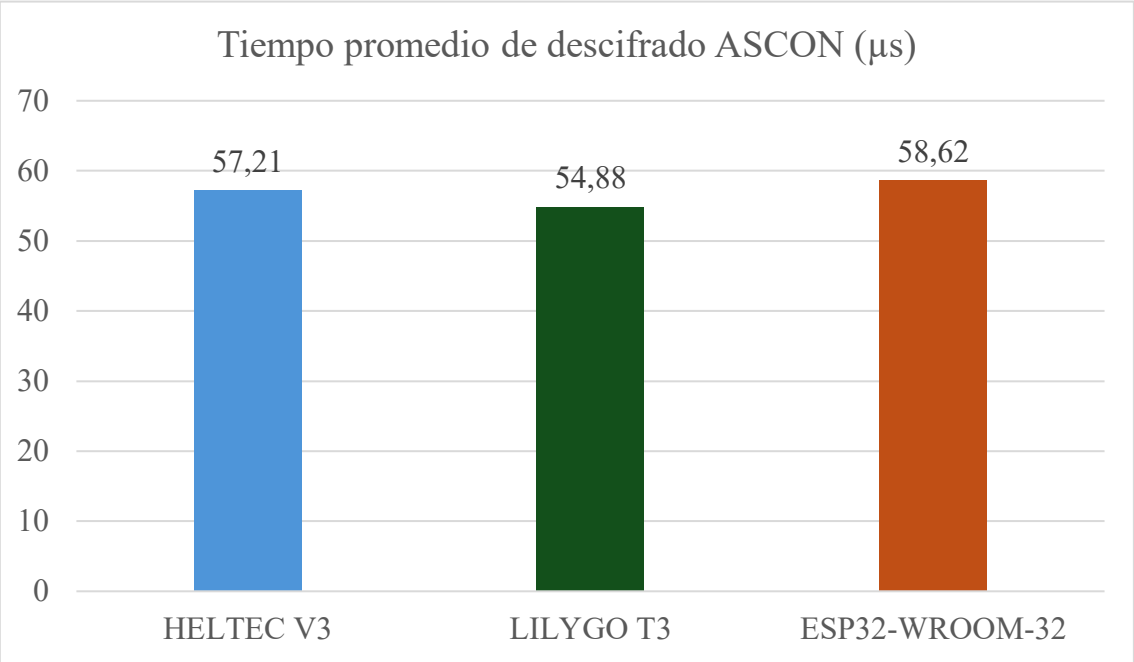
4.2.2.3 Tiempo de descifrado en el gateway

Para evaluar el comportamiento temporal del descifrado de ASCON, se analizan los mensajes cifrados generados por los diferentes nodos y procesados posteriormente en el *gateway*, considerando las mediciones obtenidas durante las diez rondas experimentales y empleando como variable de análisis el tiempo promedio requerido para completar una operación de descifrado; los valores promedio obtenidos se presentan en la Tabla 28, mientras que la evidencia correspondiente a la recuperación correcta del mensaje y a la validación del proceso de descifrado ejecutado en el *gateway* se presenta en el Anexo E

Tabla 28
Tiempo promedio de descifrado ASCON

Hardware	Repeticiones	Tiempo promedio (µs)
HELTEC V3	1000	57,21
LILYGO T3	1000	54,88
ESP32-WROOM-32	1000	58,62

Figura 23
Tiempo promedio de descifrado ASCON



Los resultados obtenidos de la Figura 26 muestran que el tiempo promedio de descifrado presenta valores cercanos entre los mensajes provenientes de las tres plataformas, la LILYGO T3 registra el menor valor con 54,88 µs, seguida por el ESP32-WROOM-32 con 58,62 µs, mientras que la HELTEC V3 alcanza 57,21 µs, estas diferencias son reducidas, por lo que el comportamiento temporal del descifrado se mantiene relativamente estable entre los mensajes procesados.

Cabe considerar que el descifrado se ejecuta directamente en el gateway, por lo que el tiempo registrado no corresponde al procesamiento criptográfico realizado por cada placa IoT, sino a la ejecución de la misma operación sobre la Raspberry Pi para los mensajes provenientes de las diferentes plataformas, las pequeñas variaciones observadas pueden relacionarse con el tamaño de los mensajes procesados y con las condiciones de ejecución del gateway durante cada medición, además de las variaciones propias de la ejecución del software y del sistema durante las pruebas.

4.2.3 Comparación del Tiempo de Cifrado y Descifrado

La comparación de los resultados permite identificar el comportamiento temporal asociado al cifrado y descifrado de ASCON-128; el cifrado realizado en los nodos IoT presenta valores promedio comprendidos entre 47,793 µs y 60,975 µs, mientras que el descifrado ejecutado en el *gateway* registra valores promedio entre 54,88 µs y 61,21 µs; estos resultados muestran que ambas operaciones presentan tiempos relativamente cercanos bajo las condiciones establecidas para la evaluación.

La diferencia entre los tiempos de cifrado y descifrado se relaciona principalmente con el entorno de ejecución de cada proceso; el cifrado se ejecuta directamente en los microcontroladores IoT mediante C/C++, mientras que el descifrado se realiza desde Python en el *gateway* mediante la interfaz `ctypes`, la cual permite acceder a la biblioteca compilada de ASCON-C.

De esta manera, ambos procesos utilizan la misma implementación criptográfica basada en ASCON-C, aunque cada operación se ejecuta en un entorno diferente; las variaciones temporales se relacionan con las características de procesamiento de cada plataforma y con la forma en que el programa accede a la biblioteca durante la ejecución.

4.2.3.1 Tiempo Total con Cifrado

A partir de los tiempos obtenidos en ambos escenarios, se plantea se estima el tiempo de procesamiento con incorporación de ASCON mediante la suma del tiempo base sin cifrado y el tiempo promedio correspondiente al cifrado, expresada mediante la ecuación 1:

Ecuación 1

$$T_{estimado} = T_{\text{sin}\,cifrado} + T_{cifrado}$$

La aplicación de esta relación permite estimar el tiempo de procesamiento asociado a la incorporación de ASCON, los valores obtenidos mediante esta estimación se presentan en la Tabla 29, donde se incluyen el tiempo de procesamiento sin cifrado, el tiempo promedio de cifrado ASCON y el tiempo estimado resultante para cada plataforma IoT.

Tabla 29
Tiempo de procesamiento total con ASCON expresada en microsegundos

Hardware	Tiempo sin cifrado (µs)	Tiempo de cifrado ASCON (µs)	Tiempo total (µs)
HELTEC V3	26638	47,79	26 685,79
LILYGO T3	25582	60,98	25 642,98
ESP32-WROOM-32	25701	60,96	25 761,96

La Tabla 29 muestra que la incorporación de ASCON incrementa el tiempo estimado de procesamiento en 47,79 µs para la HELTEC V3, 60,98 µs para la LILYGO T3 y 60,96 µs para la ESP32-WROOM-32, manteniéndose el incremento reducido frente al tiempo base registrado en cada plataforma, esta diferencia permite identificar el aporte temporal específico del cifrado dentro del proceso completo de adquisición, construcción y transmisión del mensaje.

Debido a que las mediciones obtenidas durante las pruebas se expresan inicialmente en microsegundos (µs), se realiza la conversión a segundos (s) para presentar los resultados bajo una unidad común de tiempo, considerando que un segundo equivale a un millón de microsegundos, por lo que cada valor medido en microsegundos se divide entre 1000000, mediante la siguiente expresión, los cuales los valores obtenidos después de realizar esta conversión se presentan en la Tabla 30.

Ecuación 2

$$T_S = \frac{T_{us}}{1000000}$$

Tabla 30
Tiempo de procesamiento total con ASCON expresada en segundos

Hardware	Tiempo sin cifrado (s)	Tiempo de cifrado ASCON (s)	Tiempo total (s)
HELTEC V3	0,026638	0,00004779	0,02668579
LILYGO T3	0,025582	0,00006098	0,02564298
ESP32-WROOM-32	0,025701	0,00006096	0,02576196

Para determinar la incidencia que representa el cifrado respecto al tiempo total de procesamiento, se calcula el incremento temporal asociado a ASCON en términos porcentuales mediante la siguiente expresión:

Ecuación 3

$$\boldsymbol{P_{cifrdao}} \boldsymbol{\frac{T_{cifrado}}{T_{total}} * 100\%}$$

En términos relativos, el tiempo asociado al cifrado representa aproximadamente el 0,18 % del tiempo base en la HELTEC V3, el 0,24 % en la LILYGO T3 y el 0,24 % en la ESP32-WROOM-32, estos resultados evidencian una incidencia temporal reducida frente al conjunto de operaciones ejecutadas durante el proceso evaluado.

4.2.3.2 Tiempo total con descifrado

Una vez determinado el tiempo de procesamiento sin cifrado, se incorpora el tiempo promedio requerido para el descifrado de los mensajes en el gateway, con el propósito de determinar el tiempo estimado del proceso de recepción cuando interviene en esta etapa.

El tiempo estimado con descifrado se determina mediante la suma del tiempo de procesamiento sin cifrado y el tiempo promedio de descifrado, de acuerdo con la siguiente expresión:

Ecuación 4

$$T_{estimado} = T_{\text{sin } cifrado} + T_{descifrado}$$

Los valores obtenidos permiten identificar el incremento temporal asociado al descifrado de ASCON-128, tomando como referencia el tiempo de procesamiento registrado sin la intervención del algoritmo criptográfico; a este valor base se incorpora el tiempo promedio de descifrado ejecutado en el *gateway*, permitiendo estimar el tiempo total del proceso cuando se incluye esta operación criptográfica, , cuyos resultados se presentan en la Tabla 31.

Tabla 31
Tiempo de procesamiento con descifrado ASCON-128

Hardware	Tiempo sin cifrado (µs)	Tiempo descifrado ASCON (µs)	Tiempo estimado (µs)
HELTEC V3	26638	57,21	26695,21
LILYGO T3	25582	54,88	25636,88
ESP32-WROOM-32	25701	58,62	25759,62

Los resultados muestran que la incorporación del descifrado de ASCON-128 incrementa el tiempo estimado del proceso en 57,21 µs para la HELTEC V3, 54,88 µs para la LILYGO T3 y 58,62 µs para el ESP32-WROOM-32, los incrementos corresponden exclusivamente al tiempo requerido para ejecutar el descifrado en el gateway y permiten determinar su incidencia respecto al tiempo de procesamiento sin cifrado.

Los valores obtenidos se convierten posteriormente a segundos para mantener una unidad común de análisis, como se presenta en la Tabla 32.

Tabla 32
Tiempo de procesamiento con descifrado ASCON-128 en segundos

Hardware	Tiempo sin cifrado (s)	Tiempo de descifrado ASCON (s)	Tiempo estimado (s)
HELTEC V3	0,026638	0,00005721	0,02669521
LILYGO T3	0,025582	0,00005488	0,02563688
ESP32-WROOM-32	0,025701	0,00005862	0,02575962

En términos relativos, el tiempo asociado al descifrado representa aproximadamente el 0,21 % del tiempo total en la HELTEC V3, el 0,21 % en la LILYGO T3 y el 0,23 % en el ESP32-WROOM-32; estos resultados muestran que la operación de descifrado presenta una incidencia temporal reducida respecto al tiempo de procesamiento tomado como referencia sin la intervención de ASCON.

4.2.3.3 Tiempo Total con Cifrado y Descifrado

Una vez definidos los tiempos correspondientes al cifrado y al descifrado de ASCON, se integran ambos valores con el tiempo de procesamiento sin cifrado para representar el comportamiento temporal del proceso completo, de esta manera se consideran las operaciones criptográficas que intervienen desde el procesamiento del mensaje en los nodos IoT hasta su recuperación en el gateway.

El tiempo total de procesamiento se determina mediante la suma del tiempo sin cifrado, el tiempo promedio de cifrado y el tiempo promedio de descifrado, de acuerdo con la siguiente expresión:

Ecuación 5

$$T_{estimado} = T_{\text{sin}\,cifrado} + T_{\text{c}\,ifrado} + T_{descifrado}$$

Los valores utilizados corresponden a las mediciones obtenidas para cada plataforma y se expresan en segundos para mantener una unidad común durante la

comparación, de esta manera se puede observar tanto el tiempo base del proceso como el incremento que introducen las operaciones criptográficas, cuyos resultados se presentan en la Tabla 33.

Tabla 33
Tiempo total de procesamiento con cifrado y descifrado ASCON-128

Hardware	Tiempo sin cifrado (s)	Tiempo de cifrado ASCON (s)	Tiempo de descifrado ASCON (s)	Tiempo total (s)
HELTEC V3	0,026638	0,00004779	0,00005721	0,02674300
LILYGO T3	0,025582	0,00006098	0,00005488	0,02569786
ESP32-WROOM-32	0,025701	0,00006096	0,00005862	0,02582058

Para determinar la incidencia que tienen conjuntamente el cifrado y el descifrado dentro del tiempo total de procesamiento, se consideran ambos tiempos como el aporte correspondiente a ASCON-128 y se toma el tiempo total como referencia, esta relación permite expresar en términos porcentuales la participación de las operaciones criptográficas dentro del proceso evaluado.

Ecuación 6

$$\boldsymbol{P_{cifrdao}} \frac{\boldsymbol{T_{cifrado}} + \boldsymbol{T_{descifrado}}}{\boldsymbol{T_{total}}} * \mathbf{100}\%$$

La aplicación de esta relación muestra que las operaciones de cifrado y descifrado representan aproximadamente el 0,39 % del tiempo total en la HELTEC V3, el 0,45 % en la LILYGO T3 y el 0,47 % en el ESP32-WROOM-32, donde la HELTEC V3 presenta la menor incidencia conjunta y el ESP32-WROOM-32 la mayor, aunque la diferencia entre las plataformas es reducida y los tres valores permanecen por debajo del uno por ciento, lo que evidencia que la incorporación de ASCON representa una fracción pequeña del tiempo total considerado en la evaluación.

