Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América
Facultad de Ingeniería de Sistemas e Informática
Escuela Profesional de Ingeniería de Sistemas

Implementación de un modelo de criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing

TESIS
Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Jorge Leoncio GUERRA GUERRA

ASESOR

Dra. Nora Bertha LA SERNA PALOMINO

Lima, Perú

2026



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

https://creativecommons.org/licenses/by-nc-sa/4.0/

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Guerra, J. (2026). *Implementación de un modelo de criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing*. [Tesis de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Sistemas]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios autor/ asesor

Datos de autor	
Nombres y apellidos	Jorge Leoncio Guerra Guerra
Tipo de documento de identidad	DNI
Número de documento de identidad	08473333
URL de ORCID	https://orcid.org/0000-0003-2392-7821
Datos de asesor	
Nombres y apellidos	Dra. Nora Bertha La Serna Palomino
Tipo de documento de identidad	DNI
Número de documento de identidad	07665297
URL de ORCID	https://orcid.org/0000-0002-4292-344X
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Mg. Félix Armando Fermin Pérez
Tipo de documento	DNI
Número de documento de identidad	08736347
Miembro del jurado 1	
Nombres y apellidos	Raúl Armas Calderon
Tipo de documento	DNI
Número de documento de identidad	07156168
Datos de investigación	
Línea de investigación	INTERNET DE LAS COSAS

Grupo de investigación	Internet de las Cosas
Agencia de financiamiento	
Ubicación geográfica de la investigación	Lima
Año o rango de años en que se realizó la investigación	2023-2026
URL de disciplinas OCDE	Telecomunicaciones https://purl.org/pe-repo/ocde/ford#2.02.05

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA
Escuela Profesional de Ingeniería de Sistemas

Acta Virtual de Sustentación de Tesis

Siendo las 17:30 horas del día 29 de setiembre del año 2026, se reunieron virtualmente los docentes designados como miembros de Jurado de Tesis, presidido por el Mg. Felix Armando Fermin Pérez, Ing. Raúl Armas Caleron (Miembro) y la Dra. Nora Palomino La Serna (Miembro Asesor), usando la plataforma https://meet.google.com/awm-qwfr-qzk) para la sustentación Virtual de la tesis Intitulada: "**Implementación de un modelo de criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing**", del bachiller: **Jorge Leoncio Guerra Guerra;** para obtener el Título Profesional de Ingeniero de Sistemas.

Acto seguido de la exposición de la Tesis, el Presidente invitó al Bachiller a responder las preguntas formuladas por los Miembros del Jurado.

El Bachiller, en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez las preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el bachiller obtuvo la nota de 18 **(dieciocho)**

A continuación, el Presidente del Jurado Mg. Felix Armando Fermin Pérez, declara al bachiller **Ingeniero de Sistemas.**

Siendo 18:38 horas, se levantó la sesión.

Armando Fermín Pérez.

Mg. Félix Armando Fermin Pérez

Presidente

Ing. Raul Armas Calderon
Miembro

Dra. Nora La Serna Palomino
Miembro Asesor

UNIVERSIDAD NACIONAL MAYOR DE
SAN MARCOS
Universidad del Perú, Decana de América

Anexo 1
CERTIFICADO DE SIMILITUD

Yo, **Nora Bertha la Serna Palomino**, en mi condición de asesor acreditado con Resolución Directoral de la EP N°000006-2022-EPIS-FISI/UNMSM de la **tesis** cuyo título es **Implementación de un modelo de criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing**, presentado por el **bachiller Jorge Leoncio Guerra Guerra** para optar el **TITULO PROFESIONAL DE INGENIERO DE SISTEMAS**, CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el **12%** de similitud, nivel PERMITIDO para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional.**

Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del **TITULO PROFESIONAL DE INGENIERO DE SISTEMAS.**

Lima, de 30 de octubre del 2026

Firma del Asesor

Nombres y Apellidos del asesor: Nora Bertha La Serna Palomino

DNI: 07665297

* Según corresponda: bachiller, magíster, doctor; título profesional o título de segunda especialidad

La vigencia del presente certificado corresponde a un período de noventa (90) días desde la fecha de su emisión; siempre que no haya observaciones en el texto durante el proceso de revisión por el jurado informante o surja alguna contingencia.

DEDICATORIA:

Dedicado a mi esposa Mardeli, a mis hijos Jorge, Alex, Elena y Danitza, por ser mi inspiración para ser mejor persona y mejor profesional.

AGRADECIMIENTOS

En primer lugar, agradezco a Dios, por guiar cada uno de mis pasos y a mis padres por su apoyo incondicional y por todo el amor que me brindan.

También, agradezco a la Universidad Nacional Mayor de San Marcos por abrirme sus puertas y permitirme realizar el sueño de convertirme en profesional.

Por último, un especial agradecimiento a la Dra Nora La Serna Palomino, por su apoyo que fue fundamental en la realización de esta tesis.

RESUMEN

El desarrollo de aplicaciones en Internet de las Cosas (IoT) ha ayudado de manera significativa en diferentes ámbitos de la actividad humana, como en minería, agricultura, medicina, etc. Tal es su importancia, que las inversiones en IoT han ido en aumento, aun en tiemplo de COVID, como se mostrara en este trabajo.

Hay implementaciones IoT donde se están usando sensores y tarjetas de adquisición de datos, que tienen características técnicas que incluyen preprocesamiento de datos, y transmisión mediante bluetooth o wifi. Sin embargo, hay otro grupo de dispositivos denominados constraint devices, cuya reducida memoria y poder de calculo no permiten realizar más que tareas muy básicas, lo que podría significar riesgos a nivel de seguridad

El aporte de esta investigación radica en la implementación de un módulo de seguridad basado en criptografía destinado a la validación de datos en dispositivos ligeros, colocado en el componente Edge Computing, lo que permitirá un mecanismo IAM seguro y confiable al momento de ser enviados los datos hacia los servicios cloud de la capa de almacenamiento.

La propuesta presentada permitirá que se pueda utilizar tecnología open hardware y modelos de criptografía adaptados a los dispositivos ligeros que son ampliamente usados en soluciones IoT.

Palabras Claves: Internet de las Cosas, Criptografia, Constraint Devices, Edge Computing.

ABSTRACT

There are IoT implementations where sensors and data acquisition cards are being used, which have technical characteristics that include data preprocessing, and transmission via Bluetooth or Wi-Fi. However, there is another group of devices called constraint devices, whose reduced memory and calculation power do not allow more than very basic tasks to be carried out, which could mean security risks.

The contribution of this research lies in the implementation of a security module based on cryptography for data validation in light devices, placed in the Edge Computing component, which will allow a secure and reliable IAM mechanism when data is sent. towards the cloud services of the storage layer.

The proposal presented will allow the use of open hardware technology and cryptography models adapted to lightweight devices that are widely used in IoT solutions.

Keywords: Internet of Things, Cryptography, Constraint Devices, Edge Computing.

ÍNDICE GENERAL

RESUMEN **6**
ABSTRACT **7**
LISTA DE FIGURAS **10**
LISTA DE TABLAS **12**
INTRODUCCIÓN **13**
Capítulo I: Planteamiento Metodológico **15**
1.1. Antecedentes 15
1.2. Definición del Problema 19
1.3. Objetivos 19
1.3.1. Objetivo Principal 20
1.3.2. Objetivos Secundarios 20
1.4. Justificación 20
1.5. Alcance de la Investigación 21
1.6. Hipótesis 21
1.7. Variables e Indicadores 22
1.8. Población y Muestra 22
Capítulo II: Marco Teórico **25**
2.1. Bases Teóricas 25
2.1.1. Edge Computing 25
2.1.2. Modelos de Seguridad en Edge Computing 26
2.1.3. Dispositivos de Recursos Limitados (Constraint Devices) 27
2.1.4. Diferencias entre Fog Computing y Edge Computing 29
2.1.5. Modelos de Criptografía para Constraint Devices 30
2.2. Glosario de Términos 33
Capítulo III: Estado del Arte **34**
3.1. Revisión Bibliográfica de Literatura 34
3.1.1. Revisión Sistemática en SCOPUS 34
3.2. Determinación de la Solución 38
Capítulo IV: Aporte Teórico **40**
4.1. Esquema General de la Solución 40
4.1.1. Modelo de Implementación Hardware 40
4.1.2. Device Layer 41
4.1.3. Edge Layer 41
4.1.4. Cloud Layer 42