4.2.4 Prueba PR-06: Consumo Energético Durante ASCON-128

Para evaluar el impacto energético asociado a la ejecución del algoritmo ASCON en las plataformas IoT, se analizó el consumo eléctrico durante el proceso criptográfico. La medición fue realizada utilizando el módulo INA219, encargado de registrar los valores de corriente y potencia consumida durante la ejecución del cifrado autenticado.

Para determinar el consumo energético durante la ejecución de ASCON-128 se utilizaron los valores de corriente y potencia registrados mediante el módulo INA219. La potencia eléctrica requerida por cada plataforma durante el proceso criptográfico se obtiene mediante la relación entre el voltaje de alimentación y la corriente consumida:

Ecuación 7

$$\boldsymbol{P = V * I}$$

Donde:

- $\boldsymbol{P}$ representa la potencia consumida durante la ejecución del algoritmo (W).
- $\boldsymbol{V}$ corresponde al voltaje medido por el módulo INA219 (V).
- $\boldsymbol{I}$ representa la corriente registrada durante el proceso criptográfico (A).

La energía consumida durante una operación de cifrado se determina mediante la integración de la potencia durante el tiempo de ejecución:

Ecuación 8

$$\boldsymbol{E_{ascon} = P_{ascon} * \ t_{ascon}}$$

Donde:

- $\boldsymbol{E_{ascon}}$ representa la energía requerida para completar una operación de cifrado (J).
- $\boldsymbol{P_{ascon}}$corresponde a la potencia consumida durante el proceso (W).

- t_{ascon} corresponde al tiempo de ejecución del algoritmo (s).

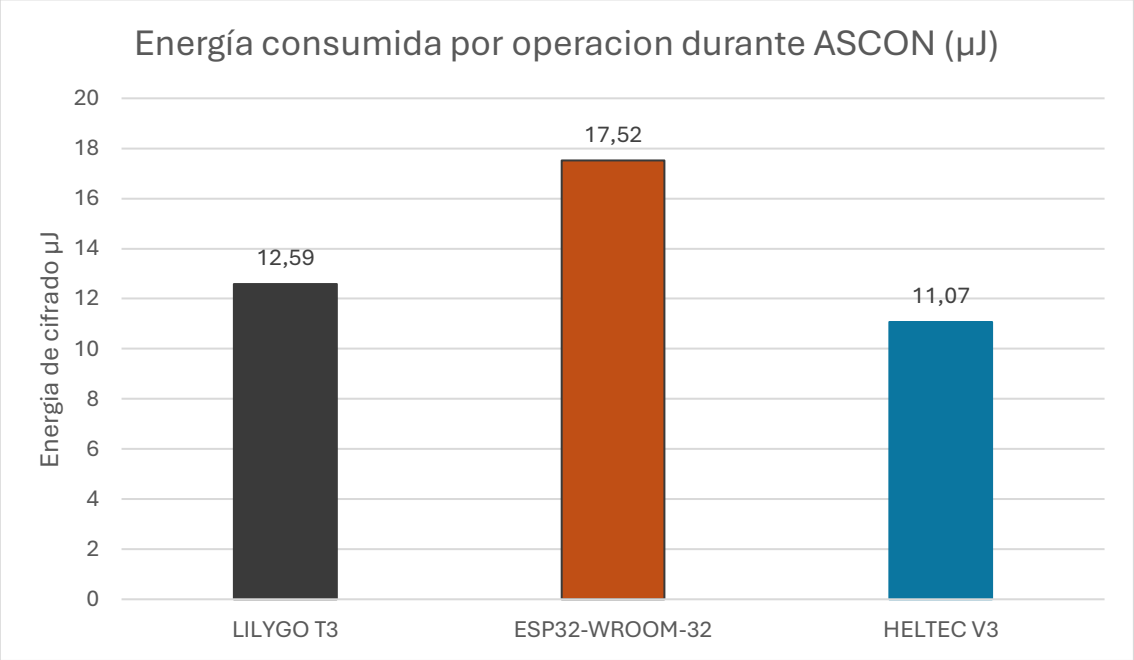
Debido a que el tiempo de ejecución registrado por el sistema se encuentra expresado en microsegundos, se realiza la conversión de unidades correspondiente para expresar la energía final en microjulios (µJ).

$$1J = 106\mu J$$

Tabla 34
Consumo energético durante cifrado ASCON

Hardware	Tiempo ASCON (µs)	Corriente (mA)	Potencia (mW)	Energía por operación (µJ)
LILYGO T3	61.13	51.7	200	12.59
ESP32-WROOM-32	61.91	53.8	278	17.52
HELTEC V3	47.91	55.0	232	11.07

Figura 24
Energía de cifrado ASCON



Los resultados obtenidos en la Tabla 34 y la Figura 27 permiten analizar el comportamiento energético del algoritmo ASCON en las plataformas IoT evaluadas. La

HELTEC V3 presentó el menor consumo energético por operación con un valor de 11.07 µJ, seguida de la LILYGO T3 con 12.59 µJ y la ESP32-WROOM-32 con 17.52 µJ.

Aunque la HELTEC V3 alcanzó el menor tiempo de procesamiento con 47.91 µs, la diferencia energética respecto a la LILYGO T3 fue reducida debido a los valores de potencia registrados durante la ejecución.

Por último, la ESP32-WROOM-32 presentó el mayor consumo energético debido al incremento de potencia requerido durante el proceso criptográfico. Estos resultados evidencian que la eficiencia energética depende de la relación entre el tiempo de ejecución y la potencia consumida por cada plataforma durante la operación de ASCON.

4.2.5 Prueba PR-07: Consumo Energético Durante Descifrado

Para complementar la evaluación del comportamiento energético de ASCON-128, se determina la energía asociada al proceso de descifrado considerando la potencia registrada durante las pruebas experimentales y el tiempo promedio requerido para completar esta operación en el gateway, esta relación permite obtener una estimación de la energía empleada durante el procesamiento de los mensajes protegidos.

Tabla 35
Energía durante el descifrado de ASCON-128

Hardware	Potencia (mW)	Tiempo de descifrado (µs)	Energía (µJ)
HELTEC V3	232	57,21	13,28
LILYGO T3	200	54,88	10,98
ESP32-WROOM-32	278	58,62	16,29

Los resultados presentados en la Tabla 35 muestran que la energía asociada al proceso de descifrado alcanza 13,28 µJ en la HELTEC V3, 10,98 µJ en la LILYGO T3 y 16,29 µJ en el ESP32-WROOM-32, la LILYGO T3 presenta el menor valor debido a la combinación entre su potencia registrada y el tiempo requerido para completar la operación, mientras que el ESP32-WROOM-32 presenta el mayor valor entre las

plataformas evaluadas, estos resultados permiten complementar el análisis temporal del descifrado mediante la caracterización de la energía asociada a la operación criptográfica.

4.3 Análisis General de Resultados

Con el objetivo de validar la estabilidad del tiempo de procesamiento del algoritmo ASCON en las plataformas IoT evaluadas, se realizaron múltiples ejecuciones del proceso criptográfico bajo las mismas condiciones experimentales.

Los valores registrados fueron procesados mediante herramientas estadísticas descriptivas con el propósito de determinar la tendencia central y la variabilidad de los tiempos obtenidos. Para este análisis se consideraron la media aritmética, varianza muestral, desviación estándar muestral e intervalo de confianza del 95 %.

1. Media aritmética

La media aritmética permite determinar el valor promedio del tiempo de ejecución obtenido durante las mediciones realizadas.

Ecuación 9. **Media aritmética**

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

2. Varianza

La varianza permite cuantificar la dispersión existente entre los valores obtenidos respecto al valor promedio.

Ecuación 10, Varianza

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

3. Desviación estándar

La desviación estándar permite determinar la variación promedio de los tiempos de ejecución respecto al valor medio.

Ecuación 11. Desviación estándar

$$\sigma = \sqrt{\sigma^2}$$

4. Intervalo de confianza del 95 %

Determina el rango donde se encuentra el valor promedio esperado de las mediciones se calculó el intervalo de confianza mediante la distribución normal:

Ecuación 12. Intervalo de confianza

$$IC = u \pm Z\left(\frac{\sigma}{\sqrt{N}}\right)$$

4.3.1 Análisis de Tiempo de Cifrado

Aplicando dichas formulas, los datos obtenidos durante las pruebas fueron procesados y los resultados estadísticos se presentan en la siguiente Tabla 36:

Tabla 36
Análisis estadístico del tiempo de ejecución

Hardware	Media (µs)	Varianza muestral	Desviación estándar (µs)	IC 95 % (µs)
HELTEC V3	47.94	0.003832	0.061905	±0.038368
LILYGO T3	61.07	0.000684	0.026162	±0.016215
ESP32-WROOM	61.91	0.000093	0.009661	±0.005988

Los resultados estadísticos obtenidos permiten determinar la estabilidad temporal del algoritmo ASCON-128 en las plataformas evaluadas. La HELTEC V3 presentó un tiempo promedio de ejecución de 47.94 µs, siendo la plataforma con menor latencia durante el proceso criptográfico. Por otra parte, la LILYGO T3 y la ESP32-WROOM-32 presentaron valores promedio de 61.07 µs y 61.91 µs, respectivamente, mostrando tiempos superiores, pero con un comportamiento estable.

En relación con la dispersión de los datos, la ESP32-WROOM-32 presentó la menor desviación estándar con un valor de 0.009661 µs, seguida de la LILYGO T3 con 0.026162 µs. La HELTEC V3 obtuvo una desviación estándar de 0.061905 µs, manteniendo igualmente una variación reducida respecto al tiempo promedio registrado.

Los intervalos de confianza calculados presentan rangos pequeños en comparación con los tiempos medios obtenidos, evidenciando que las mediciones realizadas poseen baja variabilidad y que los valores promedio representan adecuadamente el comportamiento del algoritmo bajo las condiciones experimentales establecidas.

4.3.2 Análisis de Tiempo de Descifrado

Aplicando las fórmulas estadísticas establecidas, los datos obtenidos durante las pruebas de descifrado de ASCON-128 se procesan para determinar el comportamiento temporal de la operación ejecutada en el gateway, los resultados estadísticos correspondientes a cada plataforma se presentan en la Tabla 37

Tabla 37
Análisis estadístico del tiempo de descifrado

Hardware	Media (µs)	Varianza muestral	Desviación estándar (µs)	IC 95 % (µs)
HELTEC V3	57,21	10,268895	3,204512	±1,986140
LILYGO T3	54,89	170,826504	13,070061	±8,100759

ESP32-WROOM-32	58,62	98,717377	9,935662	±6,158074

En relación con la dispersión de los datos, la HELTEC V3 presenta la menor desviación estándar con 3,204512 µs, mientras que la ESP32-WROOM-32 alcanza 9,935662 µs y la LILYGO T3 registra 13,070061 µs, la mayor variabilidad observada en estas dos últimas plataformas se relaciona con la presencia de mediciones inferiores al comportamiento predominante durante algunas rondas, mientras que la HELTEC V3 mantiene sus valores más próximos a su media

Los intervalos de confianza del 95 % presentan valores de ±1,986140 µs para la HELTEC V3, ±8,100759 µs para la LILYGO T3 y ±6,158074 µs para el ESP32-WROOM-32, estos resultados muestran un intervalo más reducido en la HELTEC V3, mientras que la LILYGO T3 presenta el rango más amplio debido a la mayor dispersión de sus mediciones, lo que permite representar de manera más clara la variabilidad temporal presente durante el proceso de descifrado

4.3.3 Análisis de Consumo Energético Durante Cifrado.

Los valores obtenidos durante las pruebas experimentales fueron procesados para determinar el comportamiento energético de las plataformas IoT durante la ejecución del algoritmo ASCON. Los resultados estadísticos obtenidos se presentan en la Tabla 38.

Tabla 38
Análisis estadístico del consumo energético durante el cifrado

Hardware	Media (µJ)	Varianza muestral	Desviación estándar (µJ)	IC 95 % (µJ)
HELTEC V3	11.09	2.618446	1.618161	±1.002928
LILYGO T3	14.38	18.884868	4.345672	±2.693426
ESP32-WROOM-32	17.73	0.017671	0.132933	±0.082391

Los resultados estadísticos obtenidos permiten analizar la estabilidad del consumo energético durante la ejecución de ASCON-128 en las plataformas evaluadas. La HELTEC V3 presentó el menor consumo promedio con un valor de 11.09 μJ por operación, seguida de la LILYGO T3 con 14.38 μJ y la ESP32-WROOM-32 con 17.73 μJ, evidenciando diferencias asociadas principalmente a las características eléctricas y capacidad de procesamiento de cada dispositivo.

En relación con la dispersión de los datos, la ESP32-WROOM-32 presentó el menor valor de desviación estándar con 0.132933 μJ, indicando una mayor estabilidad entre las mediciones realizadas. La HELTEC V3 obtuvo una desviación estándar de 1.618161 μJ, mientras que la LILYGO T3 presentó el valor más elevado con 4.345672 μJ debido a las variaciones observadas durante las ejecuciones experimentales.

Los intervalos de confianza calculados permiten establecer el rango esperado alrededor del valor promedio de energía consumida. La ESP32-WROOM-32 presentó el intervalo más reducido con ±0.082391 μJ, mientras que la LILYGO T3 obtuvo una mayor amplitud con ±2.693426 μJ debido a la dispersión presente en sus mediciones.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones:

Se concluye que la incorporación del algoritmo permite proteger los datos generados por las plataformas HELTEC V3, LILYGO T3 y ESP32-WROOM-32 durante su transmisión hacia el gateway, el proceso mantiene la estructura de comunicación mediante MQTT y permite recuperar correctamente la información original después del descifrado, por lo que la protección criptográfica se integra al sistema sin impedir el funcionamiento del proceso de adquisición y transmisión de datos

En relación con el establecimiento de claves, se concluye que el mecanismo ECDH permite generar una clave compartida entre cada nodo IoT y el gateway antes de ejecutar ASCON-128, esta implementación permite asociar el material criptográfico con cada plataforma y establecer las condiciones necesarias para que el cifrado realizado en los microcontroladores y el descifrado ejecutado en el gateway sean compatibles, manteniendo la separación entre el proceso de intercambio de claves y la protección de los mensajes.

A partir de las pruebas de cifrado y descifrado, se concluye que ASCON-128 mantiene un funcionamiento correcto en los diferentes componentes del sistema, los mensajes generados por los nodos se cifran antes de su transmisión mediante MQTT y posteriormente se recuperan en el gateway mediante una implementación basada en ASCON-C, donde la comparación entre el texto original y el texto recuperado permite verificar la correcta ejecución del proceso criptográfico.

Respecto al comportamiento temporal, se concluye que la incorporación de ASCON-128 representa un incremento reducido frente al procesamiento sin cifrado, el tiempo promedio de cifrado alcanza 47,94 µs en la HELTEC V3, 61,07 µs en la LILYGO

T3 y 61,91 µs en el ESP32-WROOM-32, mientras que el descifrado registra 57,21 µs, 54,89 µs y 58,62 µs, respectivamente, estos resultados permiten identificar el tiempo requerido por cada operación criptográfica sin generar una afectación considerable sobre el tiempo global del proceso evaluado.

El análisis estadístico permite concluir que el comportamiento temporal del cifrado presenta una dispersión reducida entre las mediciones realizadas, con desviaciones estándar de 0,061905 µs en la HELTEC V3, 0,026162 µs en la LILYGO T3 y 0,009661 µs en el ESP32-WROOM-32, mientras que el descifrado presenta una mayor variabilidad, con desviaciones estándar de 3,204512 µs, 13,070061 µs y 9,935662 µs, respectivamente, esta diferencia se refleja principalmente en la LILYGO T3, donde se registra la mayor dispersión, mientras que la incidencia conjunta del cifrado y descifrado representa aproximadamente 0,41 % en la HELTEC V3, 0,45 % en la LILYGO T3 y 0,47 % en el ESP32-WROOM-32, evidenciando que las operaciones criptográficas mantienen una participación temporal reducida dentro del procesamiento global del sistema.

Recomendaciones.

Debido a que las mediciones realizadas permiten identificar diferencias en el comportamiento temporal del cifrado y descifrado entre las plataformas evaluadas, se recomienda ampliar el número de rondas y periodos de observación, esta ampliación permite determinar con mayor precisión la variabilidad de los tiempos registrados y establecer si las diferencias observadas corresponden a comportamientos recurrentes del sistema.

Debido a que la implementación permite registrar parámetros eléctricos mediante el INA219, se recomienda profundizar en la medición del consumo de corriente, potencia y energía asociado específicamente al cifrado y descifrado de ASCON-128, esta

evaluación permite complementar el análisis temporal y determinar el impacto de la protección criptográfica sobre el consumo energético de las plataformas IoT.

Dado que las pruebas actuales se realizan con mensajes de tamaño definido para la aplicación desarrollada, se recomienda evaluar ASCON-128 utilizando diferentes longitudes de payload, esta variación permite determinar cómo cambia el tiempo de cifrado, descifrado y consumo energético conforme aumenta la cantidad de información procesada.

A partir de los resultados obtenidos con ASCON-128, se recomienda realizar futuras evaluaciones comparativas con otros algoritmos orientados a dispositivos con recursos limitados, considerando parámetros como tiempo de ejecución, consumo energético, memoria utilizada y tamaño del mensaje protegido, esta comparación permitiría determinar las condiciones bajo las cuales ASCON-128 presenta ventajas frente a otras alternativas para aplicaciones IoT.

REFERENCIAS

Barrio, M. (2018). *Internet de las cosas*. Editorial Reus. https://doi.org/10.30462/9788429020380

Buchanan, W. J., & Maglaras, L. (2023). Review of the NIST light-weight cryptography finalists. In *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)* (pp. 469–474). IEEE. https://doi.org/10.1109/DCOSS-IoT58021.2023.00079

Tonato Chuquimarca, C., & Sinche Maita, S. (2022). Análisis comparativo entre arquitecturas de sistemas IoT. *Revista de Investigación en Tecnologías de la Información, 10*(21 Especial), 55–70. https://doi.org/10.36825/RITI.10.21.006

Dobraunig, C., Eichlseder, M., Mendel, F., & Schläffer, M. (2021). Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology, 34*(3), Article 33. https://doi.org/10.1007/s00145-021-09398-9

Dobraunig, C., Eichlseder, M., Mendel, F., & Schläffer, M. (s. f.). *Ascon lightweight cryptography*. Graz University of Technology. Recuperado el 17 de julio de 2025, de https://ascon.isec.tugraz.at/

Espressif Systems. (2026). *ESP32 series datasheet* (Version 5.3) [Hoja de datos]. https://documentation.espressif.com/esp32_datasheet_en.pdf

Espressif Systems. (2026). *ESP32-WROOM-32 datasheet* (Version 3.7) [Hoja de datos]. https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf

Eterovic, J., & Cipriano, M. (2017). Introducción al estudio de algoritmos criptográficos livianos. En *XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017)* (pp. 1061–1064). Red de Universidades con Carreras en Informática. http://sedici.unlp.edu.ar/handle/10915/62719

haukex. (2026). *arduino-ascon128* [Código fuente]. GitHub. https://github.com/haukex/arduino-ascon128

Heltec Automation. (2022). *WiFi LoRa 32 V3.2: LoRa node development kit* (Rev. 1.1) [Hoja de datos]. https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3.2.pdf

ISO/IEC/IEEE. (2018). *Systems and software engineering—Life cycle processes—Requirements engineering* (ISO/IEC/IEEE 29148:2018). https://www.iso.org/standard/72089.html

International Telecommunication Union. (2012). *Overview of the Internet of things* (Recommendation ITU-T Y.4000/Y.2060). https://www.itu.int/rec/T-REC-Y.4000/en

Makhdoom, I., Abolhasan, M., Abbas, H., & Ni, W. (2019). Blockchain's adoption in IoT: The challenges, and a way forward. *Journal of Network and Computer Applications, 125*, 251–279. https://doi.org/10.1016/j.jnca.2018.10.019

Medina Vargas, Y. T., & Miranda Méndez, H. A. (2015). Comparación de algoritmos basados en la criptografía simétrica DES, AES y 3DES. *Mundo FESC, 5*(9), 14–21. https://doi.org/10.61799/2216-0388.55

Mekki, K., Bajic, E., Chaxel, F., & Meyer, F. (2019). A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express, 5*(1), 1–7. https://doi.org/10.1016/j.icte.2017.12.005

Oliveira Morilla, S. (2024). *Análisis integral de seguridad en dispositivos IoT* [Trabajo final de grado, Universitat Oberta de Catalunya]. UOC Open Access. http://hdl.handle.net/10609/150709

Ortega Chulde, C. A. (2023). *Implementación de mecanismo de seguridad para redes de sensores inalámbricos basado en criptografía de curva elíptica* [Tesis de pregrado, Universidad Técnica del Norte]. Repositorio Digital Universidad Técnica del Norte. https://repositorio.utn.edu.ec/handle/123456789/14449

Ortega Patiño, A. I. (2023). *Aplicación de criptografía de curva elíptica para redes de sensores en sistemas de alerta temprana* [Tesis de pregrado, Universidad Técnica del Norte]. Repositorio Digital Universidad Técnica del Norte. https://repositorio.utn.edu.ec/handle/123456789/14603

Poonam, G. (2023, 28 de agosto). *Managing resource-constrained IoT devices: Unlocking potential amidst limitations*. Medium. https://medium.com/plexusbit-software/managing-resource-constrained-iot-devices-unlocking-potential-amidst-limitations-be932269e846

Prieto Álvarez, C. G. (2015). *Adaptación de las metodologías tradicionales cascada y espiral para la inclusión de evaluación inicial de usabilidad en el desarrollo de productos de software en México* [Tesis de maestría, Universidad Tecnológica de la Mixteca]. Repositorio Institucional de la Universidad Tecnológica de la Mixteca. https://repositorio.utm.mx/handle/123456789/72

Python Cryptographic Authority. (2026). *Cryptography 49.0.0 documentation*. https://cryptography.io/en/49.0.0/

Roa Buendía, J. F. (2013). *Seguridad informática* (2.ª ed.). McGraw-Hill España.

Ruminot Ahumada, N. A. (2024). *Propuesta de esquemas criptográficos y de transmisión para arquitecturas LPWAN en eHealth* [Tesis de maestría, Universidad de Chile]. Repositorio Académico de la Universidad de Chile. https://repositorio.uchile.cl/handle/2250/203177

Texas Instruments. (2015). *INA219 Zerø-drift, bidirectional current/power monitor with I2C interface* (Rev. G; SBOS448G) [Hoja de datos]. https://www.ti.com/lit/ds/symlink/ina219.pdf

Trusted Firmware. (s. f.). *Mbed TLS, TF-PSA-Crypto*. Recuperado el 7 de septiembre de 2026, de https://www.trustedfirmware.org/projects/mbed-tls/

Turan, M. S., McKay, K. A., Kang, J., Kelsey, J., & Chang, D. (2025). *Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions* (NIST Special Publication 800-232). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-232

Vallejo Rosero, C. I. (2023). *Esquema de autenticación para redes de sensores (WSN) basada en tecnología de cadena de bloques y Tangle* [Tesis de pregrado, Universidad Técnica del Norte]. Repositorio Digital Universidad Técnica del Norte. https://repositorio.utn.edu.ec/handle/123456789/14218

Villa Crespo, E., & Morales Alonso, I. (2017). *Ciberseguridad IoT y su aplicación en ciudades inteligentes* (1.ª ed.). RA-MA Editorial.

Wadho, S. A., Meghji, A. F., Yichiet, A., Kumar, R., & Shaikh, F. B. (2023). Encryption techniques and algorithms to combat cybersecurity attacks: A review. *VAWKUM Transactions on Computer Sciences, 11*(1), 295–305. https://doi.org/10.21015/vtcs.v11i1.1521

Yacchirema Vargas, D. C. (2019). *Arquitectura de interoperabilidad de dispositivos físicos para el Internet de las Cosas (IoT)* [Tesis doctoral, Universitat Politècnica de València]. RiuNet. https://hdl.handle.net/10251/129858

Yugha, R., & Chithra, S. (2020). A survey on technologies and security protocols: Reference for future generation IoT. *Journal of Network and Computer Applications, 169*, 102763. https://doi.org/10.1016/j.jnca.2020.102763

ANEXOS

ANEXO A. Evidencias de la determinación del número de repeticiones

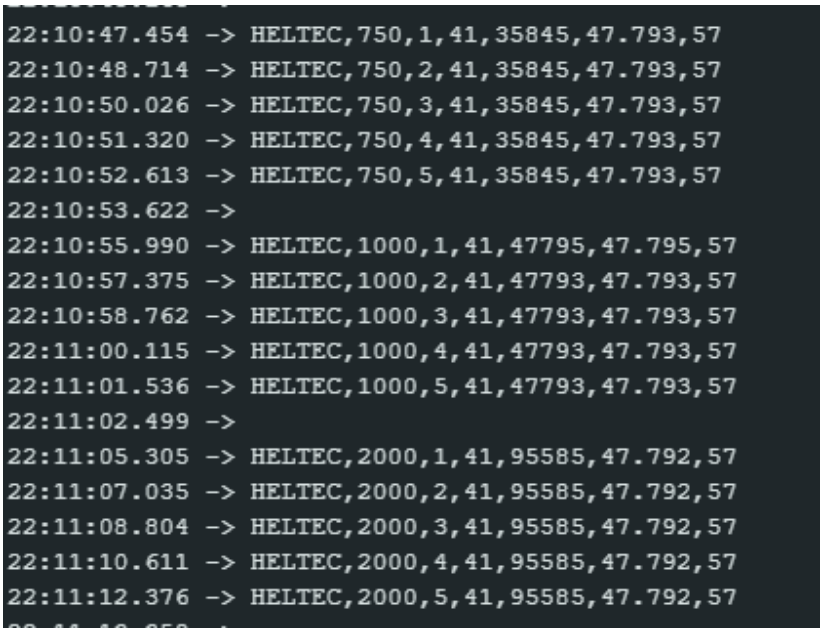
En este anexo se presentan los registros de la prueba PR-01 utilizada para determinar el número de repeticiones de ASCON-128; las capturas muestran la evolución del tiempo promedio desde ejecuciones individuales hasta 2000 repeticiones y respaldan la selección de 1000 ejecuciones como condición de referencia para las pruebas posteriores.