4.1.5. Modelo de Criptografía 42
4.2. Implementación de Software 43
4.3. Implementación 44
4.3.1. Capa Física 44
4.3.2. Capa Edge (Raspberry Pi 4) 50
4.3.3. Capa Cloud (Google Cloud) 54
4.3.4. Diagrama de Secuencia End-to-End 57
4.3.5. Consideraciones de Seguridad (Factores de Amenaza) 58
Capítulo V: Discusión de Resultados 59
5.1. Metodología de Evaluación 59
5.2. Indicadores de Seguridad y Eficiencia Criptográfica 60
5.2.1. Confidencialidad e Integridad de los Datos 60
5.2.2. Eficiencia de Tamaño de Clave 61
5.2.3. Overhead de Payload por Trama 61
5.2.4. Tiempos de Procesamiento Medidos 62
5.2.5. Análisis Comparativo frente a Implementaciones Previas 63
5.3. Revisión de los Resultados Obtenidos 64
5.3.1. Impacto en el Presupuesto de Latencia End-to-End 64
5.3.2. Análisis de Escalabilidad y Throughput 65
5.3.3. Síntesis de Mejoras Logradas 65
5.4. Evaluación Económica 66
5.4.1. Cotización de Recursos 66
5.4.2. Flujo de Caja 66
5.4.3. Valor Actual Neto (VAN) 67
5.4.4. Tasa Interna de Retorno (TIR) 67
Capítulo VI: Conclusiones y Recomendaciones 68
6.1. Conclusiones 68
6.2. Recomendaciones 69
REFERENCIAS BIBLIOGRÁFICAS 70

LISTA DE FIGURAS

Figura 1: Ataques a la capa física IoT **16**
Figura 2: Cálculo del tamaño de muestra infinita **22**
Figura 3: Estructura genérica de un Constraint Device **27**
Figura 4: Implementación física del proyecto de investigación (Device, Edge y Cloud Layer) **40**
Figura 5: Criptografía Asimétrica **42**
Figura 6: Combinación de ECC y RSA usando ECDH **43**
Figura 7: Circuito implementado Sensor 1 (ESP32 + DHT22) **44**
Figura 8: Recepción de datos enviados por el broker MQTT en RPi **46**
Figura 9: Código de creación de la base de datos SQLite **47**
Figura 10: Ejecución del código Python escucha_dbt.py **48**
Figura 11: Guardado de datos en SQLite **48**
Figura 12: Consulta de datos en SQLite **49**
Figura 13: Implementación del circuito Sensor 2 (ESP32 + Sensor de Humedad de Suelo) **49**
Figura 14: Diagrama y circuito del Sensor 3 (Arduino UNO + MQ135) **51**
Figura 15: Resultados del Sensor 3 ..... **53**
Figura 16: Hoja de cálculo 'lecturas_sensores' en Google Sheets tras varias ingestas exitosas **56**
Figura 17: Diagrama de secuencia completa End-to-End **57**
Figura 18: Tamaño de clave pública para nivel de seguridad equivalente (ECC vs. RSA) **61**
Figura 19: Overhead de payload por trama según esquema de cifrado **61**
Figura 20: Tiempo medido del esquema ECDH + AES-GCM en entorno de referencia **62**
Figura 21: Comparación normalizada de indicadores (escala 0-10) **63**

LISTA DE TABLAS

Tabla 1.1. Variables e indicadores de la investigación **22**
Tabla 2.1. Clases de Constraint devices según la norma RFC 7228 ..... **28**
Tabla 2.2. Características de IoT, amenazas, retos y oportunidades **31**
Tabla 3.1. Criterios de inclusión y exclusión en la revisión sistemática **34**
Tabla 3.2. Resultados de la pregunta de investigación Q1 (Técnicas criptográficas) **35**
Tabla 3.3. Papers de revisión para la pregunta Q2 (Criptografía en Edge Computing) **36**
Tabla 4.1. Componentes de la arquitectura por capa (Device, Edge y Cloud) **41**
Tabla 4.2. Conexiones del sensor DHT22 al ESP32 ..... **44**
Tabla 4.3. Cumplimiento de factores de amenaza e indicadores de seguridad en Google Cloud **58**
Tabla 5.1. Indicadores de evaluación de la propuesta **59**
Tabla 5.2. Valores de overhead de payload por trama según el esquema **61**
Tabla 5.3. Evaluación de tiempos medidos en el esquema ECDH + AES-GCM **62**
Tabla 5.4. Indicadores de evaluación comparativos frente a técnicas sin ECC **63**
Tabla 5.5. Presupuesto y evaluación de tiempos de envío de datos a Google Sheets **64**
Tabla 5.6. Cotización de los recursos de la solución propuesta **66**
Tabla 5.7. Flujo de caja del proyecto de implementación de la solución propuesta **66**

INTRODUCCIÓN

Según [Ahanger & Aljumah, 2019] el término «Internet de las cosas» (IoT) fue acuñado por Kevin Ashton en el año 1999 y más tarde formalmente en 2005 por la Unión Internacional de Telecomunicaciones (UIT), y desde entonces el aporte de las aplicaciones basadas en Internet de las Cosas ha sido muy importante en los últimos años, más aún, en el tiempo de pandemia el desarrollo de dispositivos IoT para temas de control de temperatura y medición de distanciamiento social han sido de gran ayuda para el control del COVID-19 y su propagación. Igualmente, en las tareas de mantenimiento de procesos industriales, durante el periodo de enclaustramiento de la población, donde la automatización de operaciones se hizo obligatorio, las soluciones IoT ayudaron de manera efectiva a tareas como gestión remota y alerta temprana, que permitieron que las empresas pudieran seguir operando, aun en las condiciones adversas de esos días; para ello se desarrollaron propuestas como [Zhi et al., 2020] en la que se desarrollan técnicas para evitar el ataque masivo de tipo productor-consumidor que afecta justamente a procesos industriales.

Además, debido al aumento de aplicaciones críticas, como eHealth o temas de contaminación acuífera, donde la toma de decisiones debe ser en tiempos muy cortos, el uso de Edge Computing se ha convertido en una implementación frecuente [Dongqi et al., 2022], impulsado aún más por los servicios de emergencia y de alerta temprana, usados en la reciente pandemia de la COVID-19. Sin embargo, en Perú, la implementación de la capa física de las aplicaciones se hace a través de dispositivos de bajo costo y de tecnología de cómputo reducida, denominada dispositivos restringidos (constraint devices) que no tienen la capacidad de ejecutar algoritmos ni procesos de criptografía o seguridad que sí se realizan en dispositivos de mayor poder de cómputo.

El objetivo de la investigación es implementar un modelo de seguridad basado en la construcción de un módulo de criptografía aplicado a dispositivos restringidos que se pueda implantar en la capa de edge computing generando un nivel de seguridad óptimo que sea equivalente a las implementaciones desarrolladas para dispositivos inteligentes (Smart devices) así como los servicios que existen en Computación en Nube.Esta tesis está organizada en 6 capítulos, detallados brevemente a continuación: El Capítulo I presenta el Planteamiento Metodológico de este trabajo de investigación, se hace mención a los antecedentes del problema tratado, así como la definición y justificación del mismo, además se detallan los objetivos y alcance del presente trabajo. El Capítulo II aborda el Marco Teórico donde se definen los conceptos necesarios para comprender las bases teóricas y el contexto sobre los cuales se desarrolla esta investigación. El Capítulo III comprende el Estado del Arte, en el cual se realiza una revisión bibliográfica, con el objetivo de identificar investigaciones en donde se hayan empleado métodos o herramientas alternativos a la solución propuesta en esta investigación para una problemática similar, además se comparan estos métodos con la solución propuesta. Mientras, el Capítulo IV titulado Aporte Teórico, describe las herramientas, metodologías, algoritmos y bibliotecas que fueron utilizados para el desarrollo de la solución de análisis emocional propuesta, también se detallan los módulos que la componen y como se integran. El Capítulo V aborda el Aporte Practico, donde se mencionan las herramientas tecnológicas utilizadas y como son aplicadas para la construcción de la solución propuesta, las interfaces de usuario, la determinación de la población y muestra, las pruebas del sistema y la validación económica de la solución. Finalmente, en el Capítulo VI se plantean las conclusiones obtenidas y las recomendaciones propuestas.