- **HELTEC V3**

1. Evidencia A.1. Registros iniciales de repeticiones en HELTEC V3

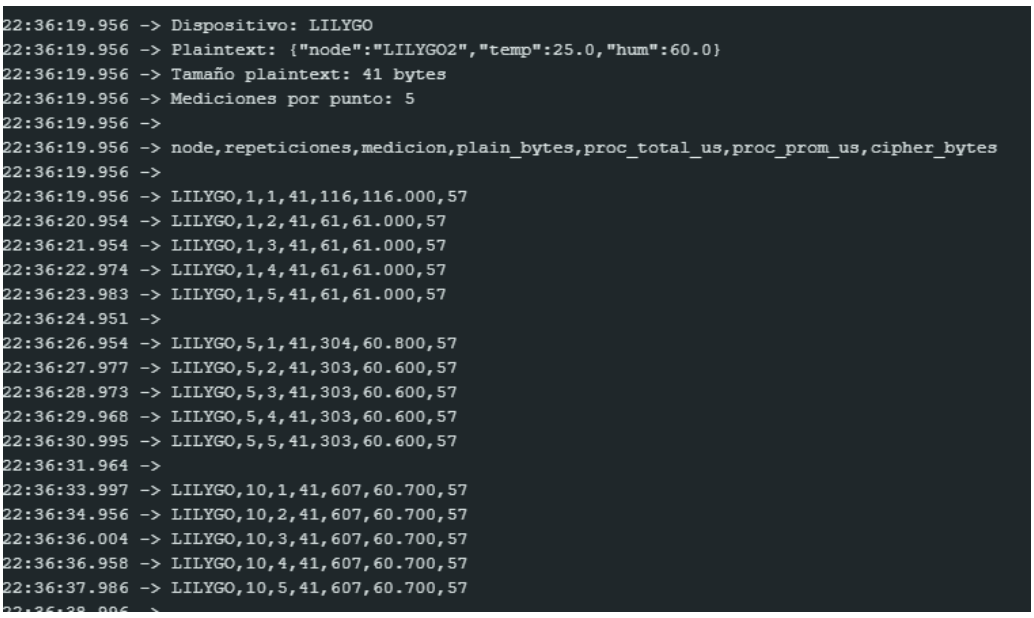
```
22:09:56.400 -> Dispositivo: HELTEC
22:09:56.400 -> Plaintext: {"node":"LILYGO2","temp":25.0,"hum":60.0}
22:09:56.400 -> Tamaño plaintext: 41 bytes
22:09:56.400 -> Mediciones por punto: 5
22:09:56.400 ->
22:09:56.400 -> node,repeticiones,medicion,plain_bytes,proc_total_us,proc_prom_us,cipher_bytes
22:09:56.400 ->
22:09:56.400 -> HELTEC,1,1,41,102,102.000,57
22:09:57.395 -> HELTEC,1,2,41,47,47.000,57
22:09:58.410 -> HELTEC,1,3,41,47,47.000,57
22:09:59.374 -> HELTEC,1,4,41,47,47.000,57
22:10:00.419 -> HELTEC,1,5,41,47,47.000,57
22:10:01.418 ->
22:10:03.381 -> HELTEC,5,1,41,237,47.400,57
22:10:04.380 -> HELTEC,5,2,41,237,47.400,57
22:10:05.424 -> HELTEC,5,3,41,237,47.400,57
22:10:06.384 -> HELTEC,5,4,41,237,47.400,57
22:10:07.429 -> HELTEC,5,5,41,237,47.400,57
22:10:08.407 ->
22:10:10.414 -> HELTEC,10,1,41,475,47.500,57
22:10:11.394 -> HELTEC,10,2,41,475,47.500,57
22:10:12.440 -> HELTEC,10,3,41,475,47.500,57
22:10:13.402 -> HELTEC,10,4,41,475,47.500,57
22:10:14.437 -> HELTEC,10,5,41,475,47.500,57
```

2. **Evidencia A.2. Registros de 750, 1000 y 2000 repeticiones en HELTEC V3**



- **LILYGO T3**

3. **Evidencia A.3. Registros iniciales de repeticiones en LILYGO T3**



4. **Evidencia A.4. Registros de 750, 1000 y 2000 repeticiones en LILYGO T3**

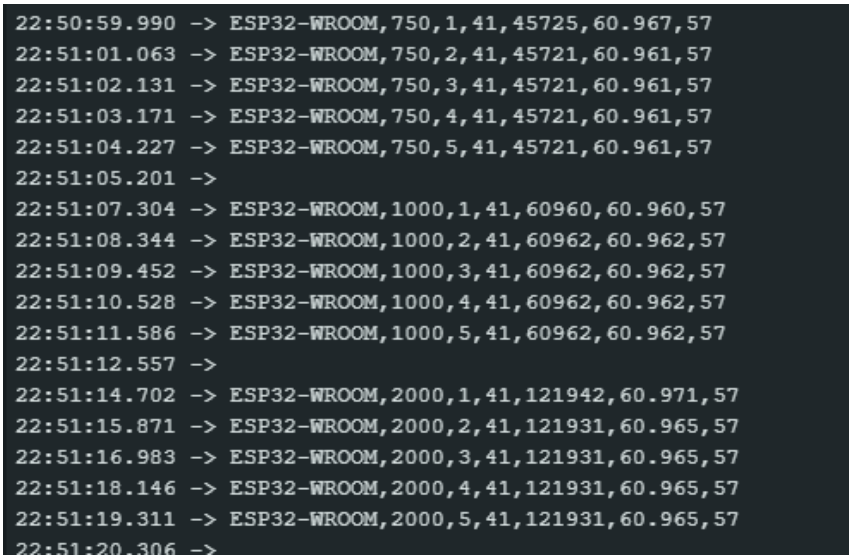
```
22:37:09.367 -> LILYGO,750,1,41,45725,60.967,57
22:37:10.407 -> LILYGO,750,2,41,45734,60.979,57
22:37:11.462 -> LILYGO,750,3,41,45734,60.979,57
22:37:12.492 -> LILYGO,750,4,41,45734,60.979,57
22:37:13.561 -> LILYGO,750,5,41,45734,60.979,57
22:37:14.543 ->
22:37:16.616 -> LILYGO,1000,1,41,60975,60.975,57
22:37:17.718 -> LILYGO,1000,2,41,60975,60.975,57
22:37:18.792 -> LILYGO,1000,3,41,60975,60.975,57
22:37:19.854 -> LILYGO,1000,4,41,60975,60.975,57
22:37:20.944 -> LILYGO,1000,5,41,60975,60.975,57
22:37:21.941 ->
22:37:24.069 -> LILYGO,2000,1,41,121952,60.976,57
22:37:25.222 -> LILYGO,2000,2,41,121960,60.980,57
22:37:26.324 -> LILYGO,2000,3,41,121960,60.980,57
22:37:27.499 -> LILYGO,2000,4,41,121960,60.980,57
22:37:28.644 -> LILYGO,2000,5,41,121960,60.980,57
```

- **ESP32-WROOM-32**

5. **Evidencia A.5. Registros iniciales de repeticiones en ESP32-WROOM-32**

```
22:50:10.611 -> Dispositivo: ESP32-WROOM
22:50:10.611 -> Plaintext: {"node":"LILYGO2","temp":25.0,"hum":60.0}
22:50:10.611 -> Tamaño plaintext: 41 bytes
22:50:10.611 -> Mediciones por punto: 5
22:50:10.611 ->
22:50:10.611 -> node,repeticiones,medicion,plain_bytes,proc_total_us,proc_prom_us,cipher_bytes
22:50:10.645 ->
22:50:10.645 -> ESP32-WROOM,1,1,41,115,115.000,57
22:50:11.612 -> ESP32-WROOM,1,2,41,61,61.000,57
22:50:12.654 -> ESP32-WROOM,1,3,41,61,61.000,57
22:50:13.650 -> ESP32-WROOM,1,4,41,61,61.000,57
22:50:14.632 -> ESP32-WROOM,1,5,41,61,61.000,57
22:50:15.630 ->
22:50:17.651 -> ESP32-WROOM,5,1,41,304,60.800,57
22:50:18.611 -> ESP32-WROOM,5,2,41,303,60.600,57
22:50:19.612 -> ESP32-WROOM,5,3,41,303,60.600,57
22:50:20.641 -> ESP32-WROOM,5,4,41,303,60.600,57
22:50:21.639 -> ESP32-WROOM,5,5,41,303,60.600,57
22:50:22.648 ->
22:50:24.657 -> ESP32-WROOM,10,1,41,607,60.700,57
22:50:25.657 -> ESP32-WROOM,10,2,41,607,60.700,57
22:50:26.614 -> ESP32-WROOM,10,3,41,607,60.700,57
22:50:27.615 -> ESP32-WROOM,10,4,41,607,60.700,57
22:50:28.643 -> ESP32-WROOM,10,5,41,607,60.700,57
```

6. **Evidencia A.6. Registros de 750, 1000 y 2000 repeticiones en ESP32-WROOM**



ANEXO B. Pruebas de cifrado en los nodos IoT

En este anexo se presentan las evidencias correspondientes al proceso de cifrado ejecutado directamente en los nodos IoT, donde se muestran las condiciones empleadas durante las pruebas y los tiempos registrados para las diferentes repeticiones, estas capturas sirven como respaldo de las mediciones utilizadas posteriormente para determinar el comportamiento temporal del cifrado de ASCON-128.

- Heltec

7. **Evidencia B.1.** ***Imagen de cifrado de microcontrolador Heltec***

```
00:12:15.926 -> ========== ENVIO ==========
00:12:15.958 -> Nodo: HELTEC
00:12:15.958 -> Modo: ASCON + ECDH
00:12:15.958 -> ASCON status: ASCON_OK
00:12:15.958 -> Repeticiones ASCON: 1000
00:12:15.958 -> Bytes cifrados reales: 40
00:12:15.958 -> Bits cifrados reales: 320
00:12:15.958 -> Ciphertext + tag bytes: 56
00:12:15.958 -> Paquete seguro bytes IV + ciphertext + tag: 72
```

- ESP32-WROOM

8. **Evidencia B.2. Imagen de cifrado de microcontrolador ESP32-WROOM**

```
00:13:37.326 -> ========== ENVIO ==========
00:13:37.326 -> Nodo: ESP32_WROOM
00:13:37.326 -> Modelo configurado: ESP32-WROOM-32
00:13:37.361 -> Chip detectado: ESP32-D0WD-V3
00:13:37.361 -> Frecuencia CPU: 240 MHz
00:13:37.361 -> Modo: ASCON + ECDH
00:13:37.361 -> ASCON status: ASCON_OK
00:13:37.361 -> Repeticiones ASCON: 1000
00:13:37.361 -> Bytes cifrados reales: 45
00:13:37.361 -> Bits cifrados reales: 360
00:13:37.361 -> Ciphertext + tag bytes: 61
00:13:37.361 -> Paquete seguro bytes IV + ciphertext + tag: 77
```

- LilyGo

9. **Evidencia B.3. Imagen de cifrado de microcontrolador LiLYGo**

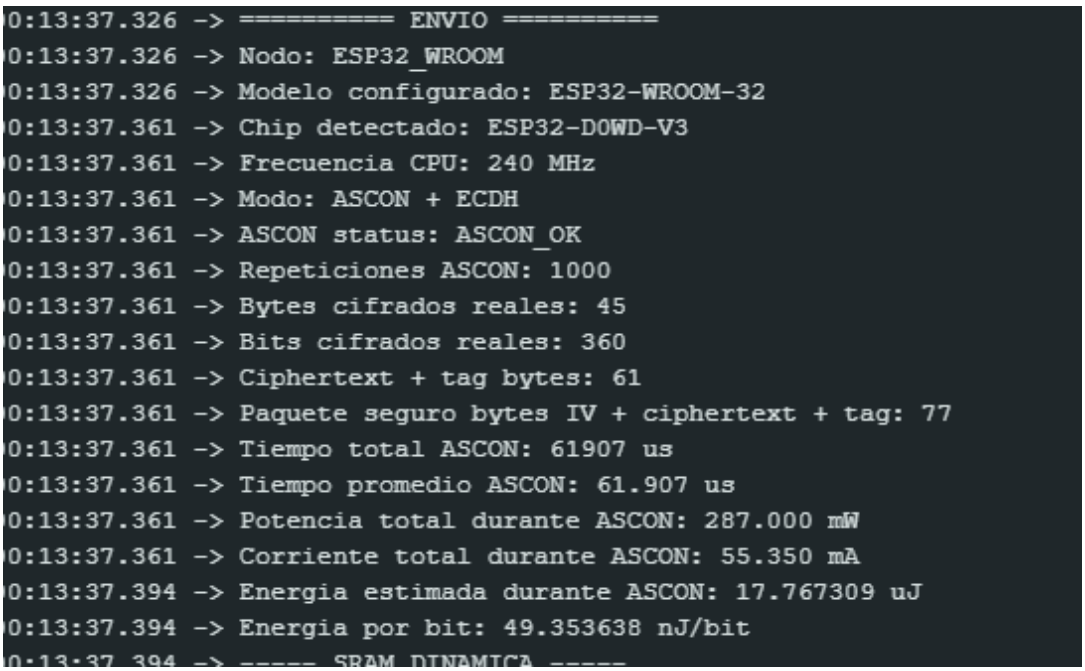
```
20:30:00.567 -> ========== ENVIO ==========
20:30:00.567 -> Nodo: LILYGO2
20:30:00.567 -> Modo: ASCON + ECDH
20:30:00.567 -> ASCON status: ASCON_OK
20:30:00.567 -> Repeticiones ASCON: 1000
20:30:00.567 -> Bytes cifrados reales: 41
20:30:00.567 -> Bits cifrados reales: 328
20:30:00.567 -> Ciphertext + tag bytes: 57
20:30:00.567 -> Paquete seguro bytes: 73
```

5 ANEXO C. Integración de ASCON-128 en el sistema IoT

En este anexo se presentan las evidencias correspondientes a la integración de ASCON-128 dentro de la arquitectura de comunicación IoT, donde se observa el intercambio de información entre los nodos y el gateway mediante el esquema implementado, estas pruebas permiten respaldar el funcionamiento conjunto de las etapas de generación, protección, transmisión, recepción y recuperación de los datos.

- **ESP32-WROOM**

10. Evidencia C.1. Ejecución de ASCON en ESP32-WROOM



- Heltec

11. Evidencia C.2. Ejecución de ASCON en Heltec

```
00:12:15.926 -> ========== ENVIO ==========
00:12:15.958 -> Nodo: HELTEC
00:12:15.958 -> Modo: ASCON + ECDH
00:12:15.958 -> ASCON status: ASCON_OK
00:12:15.958 -> Repeticiones ASCON: 1000
00:12:15.958 -> Bytes cifrados reales: 40
00:12:15.958 -> Bits cifrados reales: 320
00:12:15.958 -> Ciphertext + tag bytes: 56
00:12:15.958 -> Paquete seguro bytes IV + ciphertext + tag: 72
00:12:15.958 -> Tiempo total ASCON: 47896 us
00:12:15.958 -> Tiempo promedio ASCON: 47.896 us
00:12:15.958 -> Potencia total durante ASCON: 411.000 mW
00:12:15.958 -> Corriente total durante ASCON: 98.100 mA
00:12:15.990 -> Energia estimada durante ASCON: 19.685257 uJ
00:12:15.990 -> Energia por bit: 61.516426 nJ/bit
```

- LiLyGO

12. Evidencia C.3. Ejecución de ASCON en ESP32-WROOM

```
20:30:00.567 -> ========== ENVIO ==========
20:30:00.567 -> Nodo: LILYGO2
20:30:00.567 -> Modo: ASCON + ECDH
20:30:00.567 -> ASCON status: ASCON_OK
20:30:00.567 -> Repeticiones ASCON: 1000
20:30:00.567 -> Bytes cifrados reales: 41
20:30:00.567 -> Bits cifrados reales: 328
20:30:00.567 -> Ciphertext + tag bytes: 57
20:30:00.567 -> Paquete seguro bytes: 73
20:30:00.567 -> Tiempo total ASCON: 61107 us
20:30:00.567 -> Tiempo promedio ASCON: 61.107 us
20:30:00.567 -> SRAM dinamica total: 281960 bytes
20:30:00.603 -> SRAM libre inicial: 177344 bytes
20:30:00.603 -> SRAM libre minima observada: 175128 bytes
20:30:00.603 -> SRAM adicional maxima del proceso: 2216 bytes
20:30:00.603 -> SRAM dinamica ocupada en el pico: 106832 bytes
20:30:00.603 -> Potencia durante ASCON: 1696.000 mW
20:30:00.603 -> Corriente durante ASCON: 414.200 mA
20:30:00.603 -> Energia estimada durante ASCON: 103.637466 uJ
20:30:00.603 -> Energia por bit: 315.967896 nJ/bit
```

ANEXO D. Registros de las mediciones sin cifrado.

En este anexo se presentan los registros obtenidos durante las pruebas de procesamiento sin cifrado realizadas en las plataformas IoT evaluadas, estas mediciones se utilizan como referencia para establecer el tiempo base de procesamiento de cada plataforma y posteriormente determinar el incremento temporal asociado a la incorporación de ASCON-128, los registros muestran las rondas ejecutadas, el tiempo acumulado y el tiempo promedio obtenido durante cada prueba

- ESP32-WROOM

13. **Evidencia D.1. Tiempo sin cifrado nodo ESP32-WROOM**

```
01:51:35.640 ->
01:51:35.640 -> ====================================
01:51:35.640 ->  RESULTADO FINAL SIN CIFRADO
01:51:35.640 -> ====================================
01:51:35.640 -> Rondas realizadas: 10
01:51:35.640 -> Tiempo acumulado: 257016 us
01:51:35.640 -> Tiempo promedio sin cifrado: 25701.600 us
01:51:35.640 -> ====================================
```

- Heltec

14. **Evidencia D.2.** Tiempo sin cifrado nodo ESP32-WROOM

```
02:12:36.398 -> ====================================
02:12:36.398 ->  RESULTADO FINAL - HELTEC V3
02:12:36.398 ->        PROCESO SIN CIFRADO
02:12:36.398 -> ====================================
02:12:36.398 -> Rondas realizadas: 10
02:12:36.398 -> Tiempo acumulado: 266385 us
02:12:36.425 -> Tiempo promedio sin cifrado: 26638.500 us
02:12:36.425 -> Tiempo promedio sin cifrado: 26.638500 ms
02:12:36.425 -> ====================================
02:12:36.425 ->
```

- LiLyGo

15. **Evidencia D.3.** Tiempo sin cifrado nodo ESP32-WROOM

```
02:50:11.615 -> ====================================
02:50:11.615 ->  RESULTADO FINAL LILYGO T3
02:50:11.655 ->        SIN CIFRADO
02:50:11.655 -> ====================================
02:50:11.655 -> Rondas realizadas: 10
02:50:11.655 -> Tiempo acumulado: 255821 us
02:50:11.655 -> Tiempo promedio sin cifrado: 25582.100 us
02:50:11.655 -> Tiempo promedio sin cifrado: 25.582100 ms
02:50:11.655 -> ====================================
02:50:11.655 ->
```

ANEXO E. Verificación funcional del descifrado en el gateway

En este anexo se presenta una evidencia de verificación funcional del proceso de descifrado ejecutado en el gateway; el registro muestra la recuperación correcta del texto original y la validación de la operación criptográfica, por lo que se utiliza como respaldo funcional y no como sustituto de los valores estadísticos presentados en el Capítulo IV

- **Evidencia E.1. Verificación de recuperación del mensaje mediante ASCON-128 en el gateway**



ANEXO F. Código fuente de la implementación de ASCON-128

En este anexo se presenta el código fuente desarrollado para la implementación y evaluación de ASCON-128 en el sistema IoT, incluyendo las funciones utilizadas para el cifrado, descifrado, generación y manejo de los parámetros criptográficos, así como los procedimientos empleados para realizar las mediciones de tiempo durante las pruebas, el código presentado corresponde a la implementación utilizada para obtener los resultados finales descritos en el presente trabajo:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <Ascon128.h>
#include <esp_system.h>
#include <esp_timer.h>
#include <esp_heap_caps.h>
#include <string.h>
#include <Wire.h>
#include <math.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_INA219.h>

#include "mbedtls/ecdh.h"
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/sha256.h"

// ============================================================
// WIFI
// ============================================================

const char* ssid = "Nombre de la Red";
const char* password = "Contraseña de la Red";

// ============================================================
// MQTT
// ============================================================

const char* mqtt_server = "IP Gateway";
const uint16_t MQTT_PORT = 1883;

// ============================================================
// CONFIGURACION DEL NODO
// ============================================================

const char* DEV_ID = "HELTEC";

#define DHTPIN 4
#define DHTTYPE DHT11
```

```
const unsigned long PUBLISH_INTERVAL_MS = 10000;
const unsigned long KEY_RETRY_INTERVAL_MS = 5000;
const unsigned long KEY_HEARTBEAT_INTERVAL_MS = 15000;
const unsigned long POWER_SAMPLE_INTERVAL_MS = 1000;

// Se mantiene el mismo numero de repeticiones en todas las
pruebas.
const int REPETICIONES_ASCON = 1000;

// ============================================================
// OLED
// ============================================================

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_SDA 17
#define OLED_SCL 18
#define OLED_RST 21

Adafruit_SSD1306 display(
  SCREEN_WIDTH,
  SCREEN_HEIGHT,
  &Wire,
  OLED_RST
);

// ============================================================
// INA219
// ============================================================
TwoWire I2C_INA = TwoWire(1);
Adafruit_INA219 ina219(0x40);
bool ina_ok = false;

float ina_bus_v = 0.0;
float ina_shunt_mv = 0.0;
float ina_load_v = 0.0;
float ina_current_ma = 0.0;
float ina_current_abs_ma = 0.0;
float ina_power_mw = 0.0;

double acc_mAh = 0.0;
double acc_mWh = 0.0;
double avg_current_mA = 0.0;
double est_runtime_h = 0.0;

const float BATTERY_CAPACITY_MAH = 500.0;

unsigned long ina_last_ms = 0;
unsigned long ina_start_ms = 0;
unsigned long lastPowerSample = 0;

// ============================================================
// MQTT TOPICS
// ============================================================

WiFiClient espClient;
PubSubClient client(espClient);

const char* TOPIC_SENSOR = "tesis/ascon/sensor";
const char* TOPIC_METRICS = "tesis/ascon/metrics";
const char* TOPIC_STATUS = "tesis/ascon/status";
```

```
const char* TOPIC_KEY_NODE = "tesis/ascon/key/node";

char TOPIC_KEY_GATEWAY[80];

// ============================================================
// SENSOR
// ============================================================

DHT dht(DHTPIN, DHTTYPE);

// ============================================================
// ASCON
// ============================================================

uint8_t key[16];
bool ecdh_ready = false;

uint8_t iv[16];
uint8_t plaintext[128];
uint8_t ciphertext[144];

unsigned long lastPublish = 0;
unsigned long lastKeySend = 0;

volatile uint8_t ascon_control = 0;

// ============================================================
// ECDH
// ============================================================

mbedtls_ecdh_context ecdh;
mbedtls_entropy_context entropy;
mbedtls_ctr_drbg_context ctr_drbg;

uint8_t node_public_key[100];
size_t node_public_key_len = 0;

bool node_public_sent = false;
bool ecdh_initialized = false;

char last_gateway_public_hex[201] = "";

// ============================================================
// MEDICION UNIFICADA DE SRAM DINAMICA INTERNA
// ============================================================

// La ventana cubre la generacion del nonce, el cifrado ASCON,
// la construccion del paquete seguro, la conversion hexadecimal,
// la creacion del JSON y la publicacion MQTT.
//
// sram_used representa el incremento maximo de memoria dinamica
// durante la ventana:
//
// sram_used = SRAM libre inicial - SRAM libre minima del proceso
//
//  sram_occupied_peak  conserva  la  ocupacion  dinamica  total
observada:
//
// sram_occupied_peak = SRAM dinamica total - SRAM libre minima
//
```

```
      // La primera medida permite comparar el costo adicional del
proceso
      // entre placas, mientras la segunda muestra la ocupacion general.

      size_t sram_total_dinamica = 0;
      size_t sram_libre_inicio = 0;
      size_t sram_libre_minima = 0;
      size_t sram_ocupada_maxima = 0;
      size_t sram_incremento_proceso = 0;

      bool medicion_sram_activa = false;

      size_t obtenerSRAMLibreInterna() {
        return heap_caps_get_free_size(
          MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT
        );
      }

      size_t obtenerSRAMTotalDinamica() {
        return heap_caps_get_total_size(
          MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT
        );
      }

      void iniciarMedicionSRAMProceso() {
        sram_total_dinamica = obtenerSRAMTotalDinamica();
        sram_libre_inicio = obtenerSRAMLibreInterna();
        sram_libre_minima = sram_libre_inicio;
        sram_ocupada_maxima = 0;
        sram_incremento_proceso = 0;
        medicion_sram_activa = true;
      }
      void registrarSRAMProceso() {
        if (!medicion_sram_activa) {
          return;
        }
        size_t sram_libre_actual = obtenerSRAMLibreInterna();
        if (sram_libre_actual < sram_libre_minima) {
          sram_libre_minima = sram_libre_actual;
        }
      }
      void finalizarMedicionSRAMProceso() {
        registrarSRAMProceso();
        if (sram_total_dinamica > sram_libre_minima) {
          sram_ocupada_maxima =
            sram_total_dinamica - sram_libre_minima;
        } else {
          sram_ocupada_maxima = 0;
        }
        if (sram_libre_inicio > sram_libre_minima) {
          sram_incremento_proceso =
            sram_libre_inicio - sram_libre_minima;
        } else {
          sram_incremento_proceso = 0;
        }
        medicion_sram_activa = false;
      }

      // ============================================================
      // OLED
      // ============================================================
```

```
void mostrarEstadoASCON(const char* estado) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  display.setTextSize(1);
  display.setCursor(0, 15);
  display.println("ESTADO ASCON");

  display.setTextSize(2);
  display.setCursor(0, 35);
  display.println(estado);

  display.display();
}

void mostrarMetricasASCON(
  const char* estado,
  float tiempo_us,
  float energia_uJ
) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("PRUEBA ASCON");

  display.setCursor(0, 16);
  display.print("Estado: ");
  display.println(estado);

  display.setCursor(0, 32);
  display.print("T us: ");
  display.println(tiempo_us, 2);

  display.setCursor(0, 48);
  display.print("E uJ: ");
  display.println(energia_uJ, 4);

  display.display();
}

// ============================================================
// UTILIDADES HEX Y JSON
// ============================================================

void to_hex(const uint8_t* data, size_t len, char* out) {
  const char* hex = "0123456789ABCDEF";

  for (size_t i = 0; i < len; i++) {
    out[i * 2] = hex[(data[i] >> 4) & 0x0F];
    out[i * 2 + 1] = hex[data[i] & 0x0F];
  }

  out[len * 2] = '\0';
}

int hex_to_bytes(
  const char* hex,
  uint8_t* out,
```

```
  size_t max_len
) {
  size_t hex_len = strlen(hex);

  if (hex_len % 2 != 0) {
    return -1;
  }

  size_t out_len = hex_len / 2;

  if (out_len > max_len) {
    return -1;
  }

  for (size_t i = 0; i < out_len; i++) {
    sscanf(hex + 2 * i, "%2hhx", &out[i]);
  }

  return (int)out_len;
}

bool extract_json_value(
  const char* json,
  const char* key_name,
  char* out,
  size_t out_size
) {
  char pattern[40];

  snprintf(
    pattern,
    sizeof(pattern),
    "\"%s\":\"",
    key_name
  );

  char* start = strstr((char*)json, pattern);

  if (!start) {
    return false;
  }

  start += strlen(pattern);

  char* end = strchr(start, '"');

  if (!end) {
    return false;
  }

  size_t len = (size_t)(end - start);

  if (len >= out_size) {
    return false;
  }

  strncpy(out, start, len);
  out[len] = '\0';

  return true;
}
```

```
// ============================================================
// INA219
// ============================================================

void iniciarINA219() {
  I2C_INA.begin(41, 42);
  if (!ina219.begin(&I2C_INA)) {
    Serial.println("ERROR: No se detecta INA219");
    ina_ok = false;
    return;
  }
  ina219.setCalibration_32V_2A();
  ina_ok = true;
  unsigned long t = millis();
  ina_start_ms = t;
  ina_last_ms = t;
  lastPowerSample = t;
  Serial.println("INA219 iniciado correctamente");
}
void leerINA219(unsigned long now) {
  if (!ina_ok) {
    return;
  }
  ina_shunt_mv = ina219.getShuntVoltage_mV();
  ina_bus_v = ina219.getBusVoltage_V();
  ina_current_ma = ina219.getCurrent_mA();
  ina_power_mw = ina219.getPower_mW();
  ina_load_v = ina_bus_v + (ina_shunt_mv / 1000.0);
  ina_current_abs_ma = fabs(ina_current_ma);
  float power_abs_mw = fabs(ina_power_mw);
  if (ina_last_ms > 0 && now >= ina_last_ms) {
    double dt_h =
      (double)(now - ina_last_ms) / 3600000.0;
    acc_mAh += ina_current_abs_ma * dt_h;
    acc_mWh += power_abs_mw * dt_h;
  }
  ina_last_ms = now;
  double elapsed_h =
    (double)(now - ina_start_ms) / 3600000.0;
  if (elapsed_h > 0.0) {
    avg_current_mA = acc_mAh / elapsed_h;
    if (avg_current_mA > 0.5) {
      est_runtime_h =
        BATTERY_CAPACITY_MAH / avg_current_mA;
    } else {
      est_runtime_h = 0.0;
    }
  }
}