Capítulo I: Planteamiento Metodológico

1.1. Antecedentes

El uso de aplicaciones basadas en Internet de las Cosas se ha intensificado en los últimos años, tal como indica [Lara et al., 2021], y que además muestran, a través de mapas de calor, la penetración de IoT en el mercado y cómo es utilizado como estrategia de negocios. Por otro lado, en [Ray, 2018] se presenta un estudio bastante completo sobre la arquitectura IoT y las aplicaciones que están disponibles, destacando mecanismos de seguridad para cada capa.

En el trabajo presentado en [Kaur et al., 2023] se muestra una revisión sobre las diferentes amenazas y ataques a soluciones basadas en Internet de las Cosas, considerando las diferentes capas de la arquitectura. En la Figura 1 se muestra los ataques a la capa física IoT identificados por los autores [Kaur et al., 2023]

:

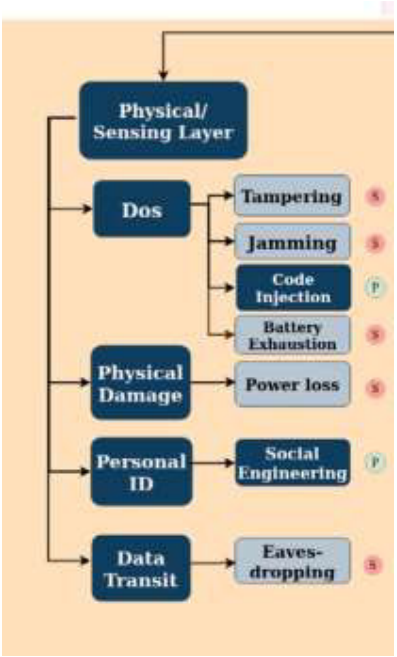


Fig 1: Ataques a la capa física IoT (fragmento. de [p33]

La preocupación por desarrollar criterios de seguridad en la capa física de la arquitectura IoT ha llevado a desarrollar el concepto de Smart Object, como lo señala [Beltran, 2018], que puede ser usado para establecer mecanismos de Identificación y Autenticación (IAM). Tal como se realiza en los procesos de software, donde la identificación mediante un código o key es necesario para que un proceso transmita datos a otro, a nivel de los dispositivos de capa física este concepto es también implementado, como en [Montenegro-Marin et al., 2017] donde se hace una evaluación del modelo IoT en base a principios de seguridad.

Dentro de las investigaciones realizadas en el tema de la seguridad en capa física se encuentran el trabajo de [Kitchin & Dodge, 2019] en el que se realiza un estudio amplio sobre las vulnerabilidades y riesgos en una arquitectura de Smart City. Una de las soluciones propuestas es implementar un protocolo de seguridad para la etapa de captura

de datos, como en [Srinivas et al., 2017] donde se plantea un protocolo de autenticación y generación de claves en WSN con biometría. En [Fremantle & Scott, 2017] se realizó un estudio completo sobre seguridad basada en IoT definiendo una matriz de retos de seguridad. Ampliando la investigación en [Khanam et al., 2020] se presenta un estudio completo de los problemas de vulnerabilidad y ataques en toda la Arquitectura IoT, incluyendo contramedidas propuestas.

El concepto de autenticación y autorización ha estado presente en la implementación de modelos de seguridad para IoT en varias propuestas desarrolladas, por ejemplo, [Wu et al., 2017] desarrolla una propuesta de autenticación utilizando clave temporal. Sin embargo, [Reddy et al., 2018] plantea un modelo alternativo usando criptografía ECC y bajo autenticación de 3 factores. En la capa de borde, [Duan et al., 2019] define un conjunto de características de privacidad (DIKW). Otros autores, como [Mukherjee et al., 2017], plantean nuevos criterios de seguridad y privacidad en Fog Computing. En una importante contribución, [Abbas et al., 2019] proponen un servicio de seguridad fog (FSS) para proporcionar seguridad de extremo a extremo en la capa fog para dispositivos IoT.

Es evidente que el desarrollo de modelos de seguridad necesarios para proteger la arquitectura IoT ha evolucionado a conceptos como Encriptación como servicio (EaaS) como se indica en [Javadpour et al., 2024]. Por otro lado, en Smart Cities destacan trabajos como [Xie & Hwang, 2019]. De estos trabajos se desprende que se evidencia la necesidad de implementar un modelo de seguridad para dispositivos limitados (constraint devices) empleando Edge/Fog Computing..

1.2. Definición del Problema

¿El desarrollo de un módulo de validación criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing permitirá mejorar el nivel de seguridad de los datos durante la adquisición y envío de datos de la capa física IoT?

1.3. Objetivos

1.3.1. Objetivo Principal

Desarrollar un módulo de validación criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing que permita mejorar el nivel de seguridad de los datos durante la adquisición y envío de datos de la capa física IoT

1.3.2. Objetivos Secundarios

- Desarrollar un modelo de criptografía adecuado a dispositivos IoT con recursos limitados
- Construir el modelo de gestión de autenticación/autorización para el uso del módulo de criptografía a usar.
- Determinar las características del módulo Edge Computing donde residirá el módulo de gestión.

1.4. Justificación

1. Permitirá implementar un modelo de seguridad eficiente dirigido a dispositivos constrained que por su naturaleza son inseguros.

2. Permitirá utilizar técnicas de criptografía de bajo costo computacional y de hardware.

3. Implementara un uso eficiente de la capa Edge computing en niveles de seguridad para todo tipo de dispositivo constrained.

4. Este proyecto definirá un modelo IoT que puede implementarse usando Open Hardware y cualquier tipo de comunicación inalámbrica, de acuerdo al alcance presentado en el presente trabajo de investigacion.

1.5. Alcance de la Investigación

Esta investigación, por su naturaleza involucra los 3 componentes TIC, es decir, la captura de datos en tiempo real mediante la capa física, preparación de datos usando la propuesta a presentada basada en hardware, y software y envio de los datos mediante la capa de comunicaciones. Por esa razón, se plantea este trabajo con las siguientes consideraciones:

a. Se considera dentro de la capa de comunicaciones transmisión de corta a larga distancia, no se considera uso de RFID o NFC

b. Se establece únicamente el uso de dispositivos de recursos limitados (resource-constrained)

c. El estudio establece el uso de dispositivos que enviaran datos independiemente que se use o no protocolos IoT.

El alcance de la investigacion esta definido de esta manera para que la mayoría de los casos de implementación de proyectos basados en Internet de las Cosas pueda ser considerados como caso de uso.

1.6. Hipótesis

La implementación de un módulo de validación criptográfica para constraint devices basado en Internet de las Cosas usando Edge Computing permite mejorar el nivel de seguridad de los datos durante la adquisición y envio de datos de la capa física IoT.

1.7. Variables e Indicadores

Las variables y sus respectivos indicadores se pueden observar en la Tabla55

Tabla 1. Variables e indicadores

Variables	Indicadores
a) **Independiente**: Módulo de validación criptográfica para constraint	• Diseño arquitectónico • Modulo criptografico

	devices basado en Internet de las Cosas usando Edge Computing	• Implementación del módulo de Edge Computing
b)	**Dependiente**: Nivel de seguridad de los datos durante la adquisición y envio de datos de la capa física IoT	• Prueba de stress • Test de seguridad • Pruebas de caja negra

1.8. Población y Muestra

- **Población**

  La población esta referida a los dispositivos que podrán ser conectados al sistema que se esta planteando, al ser un numero indefinido, se considera la población infinita.

- **Muestra**

  La muestra de esta investigación es una muestra probabilística, la cual se determinó mediante la siguiente formula, teniendo en cuenta el tamaño de la población, el nivel de confianza y el margen de error.