// ============================================================
// ECDH
// ============================================================

void liberarECDH() {
  if (!ecdh_initialized) {
    return;
  }

  mbedtls_ecdh_free(&ecdh);
```

```
  mbedtls_ctr_drbg_free(&ctr_drbg);
  mbedtls_entropy_free(&entropy);

  ecdh_initialized = false;
}

bool iniciarECDH() {
  int ret;
  const char* pers = "ecdh_ascon_node";

  mbedtls_ecdh_init(&ecdh);
  mbedtls_entropy_init(&entropy);
  mbedtls_ctr_drbg_init(&ctr_drbg);

  ecdh_initialized = true;

  ret = mbedtls_ctr_drbg_seed(
    &ctr_drbg,
    mbedtls_entropy_func,
    &entropy,
    (const unsigned char*)pers,
    strlen(pers)
  );

  if (ret != 0) {
    Serial.print("Error CTR_DRBG. Codigo: ");
    Serial.println(ret);
    liberarECDH();
    return false;
  }

  ret = mbedtls_ecdh_setup(
    &ecdh,
    MBEDTLS_ECP_DP_SECP256R1
  );

  if (ret != 0) {
    Serial.print("Error ECDH setup. Codigo: ");
    Serial.println(ret);
    liberarECDH();
    return false;
  }

  ret = mbedtls_ecdh_make_public(
    &ecdh,
    &node_public_key_len,
    node_public_key,
    sizeof(node_public_key),
    mbedtls_ctr_drbg_random,
    &ctr_drbg
  );

  if (ret != 0) {
    Serial.print(
      "Error generando clave publica ECDH. Codigo: "
    );

    Serial.println(ret);
    liberarECDH();
    return false;
  }
```

```
  Serial.println("ECDH inicializado correctamente");
  return true;
}

bool reiniciarSesionECDH() {
  ecdh_ready = false;
  node_public_sent = false;
  node_public_key_len = 0;
  last_gateway_public_hex[0] = '\0';

  memset(key, 0, sizeof(key));
  memset(node_public_key, 0, sizeof(node_public_key));

  liberarECDH();

  if (!iniciarECDH()) {
    Serial.println("No se pudo reiniciar la sesion ECDH");
    mostrarEstadoASCON("ECDH ERR");
    return false;
  }

  Serial.println("Nueva sesion ECDH generada");
  mostrarEstadoASCON("WAIT KEY");

  return true;
}

void procesarClavePublicaGateway(const char* payload) {
  char node_rx[32];
  char gateway_pub_hex[201];

  if (!extract_json_value(
        payload,
        "node",
        node_rx,
        sizeof(node_rx)
      )) {
    Serial.println("No se encontro node en respuesta ECDH");
    return;
  }

  if (strcmp(node_rx, DEV_ID) != 0) {
    Serial.println("Respuesta ECDH para otro nodo");
    return;
  }

  if (!extract_json_value(
        payload,
        "pub",
        gateway_pub_hex,
        sizeof(gateway_pub_hex)
      )) {
    Serial.println("No se encontro pub en respuesta ECDH");
    return;
  }

  if (
    ecdh_ready &&
    strcmp(
      last_gateway_public_hex,
```

```
    gateway_pub_hex
  ) == 0
) {
  Serial.println("Clave publica del gateway ya procesada");
  return;
}

uint8_t gateway_pub[100];

int gateway_pub_len = hex_to_bytes(
  gateway_pub_hex,
  gateway_pub,
  sizeof(gateway_pub)
);

if (gateway_pub_len <= 0) {
  Serial.println("Clave publica gateway invalida");
  return;
}

uint8_t gateway_pub_mbedtls[101];
size_t gateway_pub_mbedtls_len = 0;

if (
  gateway_pub_len == 65 &&
  gateway_pub[0] == 0x04
) {
  gateway_pub_mbedtls[0] = 65;

  memcpy(
    gateway_pub_mbedtls + 1,
    gateway_pub,
    (size_t)gateway_pub_len
  );

  gateway_pub_mbedtls_len =
    (size_t)gateway_pub_len + 1;
} else {
  memcpy(
    gateway_pub_mbedtls,
    gateway_pub,
    (size_t)gateway_pub_len
  );

  gateway_pub_mbedtls_len =
    (size_t)gateway_pub_len;
}

int ret = mbedtls_ecdh_read_public(
  &ecdh,
  gateway_pub_mbedtls,
  gateway_pub_mbedtls_len
);

if (ret != 0) {
  Serial.print(
    "Error leyendo clave publica gateway. Codigo: "
  );

  Serial.println(ret);
  return;
```

```
}

uint8_t secret_bytes[128];
size_t secret_len = 0;

ret = mbedtls_ecdh_calc_secret(
  &ecdh,
  &secret_len,
  secret_bytes,
  sizeof(secret_bytes),
  mbedtls_ctr_drbg_random,
  &ctr_drbg
);

if (ret != 0) {
  Serial.print(
    "Error calculando secreto compartido. Codigo: "
  );

  Serial.println(ret);
  return;
}

uint8_t hash[32];

mbedtls_sha256(
  secret_bytes,
  secret_len,
  hash,
  0
);

memcpy(key, hash, 16);

strncpy(
  last_gateway_public_hex,
  gateway_pub_hex,
  sizeof(last_gateway_public_hex) - 1
);

last_gateway_public_hex[
  sizeof(last_gateway_public_hex) - 1
] = '\0';

ecdh_ready = true;

uint8_t key_fingerprint[32];
char key_fingerprint_hex[17];

mbedtls_sha256(
  key,
  sizeof(key),
  key_fingerprint,
  0
);

to_hex(
  key_fingerprint,
  8,
  key_fingerprint_hex
);
```

```
  Serial.println(
    "Clave ASCON derivada correctamente por ECDH"
  );

  Serial.print("Huella clave ASCON: ");
  Serial.println(key_fingerprint_hex);

  mostrarEstadoASCON("ECDH OK");
}

// ============================================================
// ASCON
// ============================================================

void generate_iv() {
  for (int i = 0; i < 16; i++) {
    iv[i] = (uint8_t)(esp_random() & 0xFF);
  }
}

int ascon_encrypt(
  const uint8_t* key_in,
  const uint8_t* iv_in,
  const uint8_t* plaintext_in,
  size_t plen,
  uint8_t* ciphertext_out
) {
  Ascon128 ascon;

  if (!ascon.setKey(key_in, 16)) {
    return -1;
  }

  if (!ascon.setIV(iv_in, 16)) {
    return -1;
  }

  ascon.encrypt(
    ciphertext_out,
    plaintext_in,
    plen
  );

  uint8_t tag[16];
  ascon.computeTag(tag, 16);

  memcpy(
    ciphertext_out + plen,
    tag,
    16
  );

  return (int)plen + 16;
}

uint64_t medirASCONPromedio(
  size_t plen,
  int repeticiones,
  int* ciphertext_len_final
) {
```

```
  uint64_t suma_us = 0;
  int ciphertext_len = -1;

  for (int i = 0; i < repeticiones; i++) {
    generate_iv();
    registrarSRAMProceso();

    int64_t inicio_us = esp_timer_get_time();

    ciphertext_len = ascon_encrypt(
      key,
      iv,
      plaintext,
      plen,
      ciphertext
    );

    int64_t fin_us = esp_timer_get_time();

    registrarSRAMProceso();

    if (ciphertext_len > 0) {
      suma_us += (uint64_t)(fin_us - inicio_us);
      ascon_control ^= ciphertext[0];
    }
  }

  *ciphertext_len_final = ciphertext_len;
  return suma_us;
}

// ============================================================
// WIFI
// ============================================================

void setup_wifi() {
  Serial.print("Conectando a WiFi: ");
  Serial.println(ssid);

  mostrarEstadoASCON("START");

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.print("WiFi conectado. IP: ");
  Serial.println(WiFi.localIP());
}

// ============================================================
// MQTT
// ============================================================

void callback(
  char* topic,
  byte* payload,
```

```
  unsigned int length
) {
  char msg[600];

  if (length >= sizeof(msg)) {
    Serial.println("Mensaje MQTT muy grande");
    return;
  }

  memcpy(msg, payload, length);
  msg[length] = '\0';

  Serial.print("MQTT recibido en topic: ");
  Serial.println(topic);

  if (strcmp(topic, TOPIC_KEY_GATEWAY) == 0) {
    Serial.println("Clave publica gateway recibida");
    procesarClavePublicaGateway(msg);
  }
}

void publicarClavePublicaNodo() {
  if (!ecdh_initialized || node_public_key_len == 0) {
    Serial.println("ECDH no inicializado");
    node_public_sent = false;
    return;
  }

  char pub_hex[201];

  to_hex(
    node_public_key,
    node_public_key_len,
    pub_hex
  );

  char msg[300];

  snprintf(
    msg,
    sizeof(msg),
    "{\"node\":\"%s\",\"pub\":\"%s\"}",
    DEV_ID,
    pub_hex
  );

  bool ok = client.publish(
    TOPIC_KEY_NODE,
    msg,
    false
  );

  Serial.print("Clave publica nodo enviada: ");
  Serial.println(ok ? "OK" : "FALLO");

  node_public_sent = ok;
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Conectando a MQTT... ");
```

```
    String clientId = String("nodo-") + DEV_ID;

    if (client.connect(clientId.c_str())) {
      Serial.println("conectado");

      client.subscribe(
        TOPIC_KEY_GATEWAY,
        1
      );

      Serial.print("Suscrito a: ");
      Serial.println(TOPIC_KEY_GATEWAY);

      if (!reiniciarSesionECDH()) {
        client.disconnect();
        delay(2000);
        continue;
      }

      char status_msg[128];

      snprintf(
        status_msg,
        sizeof(status_msg),
        "{\"node\":\"%s\",\"status\":\"online\"}",
        DEV_ID
      );

      client.publish(
        TOPIC_STATUS,
        status_msg,
        false
      );

      delay(500);
      client.loop();

      publicarClavePublicaNodo();
      lastKeySend = millis();

    } else {
      Serial.print("fallo, rc=");
      Serial.println(client.state());
      delay(5000);
    }
  }
}

// ============================================================
// INFORMACION DEL HARDWARE
// ============================================================

void mostrarInformacionPlaca() {
  Serial.println();
  Serial.println("====================================");
  Serial.println("      INFORMACION DEL DISPOSITIVO");
  Serial.println("====================================");

  Serial.print("Identificador del nodo: ");
  Serial.println(DEV_ID);
```

```
Serial.print("Modelo del chip: ");
Serial.println(ESP.getChipModel());

Serial.print("Frecuencia real de CPU: ");
Serial.print(ESP.getCpuFreqMHz());
Serial.println(" MHz");

Serial.print("Cantidad de nucleos: ");
Serial.println(ESP.getChipCores());

Serial.print("Revision del chip: ");
Serial.println(ESP.getChipRevision());

Serial.print("Memoria flash: ");
Serial.print(
  ESP.getFlashChipSize() / (1024UL * 1024UL)
);
Serial.println(" MB");

Serial.print("Velocidad de flash: ");
Serial.print(
  ESP.getFlashChipSpeed() / 1000000UL
);
Serial.println(" MHz");

Serial.print("PSRAM total: ");
Serial.print(
  ESP.getPsramSize()
);
Serial.println(" bytes");

Serial.print("PSRAM libre: ");
Serial.print(
  ESP.getFreePsram()
);
Serial.println(" bytes");

Serial.print("Heap libre al iniciar: ");
Serial.print(
  ESP.getFreeHeap()
);
Serial.println(" bytes");

Serial.print("Heap minimo historico: ");
Serial.print(
  ESP.getMinFreeHeap()
);
Serial.println(" bytes");

Serial.print("SRAM interna libre: ");
Serial.print(
  heap_caps_get_free_size(
    MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT
  )
);
Serial.println(" bytes");

Serial.print("SRAM interna total asignable: ");
Serial.print(
  heap_caps_get_total_size(
```

```
      MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT
    )
  );
  Serial.println(" bytes");

  Serial.print("Tamano del programa: ");
  Serial.print(
    ESP.getSketchSize() / 1024UL
  );
  Serial.println(" KiB");

  Serial.print("Espacio libre para programa: ");
  Serial.print(
    ESP.getFreeSketchSpace() / 1024UL
  );
  Serial.println(" KiB");

  Serial.println("====================================");
  Serial.println();
}

// ============================================================
// SETUP
// ============================================================

void setup() {
  Serial.begin(115200);
  delay(2500);