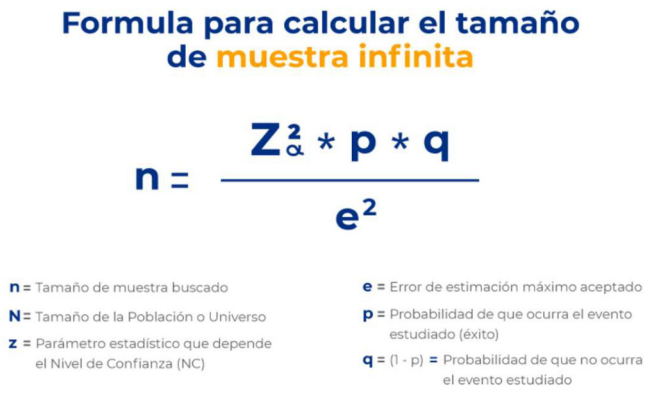


Fig 2 Calculo del tamaño de muestra infinita

Donde:

n: Tamaño de la muestra.

Z: Nivel de confianza (80%), Z = 1.28

E: Error de la muestra (10%), E = 0.1

p: Probabilidad de éxito de cada integrante de la población (0.5).

q: Probabilidad de fracaso de cada integrante de la población (0.5).

Luego, sustituyendo en la fórmula:

$$n = \frac{(1.28)^2(0.5)(0.5)}{(0.05)(0.05)}$$

$$n = \frac{(1.6384)(0.25)}{(0.0025)}$$

$$n = 163{,}84$$

$$n \cong 164$$

Finalmente, el tamaño de la muestra es de lecturas de 164 dispositivos, o 164 lecturas de dispositivos constraint.

Cabe señalar que, para el cálculo del tamaño de la muestra se utilizó un nivel de confianza del 80% asi como una tasa de error de 10%, debido a que los dispositivos constraint generalmente pueden generar valores de tipo outliers.

Capítulo II: Marco Teórico

En este capítulo se expondrán los conceptos necesarios para el entendimiento de los temas que se presentan para la definición de la propuesta de tesis.

2.1 Bases Teóricas

2.1.1 Edge Computing

Edge Computing se refiere a un modelo de computación distribuida en el que el procesamiento y análisis de datos se realiza directamente en los dispositivos IoT, en el extremo de la red, más cerca del punto de generación de los datos, según [Kong et al., 2022], lo que puede acelerar la velocidad de respuesta de los servicios. En este modelo, los dispositivos IoT son responsables de realizar tareas de procesamiento y análisis sin depender de recursos adicionales en la red [Ni et al., 2019].

El objetivo principal del Edge Computing es minimizar la latencia, reducir la carga de la red y mejorar la eficiencia al llevar la computación más cerca de los dispositivos IoT [Abbas et al., 2019]. Edge Computing es ideal para aplicaciones en las que la latencia y la privacidad son críticas [Elmassik, 2023], mejorando eficiencia, agilidad y latencia en IoT [Hamdan et al., 2020].

.Edge Computing es ideal para aplicaciones en las que la latencia, la privacidad de los datos y la capacidad de funcionamiento offline son críticas, como la monitorización en tiempo real, como indica [Elmassik, 2023], la seguridad y el procesamiento de datos en entornos con conectividad limitada.

Así, de acuerdo con [Hamdan et al., 2020], el uso de Edge permite la mejora de varios aspectos importantes en la implementación de IoT, como:

- Eficiencia: porque permite que los dispositivos IoT utilicen eficientemente los recursos informáticos en el borde.

- Cognición: un dispositivo edge es consciente de los requerimientos del cliente.

- Agilidad: es más rápido y barato de utilizar con dispositivos y clientes periféricos debido a que el. procesamiento y almacenamiento de datos se realizan cerca del usuario final.

- Latencia: la computación de borde es compatible con las aplicaciones en las que el tiempo es un factor crítico, y eso permite a las aplicaciones IoT tomar decisiones más rápido y mejor.

2.1.2 Modelos de seguridad en Edge Computing

Estos modelos se centran en abordar los desafíos de seguridad específicos que enfrentan estos dispositivos en entornos de Edge Computing [Kalariya et al., 2022]:

- Modelo de seguridad basado en capas: Este enfoque propone una arquitectura de seguridad en capas, donde cada capa implementa diferentes mecanismos de seguridad. Por ejemplo, se pueden aplicar capas de cifrado, autenticación y control de acceso, detección de anomalías y seguridad en la comunicación para proteger los Constraint devices y sus datos [Alwarafy et al., 2021].
- Modelo de seguridad basado en confianza: Este modelo se basa en el establecimiento de niveles de confianza entre los dispositivos y los sistemas en el entorno de Edge Computing [Sha et al., 2020]. Se utilizan técnicas de autenticación, verificación y certificación para establecer la confianza entre los Constraint devices y los componentes de Edge Computing.

- Modelo de seguridad basado en virtualización: Este enfoque propone el uso de técnicas de virtualización para proporcionar un entorno seguro y aislado para los Constraint devices en Edge Computing. Mediante la virtualización, se pueden crear instancias virtuales que se ejecutan en dispositivos físicos compartidos, pero están aisladas unas de otras, proporcionando una capa adicional de seguridad [Sitton, R.,2020].
- Modelo de seguridad basado en la minimización de recursos: Este modelo se centra en la minimización de la exposición de los Constraint devices al reducir la superficie de ataque y limitar las funcionalidades y servicios que se ejecutan en los dispositivos. Se implementan mecanismos para deshabilitar servicios innecesarios, eliminar funciones no utilizadas y aplicar configuraciones de seguridad estrictas para reducir las vulnerabilidades [Sha, K. et al.,2020].

2.1.3 Dispositivos de recursos limitados (Constraint Devices)

Los dispositivos limitados son dispositivos que contienen sensores/actuadores que se utilizan en aplicaciones específicas. Se caracterizan por: recursos limitados de procesamiento [Pathmabandu et al., 2023], restricciones de energía [Sabovic et al., 2023], limitaciones de almacenamiento [Nasir et al., 2022], conectividad limitada [Chowdhury & Emeroylariffion, 2022], y tamaño compacto [Pathmabandu et al., 2023]. La norma RFC 7228 clasifica estos dispositivos en Clases 0, 1 y 2 [El-Hajj, 2020].

En la siguiente figura se presenta un diagrama de la estructura de un constraint device

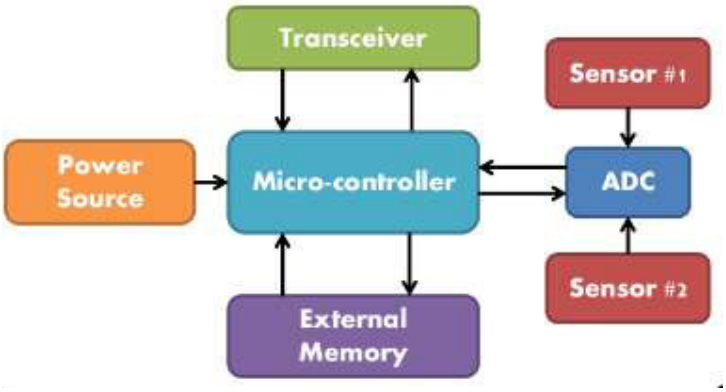


Fig 3: Estructura genérica de un Constraint Device (elab. propia)

A grandes rasgos, los siguientes criterios son comúnmente considerados al clasificar un dispositivo como constraint device:

a. Recursos limitados de procesamiento: Los constraint devices suelen tener capacidades de procesamiento limitadas en comparación con dispositivos más potentes. Pueden tener una menor capacidad de cálculo, menor velocidad de procesador y menos memoria disponible [Pathmabandu, C., et al.,2023].

b. Restricciones de energía: Los constraint devices a menudo están alimentados por baterías u otras fuentes de energía limitadas. Por lo tanto, deben ser eficientes en términos de consumo energético para prolongar la duración de la batería o utilizar técnicas de ahorro de energía [Sabovic, A. et al.,2023].

c. Limitaciones de almacenamiento: Estos dispositivos pueden tener una capacidad de almacenamiento limitada para almacenar datos y aplicaciones. Por lo tanto, es posible

que necesiten utilizar estrategias de almacenamiento optimizado o depender de servicios de almacenamiento en la nube [Nasir, M., et al.,2022].

d. Conectividad limitada: Los constraint devices pueden tener restricciones en términos de conectividad de red. Pueden tener una menor capacidad de transmisión de datos, menor ancho de banda o estar sujetos a restricciones de red, como una conexión intermitente o limitada [Chowdhury, R. R., & Emeroylariffion, P. E.,2022].

e. Tamaño y forma compactos: Estos dispositivos suelen tener un tamaño y una forma compactos para facilitar su integración en diferentes entornos y aplicaciones. Pueden ser pequeños y ligeros para ser fácilmente transportables o tener un diseño especializado para ajustarse a restricciones físicas [Pathmabandu, C., et al.,2023].

La norma RFC 7228[1] clasifica los dispositivos constraint en tres categorías basadas en la memoria y el tamaño de los datos que puede almacenar, tal y como se muestra en la Tabla 2.

Tabla 2: Clases de Constraint devices, según [El-Hajj, M.,2020]

Clase	Flash	RAM	Uso
0	<100KB	<1KB	Uso de gateway para la comunicación
1	~100KB	~10KB	Interactúa con otros dispositivos sin necesidad de gateway
2	~250KB	~50KB	Compatible con IPV4 e IPv6

2.1.4 Diferencias entre Fog Computing y Edge Computing

Fog Computing se sitúa entre los dispositivos IoT y la nube [Aazam et al., 2018]. Es beneficioso en entornos como eHealth [Kraemer et al., 2017] y se considera una forma de implementación de Edge Computing [Ai et al., 2018].:

2.1.4.1 Fog Computing:

Fog Computing se refiere a un modelo de computación distribuida que se sitúa entre los dispositivos IoT y la nube. También se le conoce como "computación en la niebla" o "computación nebulosa" [35].

En Fog Computing, se colocan recursos informáticos y de almacenamiento en puntos intermedios de la red, como routers, gateways y servidores en el borde de la red.

El objetivo principal del Fog Computing es reducir la latencia y mejorar la eficiencia al procesar datos y ejecutar aplicaciones más cerca de los dispositivos IoT, lo que permite una respuesta más rápida y un menor tráfico de datos hacia la nube, una de sus aplicaciones mas conocidas en eHealth [36]..

Fog Computing permite realizar tareas de procesamiento, análisis y almacenamiento en el nivel intermedio de la red, más cerca de los dispositivos, lo que resulta beneficioso

[1] https://www.rfc-editor.org/rfc/rfc7228

para aplicaciones que requieren baja latencia, procesamiento en tiempo real y una mayor autonomía en la toma de decisiones.

2.1.4.2 Diferencias clave entre Edge Computing y Fog Computing:

Fog Computing se sitúa en puntos intermedios de la red, mientras que Edge Computing se realiza directamente en los dispositivos IoT.

Fog Computing se basa en recursos adicionales en la red, como gateways y servidores, mientras que Edge Computing utiliza los recursos de los dispositivos IoT.

Fog Computing se enfoca en tareas de procesamiento y análisis intermedias, mientras que Edge Computing se centra en llevar la computación al extremo de la red, cerca de los dispositivos IoT. Sin embargo, dentro de los conceptos actuales [37], Fog Computing es una forma de implementación Edge Computing.

2.1.5 Modelos de criptografía para constraint devices

Desde el inicio del desarrollo de aplicaciones basadas en IoT, el tema de la seguridad ha estado presente en las diferentes formas de implementación. Como se muestra en [Chowdhury & Abas, 2022], las arquitecturas IoT requieren la evaluación de seguridad.

Sin embargo, en este trabajo, se ha determinado varios modelos de criptografía para dispositivos IoT con limitaciones, cada uno con sus propias ventajas y desventajas. Algunos de los modelos más comunes son:

.

2.1.5.1 Criptografía asimétrica:

Incluye algoritmos como RSA y Elliptic Curve Cryptography (ECC) [Taleb & Toufik, 2024]. ECC es más adecuado para dispositivos IoT con recursos limitados. Las técnicas más conocidas en el uso de capa física IoT son:

a. RSA: Este es un algoritmo de criptografía asimétrica ampliamente utilizado que es considerado muy seguro. Sin embargo, es computacionalmente intensivo, lo que lo hace inadecuado para dispositivos con recursos limitados.
b. Elliptic curve cryptography (ECC): ECC es una alternativa más ligera a RSA que ofrece un nivel de seguridad comparable [Taleb, F., & Toufik, L,2024]. Es más adecuado para dispositivos IoT con recursos limitados.

2.1.5.2 Criptografía simétrica:

La criptografía simétrica es una tecnología de cifrado que cifra y descifra datos enviados a través de una red utilizando la misma clave [Ekwueme et al, 2024]. Es segura y rápida, sin embargo, el problema es compartir la clave, ya que, si es capturada por un atacante, los datos cifrados quedan comprometidos

a. Advanced Encryption Standard (AES): AES es un algoritmo de criptografía simétrica ampliamente utilizado que es considerado muy seguro. Es más eficiente computacionalmente que RSA, lo que lo hace más adecuado para dispositivos IoT.
b. Lightweight Encryption Algorithm (LEA): LEA es un algoritmo de criptografía simétrica ligero que fue diseñado específicamente para dispositivos IoT. Es aún más eficiente computacionalmente que AES, pero puede no ser tan seguro.

2.1.5.3 Hashing:

El algoritmo SHA (Secure Hash Algorithm) fue desarrollado por el Instituto Nacional de Normas y Tecnología (NIST), que es una entidad especializada en normalización y tecnología [Ghazi et al., 2023]. Se compone de tres conjuntos únicos de funciones hash criptográficas, cada una de las cuales se denomina su propio número: SHA-1, SHA-2 y SHA-3, respectivamente.

a. SHA-256: SHA-256 es un algoritmo de hashing ampliamente utilizado que se utiliza para garantizar la integridad de los datos. Es computacionalmente eficiente y adecuado para dispositivos IoT.

b. BLAKE2: BLAKE2 es un algoritmo de hashing más reciente que ofrece un nivel de seguridad comparable a SHA-256. Es más eficiente computacionalmente que SHA-256, lo que lo hace más adecuado para dispositivos IoT.

2.1.6 Consideraciones de seguridad

Un tema importante es la identificación de las amenazas que pueden producirse en el uso de dispositivos de tipo constraint devices, por esa razón en [Zhou et al.,2019] se presenta una tabla identificando los principales retos que pueden aparecer en la implementación de aplicaciones basadas en IoT.

Tabla 3: Características de IoT, retos y oportunidades, de [Zhou et al.,2019]

AMENAZAS, RETOS Y OPORTUNIDADES DE CADA UNA DE LAS CARACTERÍSTICAS DE IoT

Característica	**Amenaza**	**Desafío**	**Oportunidad**
Interdependencia	Eludir las defensas estáticas, Sobreprivilegios	Control de acceso y gestión de privilegios	Context-based permission
Diversidad	Protocolos inseguros	Fragmentacion	Plataforma de simulación de análisis dinámico, IDS
Restringido	Sistemas inseguros	Defensas y protocolos ligeros	Combinación de características biológicas y físicas
Myriad	IoT botnet, DDoS	Detección y prevención de intrusiones	IDS
Sin vigilancia	Ataque remoto	Verificacion remota	Atestación remota, ejecución ligera de confianza
Intimidad	Filtración de datos personales	Proteccion de la Privacidad	Cifrado homomórfico, Protocolos anónimos
Movil	Propagacion de Malware	Identificación y confianza entre dominios	Configuracion dinamica
Ubicuidad	Configuracion Insegura	\	Conciencia de seguridad

2.2 Glosario de Términos

2.2.1 Constrained Device: Es una tecnología que se basa en la observación del estado de un usuario a través de la extracción y decodificación de información, procedente del registro de su actividad cerebral, con el objetivo de emplearla para generar alguna salida funcional.

2.2.2 Gateway IoT: Dispositivo que convierte un protocolo o formato en otro. Una pasarela de red convierte paquetes de un protocolo a otro. Una pasarela de aplicación convierte comandos y/o datos de un formato a otro. Una pasarela de correo electrónico convierte mensajes de un formato de correo a otro. Véase pasarela IP, servidor proxy, pasarela por defecto, pasarela multimedia, pasarela NAS, pasarela de mensajería, pasarela de señalización, pasarela WAP, softswitch, EGP, BGP e IGP.