  // Esta prueba se ejecuta una sola vez y queda fuera
  // de la medicion del tiempo de cifrado ASCON.
  mostrarInformacionPlaca();

  snprintf(
    TOPIC_KEY_GATEWAY,
    sizeof(TOPIC_KEY_GATEWAY),
    "tesis/ascon/key/gateway/%s",
    DEV_ID
  );

  Wire.begin(OLED_SDA, OLED_SCL);

  if (!display.begin(
        SSD1306_SWITCHCAPVCC,
        0x3C
      )) {
    Serial.println("Error al iniciar OLED");
  } else {
    mostrarEstadoASCON("START");
  }

  iniciarINA219();
  dht.begin();
  setup_wifi();

  client.setServer(
    mqtt_server,
    MQTT_PORT
  );

  client.setCallback(callback);
```

```
  client.setBufferSize(1536);

  Serial.println(
    "Nodo HELTEC ECDH + ASCON con medicion de SRAM iniciado"
  );
}

// ============================================================
// LOOP
// ============================================================

void loop() {
  if (!client.connected()) {
    reconnect();
  }

  client.loop();

  unsigned long now = millis();

  if (!ecdh_ready) {
    mostrarEstadoASCON("WAIT KEY");

    if (
      now - lastKeySend >=
      KEY_RETRY_INTERVAL_MS
    ) {
      lastKeySend = now;

      if (client.connected()) {
        publicarClavePublicaNodo();
      }
    }

    delay(100);
    return;
  }

  if (
    now - lastKeySend >=
    KEY_HEARTBEAT_INTERVAL_MS
  ) {
    lastKeySend = now;

    if (client.connected()) {
      publicarClavePublicaNodo();
    }
  }

  if (
    ina_ok &&
    now - lastPowerSample >=
    POWER_SAMPLE_INTERVAL_MS
  ) {
    lastPowerSample = now;
    leerINA219(now);
  }

  if (
    now - lastPublish <
    PUBLISH_INTERVAL_MS
```

```
) {
  return;
}

lastPublish = now;

float temp = dht.readTemperature();
float hum = dht.readHumidity();

if (isnan(temp) || isnan(hum)) {
  Serial.println("Error al leer sensor DHT");
  mostrarEstadoASCON("DHT ERR");
  return;
}

int plen = snprintf(
  (char*)plaintext,
  sizeof(plaintext),
  "{\"node\":\"%s\",\"temp\":%.1f,\"hum\":%.1f}",
  DEV_ID,
  temp,
  hum
);

if (
  plen <= 0 ||
  plen >= (int)sizeof(plaintext)
) {
  Serial.println("Error: buffer plaintext insuficiente");
  mostrarEstadoASCON("BUF ERR");
  return;
}

leerINA219(millis());

float power_ini_mw =
  ina_ok ? fabs(ina_power_mw) : 0.0;

float current_ini_ma =
  ina_ok ? fabs(ina_current_ma) : 0.0;

// La ventana de memoria comienza justo antes de generar
// los nonces y ejecutar las repeticiones de ASCON.

iniciarMedicionSRAMProceso();
registrarSRAMProceso();

int ciphertext_len = 0;

uint64_t proc_total_us = medirASCONPromedio(
  (size_t)plen,
  REPETICIONES_ASCON,
  &ciphertext_len
);

registrarSRAMProceso();

leerINA219(millis());

float power_fin_mw =
  ina_ok ? fabs(ina_power_mw) : 0.0;
```

```
float current_fin_ma =
  ina_ok ? fabs(ina_current_ma) : 0.0;

float power_ascon_mw = 0.0;
float current_ascon_ma = 0.0;

if (ina_ok) {
  power_ascon_mw =
    (power_ini_mw + power_fin_mw) / 2.0;

  current_ascon_ma =
    (current_ini_ma + current_fin_ma) / 2.0;
}

float proc_prom_us =
  (float)proc_total_us /
  (float)REPETICIONES_ASCON;

float proc_prom_ms =
  proc_prom_us / 1000.0;

int bits_cifrados = plen * 8;
int paquete_seguro_bytes =
  16 + ciphertext_len;

float energia_ascon_uJ =
  power_ascon_mw * proc_prom_ms;

float energia_nJ_bit = 0.0;

if (bits_cifrados > 0) {
  energia_nJ_bit =
    (energia_ascon_uJ /
     bits_cifrados) * 1000.0;
}

char encrypted_hex[321] = "";
char ascon_status[24] = "";

if (ciphertext_len < 0) {
  strcpy(ascon_status, "ERROR");
  strcpy(encrypted_hex, "ERROR");
} else {
  uint8_t outbuf[160];
  size_t outlen =
    16 + (size_t)ciphertext_len;

  memcpy(outbuf, iv, 16);

  memcpy(
    outbuf + 16,
    ciphertext,
    (size_t)ciphertext_len
  );

  registrarSRAMProceso();

  to_hex(
    outbuf,
    outlen,
```

```
    encrypted_hex
  );

  strcpy(ascon_status, "ASCON_OK");
}

registrarSRAMProceso();

mostrarMetricasASCON(
  ascon_status,
  proc_prom_us,
  energia_ascon_uJ
);

char sensor_msg[800];

snprintf(
  sensor_msg,
  sizeof(sensor_msg),
  "{\"node\":\"%s\","
  "\"mode\":\"ascon\","
  "\"temp\":%.1f,"
  "\"hum\":%.1f,"
  "\"encrypted\":\"%s\","
  "\"decrypted\":null,"
  "\"nonce\":\"%.*s\","
  "\"ecdh_status\":\"CORRECTO\","
  "\"ascon_status\":\"%s\","
  "\"key_mode\":\"ECDH\"}",
  DEV_ID,
  temp,
  hum,
  encrypted_hex,
  32,
  encrypted_hex,
  ascon_status
);

registrarSRAMProceso();

bool ok_sensor = client.publish(
  TOPIC_SENSOR,
  sensor_msg,
  false
);

registrarSRAMProceso();
finalizarMedicionSRAMProceso();

// El mensaje de metricas se crea despues de cerrar la ventana
// para evitar que su propio buffer altere el valor medido.

char metrics_msg[1400];

snprintf(
  metrics_msg,
  sizeof(metrics_msg),
  "{\"node\":\"%s\","
  "\"mode\":\"ascon\","
  "\"ascon_reps\":%d,"
  "\"plain_bytes\":%d,"
```

```
"\"bits_cifrados\":%d,"
"\"cipher_bytes\":%d,"
"\"secure_packet_bytes\":%d,"
"\"proc_total_us\":%llu,"
"\"proc_us\":%.3f,"
"\"proc_prom_us\":%.3f,"
"\"proc_prom_ms\":%.6f,"
"\"power_ascon_mw\":%.3f,"
"\"current_ascon_ma\":%.3f,"
"\"energy_ascon_uJ\":%.6f,"
"\"energy_uj\":%.6f,"
"\"energy_nJ_bit\":%.6f,"
"\"sram_total_dynamic\":%lu,"
"\"sram_free_initial\":%lu,"
"\"sram_free_min_process\":%lu,"
"\"sram_used\":%lu,"
"\"sram_process_delta\":%lu,"
"\"sram_occupied_peak\":%lu,"
"\"heap_free\":%lu,"
"\"heap_min\":%lu,"
"\"payload_size\":%u,"
"\"mqtt_ok\":%s,"
"\"ina_ok\":%s,"
"\"bus_v\":%.3f,"
"\"load_v\":%.3f,"
"\"shunt_mv\":%.3f,"
"\"current_ma\":%.3f,"
"\"current_abs_ma\":%.3f,"
"\"power_mw\":%.3f,"
"\"charge_mAh\":%.6f,"
"\"energy_mWh\":%.6f,"
"\"avg_current_ma\":%.3f,"
"\"est_runtime_h\":%.2f}",
DEV_ID,
REPETICIONES_ASCON,
plen,
bits_cifrados,
ciphertext_len,
paquete_seguro_bytes,
(unsigned long long)proc_total_us,
proc_prom_us,
proc_prom_us,
proc_prom_ms,
power_ascon_mw,
current_ascon_ma,
energia_ascon_uJ,
energia_ascon_uJ,
energia_nJ_bit,
(unsigned long)sram_total_dinamica,
(unsigned long)sram_libre_inicio,
(unsigned long)sram_libre_minima,
(unsigned long)sram_incremento_proceso,
(unsigned long)sram_incremento_proceso,
(unsigned long)sram_ocupada_maxima,
(unsigned long)ESP.getFreeHeap(),
(unsigned long)ESP.getMinFreeHeap(),
(unsigned int)strlen(sensor_msg),
ok_sensor ? "true" : "false",
ina_ok ? "true" : "false",
ina_bus_v,
ina_load_v,
```

```
  ina_shunt_mv,
  ina_current_ma,
  ina_current_abs_ma,
  ina_power_mw,
  acc_mAh,
  acc_mWh,
  avg_current_mA,
  est_runtime_h
);

bool ok_metrics = client.publish(
  TOPIC_METRICS,
  metrics_msg,
  false
);

Serial.println("========== ENVIO ==========");

Serial.print("Nodo: ");
Serial.println(DEV_ID);

Serial.println("Modo: ASCON + ECDH");

Serial.print("ASCON status: ");
Serial.println(ascon_status);

Serial.print("Repeticiones ASCON: ");
Serial.println(REPETICIONES_ASCON);

Serial.print("Bytes cifrados reales: ");
Serial.println(plen);

Serial.print("Bits cifrados reales: ");
Serial.println(bits_cifrados);

Serial.print("Ciphertext + tag bytes: ");
Serial.println(ciphertext_len);

Serial.print(
  "Paquete seguro bytes IV + ciphertext + tag: "
);

Serial.println(paquete_seguro_bytes);

Serial.print("Tiempo total ASCON: ");
Serial.printf(
  "%llu us\n",
  (unsigned long long)proc_total_us
);

Serial.print("Tiempo promedio ASCON: ");
Serial.print(proc_prom_us, 3);
Serial.println(" us");

Serial.print("Potencia total durante ASCON: ");
Serial.print(power_ascon_mw, 3);
Serial.println(" mW");

Serial.print("Corriente total durante ASCON: ");
Serial.print(current_ascon_ma, 3);
Serial.println(" mA");
```

```
  Serial.print("Energia estimada durante ASCON: ");
  Serial.print(energia_ascon_uJ, 6);
  Serial.println(" uJ");

  Serial.print("Energia por bit: ");
  Serial.print(energia_nJ_bit, 6);
  Serial.println(" nJ/bit");

  Serial.println("----- SRAM DINAMICA -----");

  Serial.print("SRAM dinamica total: ");
  Serial.print((unsigned long)sram_total_dinamica);
  Serial.println(" bytes");

  Serial.print("SRAM libre inicial: ");
  Serial.print((unsigned long)sram_libre_inicio);
  Serial.println(" bytes");

  Serial.print("SRAM libre minima observada: ");
  Serial.print((unsigned long)sram_libre_minima);
  Serial.println(" bytes");

  Serial.print("SRAM adicional maxima del proceso: ");
  Serial.print((unsigned long)sram_incremento_proceso);

  Serial.print("SRAM dinamica ocupada en el pico: ");
  Serial.print((unsigned long)sram_ocupada_maxima);
  Serial.println(" bytes");
  Serial.print("Incremento durante el proceso: ");


 Serial.print((unsigned long)sram_incremento_proceso);
  Serial.println(" bytes");

  Serial.print("MQTT sensor: ");
  Serial.println(ok_sensor ? "OK" : "FALLO");

  Serial.print("MQTT metrics: ");
  Serial.println(ok_metrics ? "OK" : "FALLO");

  Serial.print("Encrypted HEX: ");
  Serial.println(encrypted_hex);

  Serial.println("============================");
}
```

ANEXO G. Código fuente del gateway del sistema IoT

En este anexo se presenta el código fuente implementado en la Raspberry Pi 3 para el funcionamiento del gateway del sistema IoT; el programa gestiona la comunicación mediante MQTT, el intercambio de claves públicas mediante ECDH, la derivación de la clave utilizada por ASCON-128 y el descifrado de los mensajes recibidos

mediante la biblioteca ASCON-C; además, incorpora el registro de las métricas obtenidas, el almacenamiento de la información y su integración con la interfaz web desarrollada mediante Flask.