2.2.3 IoT Hub: Es el controlador central que conecta diversos dispositivos inteligentes en el hogar, como cerraduras de puertas, cámaras, luces y sensores conectados

2.2.4 OWASP: (Open Web Application Security Project) Organización fundada por Mark Curphey en 2001, para contribuir a la seguridad del software de código abierto. Con comunidades de miembros en todo el mundo, los proyectos OWASP se ocupan de lenguajes de programación, funciones y aplicaciones específicas, así como de normas generales para desarrollar aplicaciones protegidas. De acuerdo a [Kumar & Rani, 2022]. El primer OWASP (2003) publicó las 10 vulnerabilidades de seguridad de aplicaciones web más críticas que deben tenerse en cuenta a la hora de la creación de aplicaciones web seguras, con una actualización de las últimas vulnerabilidades en 2004.

Capítulo III: Estado del Arte

En este capítulo se realizó una revisión bibliográfica, con el objetivo de identificar investigaciones, artículos científicos o publicaciones en donde los autores hayan empleado métodos o herramientas alternativos a la solución propuesta en esta investigación para el problema de implementar un modelo criptografico para dispositivos de tipo constraint devices.

3.1 Revisión Bibliográfica de Literatura

La revisión de la literatura requerida para la presente tesis deberá encontrar artículos en journals, tesis, o proceddings que servirán para resolver las preguntas de investigación:

Q1: Cuales son las técnicas de criptografía que pueden ser usadas para la seguridad de datos en dispositivos de tipo constraint en IoT

Q2: Cuales técnicas de criptografía aplicadas en IoT pueden ser implementadas en Edge computing.

3.1.1 Revisión Sistemática en SCOPUS

Sea crea la cadena de búsqueda requerida, cuyo valor es el siguiente;

TITLE-ABS-
KEY (cryptography AND for AND constraint AND devices AND using AND edge AND computi ng) AND PUBYEAR > 2018 AND PUBYEAR < 2025 AND (LIMIT-
TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI") OR LIMIT-
TO (SUBJAREA , "MATH"))

Tabla 4: criterios de inclusión y exclusion

Criterios de inclusion	**Criterios de exclusion**
Los papers deben ser del periodo 2019-2024	Solo eran artículos en Ingles
Se consideran artículos de procedding o journal	Solo serán validos validos artículos que consideren criptografía en dispositivos IoT
Se consideran temas en Computer Science, Ingenieria y Matematicas	

Después de la búsqueda definida se encontraron 43 artículos.

Posteriormente, después de la revisión de título y abstract, el resultado será:

Papers candidatos	43
Papers aprobados	26

Posteriormente, se tuvo acceso a la Biblioteca Virtual CONCYTEC y haciendo uso del acceso como investigador, se buscó papers usando el tema y criterios de inclusión y exclusiones similares, obteniéndose el resultado siguiente:

Papers candidatos	22
Papers aprobados	22

A continuación, se resolverán las preguntas de investigación, con el fin de determinar el modelo de desarrollo para el problema planteado

Q1. Cuáles son las técnicas de criptografía que pueden ser usadas para la seguridad de datos en dispositivos de tipo constraint en IoT

Las técnicas más usadas identificadas en la literatura son ECC [Taleb & Toufik, 2024; Wu et al., 2017; Fadia & Toufik, 2024; Simon Francia et al., 2022; Lara-Nino et al., 2018], RSA [Bongale et al., 2014], AES [Shah & Venkatesan, 2018; Mosanu et al., 2019], LEA [Mohamad Al-Aboosi et al., 2021], ABE/PRNG [Fischer et al., 2019; Gupta & Kumar, 2020; Smithamol & Rajeswari, 2017] y Hashing [Kaur et al., 2023; Pinto & Costa, 2016].

De acuerdo a los resultados obtenidos, se presenta la siguiente tabla 5

Tabla 5: Resultados de la pregunta Q1:

Tecnica de Criptografia	Metodo	Papers
Criptografía Asimétrica	ECC	[Taleb & Toufik, 2024; Wu et al., 2017; Fadia & Toufik, 2024; Simon Francia et al., 2022; Lara-Nino et al., 2018],
Criptografía Asimétrica	RSA	[Bongale et al., 2014]
Criptografía Simétrica	AES	[Shah & Venkatesan, 2018; Mosanu et al., 2019]
Criptografía Simétrica	LEA	[Mohamad Al-Aboosi et al., 2021]
Generador de Criptografía	ABE, PRNG, CP-ABE	[[Fischer et al., 2019; Gupta & Kumar, 2020; Smithamol & Rajeswari, 2017]
Hashing		[[Fischer et al., 2019; Gupta & Kumar, 2020; Smithamol & Rajeswari, 2017].

Q2: Cuales técnicas de criptografía aplicadas en IoT pueden ser implementadas en Edge computing.

En base a los resultados de la revisión bibliográfica, se obtienen los siguientes resultados:

Destacan esquemas como ZKP Hash-Chain [Pardeshi et al., 2022], autenticación de n-pasos [Sha et al., 2020], autenticación multifactor [Alwarafy et al., 2020], Fog Security Service (FSS) [Abbas et al., 2019], criptografía ligera [Cecílio et al., 2024], protocolos ECC sobre MEC [Kaur et al., 2019], hardware ECC en Artix-7 [Vishnubhai, 2023] y esquemas ECC resilientes a CPA/CCA [AlMajed & AlMogren, 2020].

Tabla 6: Papers de revisión para la pregunta Q2

Autores	Titulo	Tecnicas

[56]	Hash-Chain Fog/Edge: A Mode-Based Hash-Chain for Secured Mutual Authentication Protocol Using Zero-Knowledge Proofs in Fog/Edge	zero-knowledge proof (ZKP)
[57]	A survey of edge computing-based designs for IoT security	Tecnicas de autenticacion de n pasos
[58]	A Survey on Security and Privacy Issues in Edge-Computing-Assisted Internet of Things	Autenticación multifactor
[17]	A Mechanism for Securing IoT-enabled Applications at the Fog Layer	RSA
[59]	Software-based Security Framework for Edge and Mobile IoT	Criptografia Ligera
[60]	A Lightweight and Privacy-Preserving Authentication Protocol for Mobile Edge Computing	ECC
[61]	Design and Implementation of Efficient Elliptic Curve Cryptography on Reconfigurable PlatformsDesign and Implementation of Efficient Elliptic Curve Cryptography on Reconfigurable Platforms	ECC
[62]	A Secure and Efficient ECC-Based Scheme for Edge Computing and Internet of Things	ECC

En la revisión se observa el papel clave que se le da a la capa Edge como componente importante para usar como mecanismo de seguridad en la arquitectura IoT. En [Pardeshi, M.S. et al ,2022] se estudia el concepto de autenticación dinámica, tomando como base la interacción Edge/fog para el envío de datos de la capa física, este tipo de autenticación se testea usando pruebas de conocimiento cero (ZKP): Este tipo de prueba requiere que todos los dispositivos conectados a los entornos fog necesitan resolver el modelo reto-respuesta para la autenticación efectiva, que tiene tres propiedades básicas que deben cumplirse. En [Sha, K. et al, 2020] se presenta una revisión de implementaciones de seguridad IoT tomando como base el uso de Edge Computing. Estas soluciones cubren IoT, como arquitectura de seguridad integral, firewalls, sistemas de detección de intrusiones, mecanismos de autenticación y autorización, así como privacidad y seguridad, indicándose que, en los mecanismos de autenticación y autorización, así como diseños que preservan la privacidad, el uso de autenticación de n pasos como medio de preservación de datos. En el caso de [Alwarafy A. et al, 2020] se presenta un estudio exhaustivo sobre privacidad en IoT basada en el uso de Edge Computing, para ello, se presenta un análisis de la seguridad y privacidad haciendo una clasificación y tipos de posibles ataques en IoT, asi como que contramedidas se pueden aplicar, haciendo énfasis en la autenticación multifactor, por otro lado en [Abbas et al, 2019] presenta un modelo de seguridad para los dispositivos constraint devices, que consiste en descargar las operaciones adicionales relacionadas con la seguridad a un paradigma con más recursos, como una capa fog, para ello desarrollan el concepto Fog Security Service (FSS), que proporciona autenticación basada en identidad, integridad de datos y no repudio. de los datos y el no repudio de los nodos conectados a través de un generador de claves privadas (PKG) en la capa fog. FSS mejora la seguridad de extremo a extremo entre la capa IoT y la capa fog. En [Cecílio J. et al, 2024] se muestra un trabajo en ejecución, el cual muestra el diseño de una comunicación segura entre servidores remotos y dispositivos IoT para gestionar eficientemente la seguridad y el costo energético.