```
from flask import Flask, render_template, send_file, request
import pandas as pd
from flask_socketio import SocketIO
import paho.mqtt.client as mqtt
import json
import time
import hashlib
import ctypes
from datetime import datetime
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization
from database import (
init_db,
save_sensor_data,
get_limited_sensor_data,
save_metrics_data,
get_limited_metrics_data,
save_decrypt_metrics,
get_limited_decrypt_metrics
)

# ============================================================
# FLASK
# ============================================================

app = Flask(__name__)

socketio = SocketIO(
app,
cors_allowed_origins="*"
)

sensor_data = {}

# ============================================================
# MQTT
# ============================================================

MQTT_BROKER = "localhost"
MQTT_PORT = 1883

TOPIC_SENSOR = "tesis/ascon/sensor"
TOPIC_METRICS = "tesis/ascon/metrics"
TOPIC_KEY_NODE = "tesis/ascon/key/node"

# ============================================================
# ECDH / ASCON
# ============================================================
ASCON_KEYS = {}
GATEWAY_PUBLIC_KEYS = {}
REPETICIONES_DESCIFRADO = 1000

# ============================================================
```

```
# BIBLIOTECA ASCON-C
# ============================================================

LIB_PATH = (
"/home/admin/envtesis/ascon-c/libascon128.so"
)
try:
lib = ctypes.CDLL(
LIB_PATH
)
print(
"Biblioteca ASCON-C cargada:"
)
print(
LIB_PATH
)

except Exception as e:
print(
"ERROR cargando biblioteca ASCON-C:"
)
print(e)
lib = None

# ============================================================
# PROTOTIPO crypto_aead_decrypt()
# ============================================================
#
# Firma C:
#
# int crypto_aead_decrypt(
#     unsigned char* m,
#     unsigned long long* mlen,
#     unsigned char* nsec,
#     const unsigned char* c,
#     unsigned long long clen,
#     const unsigned char* ad,
#     unsigned long long adlen,
#     const unsigned char* npub,
#     const unsigned char* k
# );
#
# ============================================================

if lib is not None:
lib.crypto_aead_decrypt.argtypes = [
ctypes.POINTER(
ctypes.c_ubyte
),
ctypes.POINTER(
ctypes.c_ulonglong
),
ctypes.POINTER(
ctypes.c_ubyte
),
ctypes.POINTER(
ctypes.c_ubyte
),
ctypes.c_ulonglong,
ctypes.POINTER(
ctypes.c_ubyte
```

```
),
ctypes.c_ulonglong,
ctypes.POINTER(
ctypes.c_ubyte
),
ctypes.POINTER(
ctypes.c_ubyte
)
]
lib.crypto_aead_decrypt.restype = ctypes.c_int

# ============================================================
# UTILIDADES HEX
# ============================================================

def bytes_to_hex(data):
return data.hex().upper()
def hex_to_bytes(data):
return bytes.fromhex(data)
# ============================================================
# ECDH
# ============================================================

def generar_clave_ecdh_para_nodo(
node,
node_public_hex
):
try:

node_public_bytes = hex_to_bytes(
node_public_hex
)
if (
len(node_public_bytes) == 66
and node_public_bytes[0] == 65
):
node_public_bytes = (
node_public_bytes[1:]
)
if (
len(node_public_bytes) != 65
or node_public_bytes[0] != 0x04
):
print(
"Clave pública nodo inválida"
)
return None
node_public_key = (
ec.EllipticCurvePublicKey
.from_encoded_point(
ec.SECP256R1(),
node_public_bytes
)
)
gateway_private_key = (
ec.generate_private_key(
ec.SECP256R1()
)
)
shared_secret = (
gateway_private_key.exchange(
```

```
ec.ECDH(),
node_public_key
)
)
hash_key = hashlib.sha256(
shared_secret
).digest()
ascon_key = hash_key[:16]
ASCON_KEYS[node] = (
ascon_key
)
gateway_public_key = (
gateway_private_key.public_key()
)
gateway_public_bytes = (
gateway_public_key.public_bytes(
encoding=serialization.Encoding.X962,
format=serialization.PublicFormat.UncompressedPoint
)
)
gateway_public_hex = (
bytes_to_hex(
gateway_public_bytes
)
)
GATEWAY_PUBLIC_KEYS[node] = (
gateway_public_hex
)
print(
"ECDH OK para nodo:",
node
)
print(
"ASCON KEY:",
bytes_to_hex(
ascon_key
)
)
return gateway_public_hex
except Exception as e:
print(
"Error ECDH:",
e
)
return None

# ============================================================
# DESCIFRADO ASCON-C
# ============================================================

def descifrar_ascon_c(
key,
iv,
ciphertext_tag
):
if lib is None:
print(
"ERROR: biblioteca ASCON-C no disponible"
)
return (
None,
```

```
0.0,
-1
)
if len(key) != 16:

print(
"ERROR: la clave ASCON debe tener 16 bytes"
)
return (
None,
0.0,
-1
)
if len(iv) != 16:

print(
"ERROR: el IV debe tener 16 bytes"
)
return (
None,
0.0,
-1
)
if len(ciphertext_tag) < 16:
print(
"ERROR: ciphertext insuficiente"
)
return (
None,
0.0,
-1
)
# ------------------------------------------------------
# BUFFER CIPHERTEXT
# ------------------------------------------------------

cipher_buffer = (
ctypes.c_ubyte *
len(ciphertext_tag)
).from_buffer_copy(
ciphertext_tag
)
# ------------------------------------------------------
# BUFFER KEY
# ------------------------------------------------------

key_buffer = (
ctypes.c_ubyte *
len(key)
).from_buffer_copy(
key
)
# ------------------------------------------------------
# BUFFER IV / NONCE
# ------------------------------------------------------
iv_buffer = (
ctypes.c_ubyte *
len(iv)
).from_buffer_copy(
iv
)
```

```
# ------------------------------------------------------
# BUFFER PLAINTEXT
# ------------------------------------------------------
plaintext_buffer = (
ctypes.c_ubyte *
len(ciphertext_tag)
)()
plaintext_len = (
ctypes.c_ulonglong(0)
)

# ------------------------------------------------------
# DESCIFRADO REPETIDO
# ------------------------------------------------------
inicio = time.perf_counter_ns()
resultado = -1
for _ in range(
REPETICIONES_DESCIFRADO
):
resultado = (
lib.crypto_aead_decrypt(
plaintext_buffer,
ctypes.byref(
plaintext_len
),
None,
cipher_buffer,
len(ciphertext_tag),
None,
0,
iv_buffer,
key_buffer
)
)
if resultado != 0:

break
fin = time.perf_counter_ns()
tiempo_total_us = (
fin - inicio
) / 1000.0
if resultado != 0:
return (
None,
tiempo_total_us,
resultado
)
plaintext = bytes(
plaintext_buffer[
:plaintext_len.value
]
)
tiempo_prom_us = (
tiempo_total_us
/ REPETICIONES_DESCIFRADO
)
print(
"================================="
)
print(
"DESCIFRADO ASCON-C"
```

```
)
print(
"Bytes descifrados:",
len(plaintext)
)
print(
"Bits descifrados:",
len(plaintext) * 8
)
print(
"Repeticiones:",
REPETICIONES_DESCIFRADO
)
print(
"Tiempo total:",
f"{tiempo_total_us:.3f}",
"us"
)
print(
"Tiempo promedio:",
f"{tiempo_prom_us:.3f}",
"us"
)
print(
"================================="
)
return (
plaintext,
tiempo_total_us,
resultado
)
# ============================================================
# DESCIFRADO DESDE HEX
# ============================================================

def decrypt_ascon_from_hex(
node,
encrypted_hex
):
try:
if node not in ASCON_KEYS:
print(
"No existe clave ECDH para:",
node
)
return None
if (
not encrypted_hex
or encrypted_hex == "SIN_CIFRADO"
):
return None
raw = bytes.fromhex(
encrypted_hex
)
if len(raw) < 32:
print(
"Paquete cifrado demasiado pequeño"
)
return None
# ------------------------------------------------
# SEPARAR NONCE
```

```
# ------------------------------------------------
iv = raw[:16]
# ------------------------------------------------
# SEPARAR CIPHERTEXT + TAG
# ------------------------------------------------
ciphertext_tag = raw[16:]
# ------------------------------------------------
# DESCIFRADO
# ------------------------------------------------
(
plaintext,
total_us,
resultado
) = descifrar_ascon_c(
ASCON_KEYS[node],
iv,
ciphertext_tag
)
bytes_dec = (
len(plaintext)
if plaintext is not None
else 0
)
bits_dec = (
bytes_dec * 8
)
proc_us = (
total_us
/ REPETICIONES_DESCIFRADO
if resultado == 0
else 0
)
status = (
"OK"
if resultado == 0
else "ERROR"
)
# ------------------------------------------------
# GUARDAR METRICAS
# ------------------------------------------------
decrypt_data = {
"timestamp":
datetime.now().strftime(
"%Y-%m-%d %H:%M:%S"
),

"node":
node,

"bytes_decrypted":
bytes_dec,

"bits_decrypted":
bits_dec,

"repetitions":
REPETICIONES_DESCIFRADO,

"total_us":
total_us,
```

```
"proc_us":
proc_us,

"status":
status
}
save_decrypt_metrics(
decrypt_data
)
# ------------------------------------------------
# RESULTADO
# ------------------------------------------------
print()
print(
"======================================"
)
print(
"DESCIFRADO ASCON EN GATEWAY"
)
print(
"Nodo:",
node
)
print(
"Resultado:",
resultado
)
print(
"Bytes descifrados:",
bytes_dec
)
print(
"Bits descifrados:",
bits_dec
)
print(
"Tiempo total:",
f"{total_us:.3f}",
"us"
)
print(
"Tiempo promedio:",
f"{proc_us:.3f}",
"us"
)
print(
"Estado:",
status
)
print(
"======================================"
)
if plaintext is None:

return None
return plaintext.decode(
"utf-8"
)
except Exception as e:
print(
"Error descifrando ASCON:",
```

```
e
)
return None
# ============================================================
# MQTT CONNECT
# ============================================================
def on_connect(
client,
userdata,
flags,
rc
):
print(
"Conectado a MQTT, codigo:",
rc
)
client.subscribe(
TOPIC_SENSOR
)
client.subscribe(
TOPIC_METRICS
)
client.subscribe(
TOPIC_KEY_NODE
)
print(
"Suscrito a:",
TOPIC_SENSOR
)
print(
"Suscrito a:",
TOPIC_METRICS
)
print(
"Suscrito a:",
TOPIC_KEY_NODE
)
# ============================================================
# MQTT MESSAGE
# ============================================================
def on_message(
client,
userdata,
msg
):
try:
payload = (
msg.payload.decode(
"utf-8"
)
)
data = json.loads(
payload
)
# ================================================
# ECDH
# ================================================
if msg.topic == TOPIC_KEY_NODE:
node = data.get(
"node"
)
```

```
node_public_hex = data.get(
"pub"
)
if (
not node
or not node_public_hex
):
print(
"Mensaje ECDH invalido"
)
return
gateway_public_hex = (
generar_clave_ecdh_para_nodo(
node,
node_public_hex
)
)
if gateway_public_hex:

topic_response = (
f"tesis/ascon/key/gateway/{node}"
)
response = {

"node":
node,

"pub":
gateway_public_hex
}
client.publish(

topic_response,

json.dumps(
response,
separators=(
',',
':'
)
),

qos=1,

retain=False
)
print(
"Clave publica gateway enviada:",
topic_response
)
return
# ================================================
# TIMESTAMP
# ================================================
data["timestamp"] = (
datetime.now().strftime(
"%Y-%m-%d %H:%M:%S"
)
)
# ================================================
# DATOS SENSOR
```

```
# ================================================
if msg.topic == TOPIC_SENSOR:
node = data.get(
"node",
"desconocido"
)
encrypted_hex = data.get(
"encrypted"
)
if encrypted_hex:

decrypted_gateway = (
decrypt_ascon_from_hex(
node,
encrypted_hex
)
)
if decrypted_gateway:
try:
data["decrypted"] = (
json.loads(
decrypted_gateway
)
)
except Exception:

data["decrypted"] = (
decrypted_gateway
)
data["ascon_status"] = (
"OK_GATEWAY_ECDH"
)
else:
data["decrypted"] = (
"ERROR"
)
data["ascon_status"] = (
"ERROR_GATEWAY_ECDH"
)
sensor_data[node] = data
save_sensor_data(
data
)
print(
"Dato sensor recibido:",
data
)
socketio.emit(
"sensor_update",
data
)
# ================================================
# METRICAS DEL NODO
# ================================================
elif msg.topic == TOPIC_METRICS:
save_metrics_data(
data
)
print(
"Metrica recibida:",
data
```

```
)
socketio.emit(
"metrics_update",
data
)
except Exception as e:

print(
"Error procesando MQTT:",
e
)
# ============================================================
# FLASK
# ============================================================
@app.route("/")
def index():
return render_template(
"index.html",
sensor_data=sensor_data
)
# ============================================================
# EXPORTAR EXCEL
# ============================================================
@app.route(
"/descargar_excel"
)
def descargar_excel():
limit = request.args.get(
"limit",
"1000"
)
sensor = (
get_limited_sensor_data(
limit
)
)
metrics = (
get_limited_metrics_data(
limit
)
)
decrypt = (
get_limited_decrypt_metrics(
limit
)
)
columnas_sensor = [

"ID",
"Fecha_Hora",
"Nodo",
"Modo",
"Dato_Descifrado",
"Nonce",
"Estado_nonce",
"Estado_ECDH",
"Estado_ASCON"
]
columnas_metrics = [

"ID",
```

```
"Fecha_Hora",
"Nodo",
"Modo",
"Tiempo_procesamiento_us",
"SRAM_utilizado_bytes",
"Tamaño_payload_bytes",
"Corriente_mA",
"Potencia_mW",
"Energia_operacion_uJ"
]
columnas_decrypt = [

"ID",
"Fecha_Hora",
"Nodo",
"Bytes_descifrados",
"Bits_descifrados",
"Repeticiones",
"Tiempo_total_us",
"Tiempo_prom_us",
"Estado"
]
df_sensor = pd.DataFrame(
sensor,
columns=columnas_sensor
)
df_metrics = pd.DataFrame(
metrics,
columns=columnas_metrics
)
df_decrypt = pd.DataFrame(
decrypt,
columns=columnas_decrypt
)
archivo = (
f"datos_ascon_{limit}.xlsx"
)
with pd.ExcelWriter(
archivo,
engine="openpyxl"
) as writer:

df_sensor.to_excel(
writer,
sheet_name="Datos_Sensor",
index=False
)
df_metrics.to_excel(
writer,
sheet_name="Metricas_Placas",
index=False
)
df_decrypt.to_excel(
writer,
sheet_name="Descifrado_ASCON",
index=False
)
return send_file(
archivo,
as_attachment=True
)
```

```
# ============================================================
# INICIALIZACION
# ============================================================
init_db()
mqtt_client = mqtt.Client()
mqtt_client.on_connect = (
on_connect
)
mqtt_client.on_message = (
on_message
)
mqtt_client.connect(
MQTT_BROKER,
MQTT_PORT,
60
)
mqtt_client.loop_start()
# ============================================================
# EJECUTAR
# ============================================================
if __name__ == "__main__":
socketio.run(
app,
host="0.0.0.0",
port=5000,
debug=False,
use_reloader=False
)
```