Para ello se propone un marco arquitectónico que integre los servidores Edge y un servidor de nombres central, junto con el despliegue de criptografía ligera, lo que permite mejorar la seguridad respetando las limitaciones de recursos de los dispositivos IoT. recursos limitados de los dispositivos IoT, además en [Kaur K. et al, 2019] se propone un protocolo de autenticación mutua ligero y que preserva la privacidad para entornos Mobile Edge Computing. dicho protocolo se basa en ECC, funciones hash unidireccionales y operaciones de concatenación para su funcionalidad segura y ligera. Esta combinación de técnicas de encriptación permite una transferencia segura de datos a través de MEC, indispensable en la comunicación entre dispositivos móviles. Por otro lado, en [Vishnubhai PK., 2023] el tesista de Doctorado presenta una propuesta de construcción de una arquitectura ECC usada en Edge Computing, destinada a manejar seguridad para dispositivos de tipo constraint, la propuesta define un diseño usando Artix 7 mejoró la velocidad de criptográfica entre un 10% y un 20% de media y optimizó el área entre un 5% a 10% en comparación con modelos existentes. Finalmente, en [AlMajed, H., & AlMogren, A., 2020] se propone un modelo de criptografía de curva elíptica (ECC) que ofrece mejorar la codificación del mensaje de forma eficaz y asignarlo ya codificado a una curva elíptica, lo que es una novedad ya que normalmente no se da importancia a la fase de codificación. esta etapa fue diseñada para resistir ataques CPA, CCA y de maleabilidad. También llevó a cabo un análisis de seguridad para presentar una prueba de la resistencia del esquema propuesto frente a ataques de cifrado específicos, por otro lado, el estudio llevó a cabo una evaluación del rendimiento para comparar el impacto de la mejora de la seguridad del esquema propuesto en la sobrecarga de tiempo, el uso de espacio y el consumo de energía con otros esquemas, lo que demostraría su amplia mejora para entornos IoT donde criterios como uso limitado de energía o baja potencia es lo que interesa al tema propuesto

3.2 Determinación de la Solución

La revisión sistemática, muestra que en la combinación de respuestas de las 2 preguntas de investigación arroja que la solución más utilizada seria **Criptografía de Curva Elíptica (ECC) sobre nodo Edge Computing.** Los beneficios que se obtiene serian los siguientes:

1. Seguridad con claves más pequeñas

ECC ofrece un nivel de seguridad equivalente a RSA/DH pero con claves mucho más pequeñas. Por ejemplo: Una clave ECC de 256 bits ofrece una seguridad comparable a una clave RSA de 3072 bits [Ferraiolo, H., & Regenscheid, A., 2024].

Esto reduce el uso de ancho de banda, el almacenamiento necesario y mejora los tiempos de procesamiento.

2. Bajo consumo de energía

Dado que las operaciones criptográficas con ECC requieren menos cálculos, se consume menos batería, lo que es ideal para sensores alimentados por baterías o energía solar [Banerjee, U. et al., 2019]

3. Rendimiento mejorado

ECC permite tiempos más rápidos de firma, verificación y cifrado/descifrado, lo que es crucial para aplicaciones en tiempo real y nodos edge que deben procesar y reenviar datos rápidamente [AlMajed, H., & AlMogren, A., 2020].

4. Ideal para dispositivos con recursos limitados

ECC es ideal para microcontroladores y sensores con baja capacidad de CPU y poca RAM/ROM [Islam, T. et al, 2022], como los basados en ESP32, ARM Cortex-M, etc.

5. Menor uso de ancho de banda

Dado que ECC requiere menos datos para transmitir claves públicas, reduce el tráfico en redes inalámbricas de baja potencia como LoRaWAN, Zigbee, o NB-IoT [Mishra, A. R., & Singh, M., 2012].

6. Escalabilidad y compatibilidad con estándares modernos

ECC está soportado por TLS 1.3, MQTT-SN, CoAP, y otros protocolos de IoT.

Permite integrar fácilmente los nodos edge a arquitecturas más grandes y seguras de IoT [Seoane, V. et al, 2021].

7. Adecuado para entornos con baja latencia

En edge computing, se busca tomar decisiones rápidas sin depender del servidor central. ECC permite cifrado/descifrado local sin retrasos significativos [Di Matteo, S. et al, 2021]

Capítulo IV: Aporte Teórico

En este capítulo se determinará la implementación de la propuesta técnica basada en la elección del algoritmo de criptografía establecido en el estado del arte.

4.1 Esquema General de la Solución

La arquitectura propuesta consta de tres capas principales: Device Layer, Edge Layer y Cloud Layer.

El cifrado asimétrico asume que al cifrar datos con una clave pública, se podrán descifrar posteriormente con la clave privada correspondiente [Shah et al., 2023]. En esta propuesta se implementa un esquema híbrido ECDH + AES-GCM.

4.1.1 Modelo de implementación Hardware

La figura siguiente muestra la arquitectura de tres capas ya adaptada a Google Cloud. Las capas Device y Edge conservan su diseño original; la capa Cloud se reconstruye íntegramente sobre servicios administrados de Google Cloud:

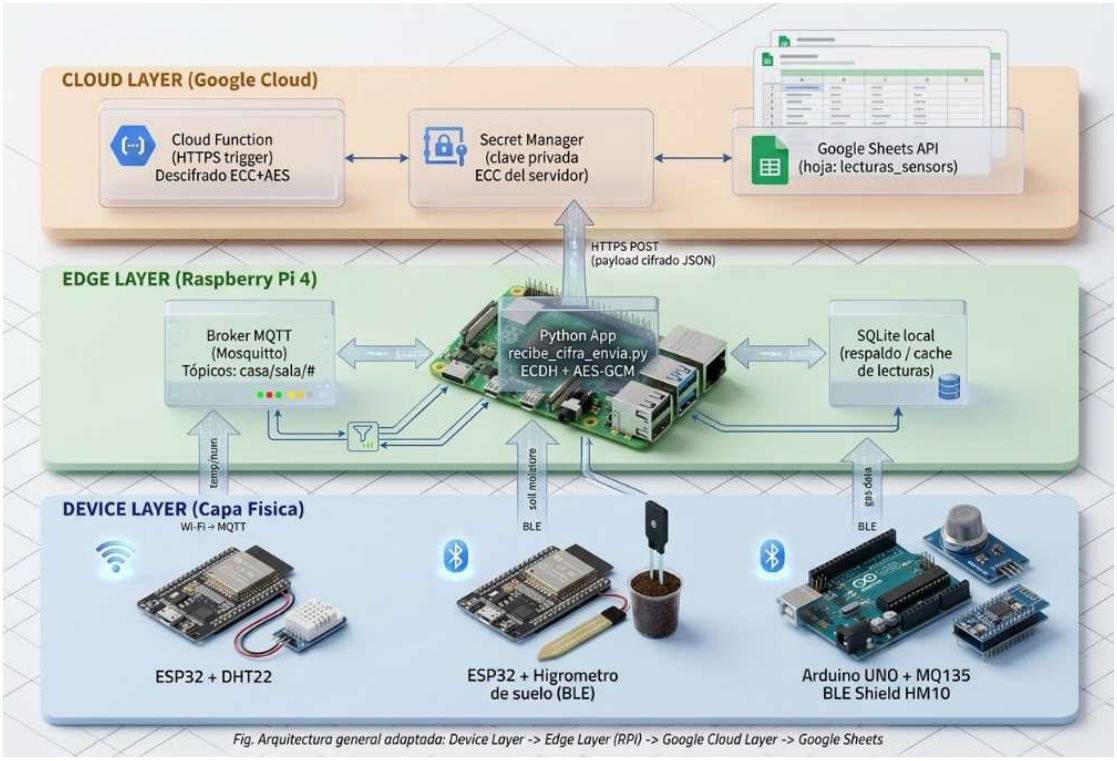


Fig. 4: Implementación física del proyecto de investigación

El modelo constituido por los siguientes componentes:

Tabla 7: Componentes por capa

Capa	Componente	Función

Device	ESP32 + DHT22 (p1)	Temperatura y humedad relativa, envío por Wi-Fi/MQTT
Device	ESP32 + higrómetro de suelo (p2)	Humedad de suelo, envío por BLE (GATT notify)
Device	Arduino UNO + MQ135 + HM10 (p3)	Calidad de aire, envío por BLE (UART, shield HM10)
Edge	Broker MQTT (Mosquitto)	Recepción de p1; unifica también p2/p3 vía el puente BLE
Edge	edge_cifra_envia.py	Cifrado híbrido ECC (ECDH+AES-GCM) y envío HTTPS a Google Cloud
Edge	SQLite local	Caché/respaldo de lecturas si falla la conexión a Internet
Cloud	Cloud Function (HTTP)	Descifrado de la trama y escritura en Google Sheets
Cloud	Secret Manager	Custodia de la clave privada ECC del servidor
Cloud	Google Sheets API	Repositorio final de datos (hoja "lecturas_sensores")

4.1.2 Device Layer

Esta capa se ha organizado para definir diferentes maneras de conectar los dispositivos de capa física con la capa Edge, De acuerdo a ello, tener tres dispositivos constraint:

i. Sensor 1 (p1):

- sensor de temperatura y humedad relativa DHT22
- Controlador: esp32 5V
- Comunicación: Bluetooth BLE incorporado

ii. Sensor 2 (p2):

- sensor de humedad de suelo (soil moisture)
- Controlador: Arduino Pro Mini 3.3v
- Comunicación: Bluetooth BLE (Shield)

iii. Sensor 3 (p3);

- sensor de calidad de aire MQ135

- Controlador: Arduino UNO 5v
- Comunicación: Bluetooth Shield HM10 BLE

4.1.3 Edge Layer

La capa Edge, que es la principal contribución de este trabajo de investigación tendrá las siguientes características:

Componente: Raspberry PI 4

- Sistema Operativo: Raspberry OS
- Comunicaciones: Bluetooth 5 (conexión con capa física), Wifi (con capa cloud)
- Almacenamiento: 32 GB RAM, usados para el modulo de criptografia basado en Python y almacenamiento de datos.
- CPU Quad-core ARM® Cortex®-A57 MPCore processor
- Memory 4 GB 64-bit LPDDR4 1600MHz - 25.6 GB/s
- Storage 16 GB eMMC 5.1 Flash
- Video Encode 250 MP/sec

4.1.4 Cloud Layer

Esta capa recibirá los datos encriptados de la capa Edge y se desencriptará para recuperar la información transmitida

- Nube: Google Cloud
- Protocolo MQTT
- Base de Datos: SQLite
- Framework: Django

Nota: En la nube se recuperará la información original enviada, aplicando la desencriptación propuesta en este trabajo de investigación

4.1.5 Modelo de criptografía

En este punto se presenta la implementación del cifrado/descifrado de clave pública basado en la curva elíptica (esquema de cifrado asimétrico basado en ECC). En esta propuesta se va a construir el modelo de criptografía considerando el diseño de un esquema de cifrado híbrido, en el que intervienen la criptografía ECC, el intercambio de claves ECDH y un algoritmo de cifrado simétrico.

Si se considera el uso de un par de claves ECC privada-pública, lo que se desea es cifrar y descifrar datos utilizando estas claves. Por definición, el cifrado asimétrico funciona de la siguiente manera: si ciframos datos con una clave pública, podremos descifrar el texto cifrado posteriormente con la clave privada correspondiente [Shah, H. et al, 2023].

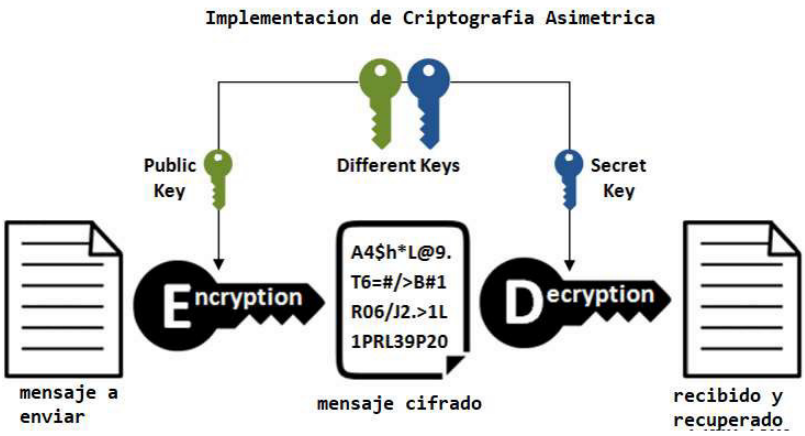


Fig. 5: Criptografía Asimétrica (imagen propia)

El proceso anterior puede aplicarse directamente al criptosistema RSA, pero no al modelo ECC. La criptografía de curva elíptica (ECC) no proporciona directamente un método de cifrado, por esa razón, se diseña un esquema de cifrado híbrido utilizando el esquema de intercambio de claves ECDH (Elliptic Curve Diffie-Hellman) para derivar una clave secreta compartida para el cifrado y descifrado simétrico de datos.

Así es como funcionan la mayoría de los esquemas de cifrado híbrido (el proceso de cifrado)



Fig 6: Combinación de ECC y RSA usando ECDH (imagen propia)

4.2 Implementación de Software

4.2.1 Modelo de Datos

Para el almacenamiento de los datos se planteó el siguiente diseño:

a. Capa Edge

   Se considera el uso de MariaDB/MySql para el almacenamiento de los datos. Su uso en Jetson Nano está bien documentado.

b. Capa Cloud.

En la capa desarrollada usando AWS, el repositorio de datos será de tipo NoSQL, el cual alimentará con datos una página web mostrando los valores, usando Websockets.

4.2.2 Validación de resultados

Una vez desarrollado el prototipo de la propuesta presentada, se efectúa el recojo de resultados a partir de la medición de datos usando 3 sensores de tipo constraint:

a. Sensor de calidad de aire (MQ135)
b. Sensor de temperatura DHT22
c. Sensor de humedad de suelo (Soil moisture)

Sensores usados regularmente en prototipos de prueba IoT y en soluciones de tipo ambientales.

Para la validación de resultados, es decir, la prueba del módulo de criptografía basado en Edge Computing, se utilizaran los conceptos presentados en [p32] considerando los factores de amenaza como:

1. **Procedencia e integridad de los datos**: Responde a las preguntas ¿Cómo se puede confiar en los datos procedentes del sensor? y ¿Cómo se puede garantizar que la integridad de los datos no se ha visto comprometida la integridad de los datos?
2. **Gestión de identidades**: Responde a la pregunta ¿Está autorizado un sensor a enviar datos al nodo?
3. **Gestión de la confianza**: La cuestión aquí es: ¿Se puede confiar en el dispositivo que transmite los datos?
4. **Privacidad**: Las tecnologías de protección de la privacidad pueden proteger los datos confidenciales de los usuarios y, al mismo tiempo, preservar la funcionalidad de las aplicaciones de nivel superior.

4.3 Implementación

4.3.1 Capa Física

4.3.1.1 Sensor1 (p1) conectado a RPI usando wifi y MQTT

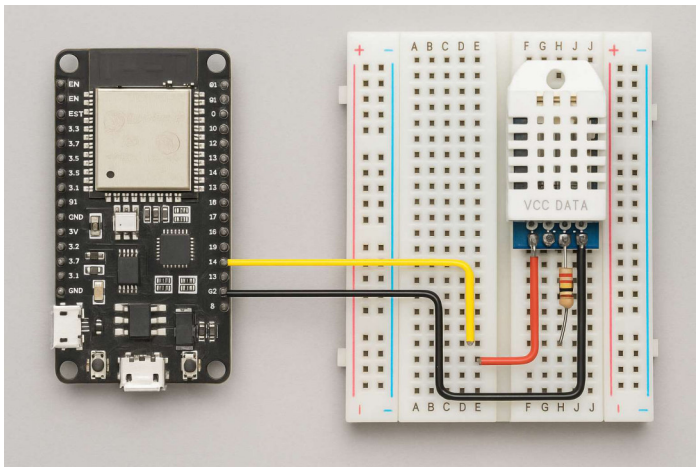
Este circuito consta de una tarjeta esp32 que leerá un sensor DHT22 o sensor DHT11, sensor de temperatura y humedad relativa, trabajando a 3.3v

Tabla 8: Conexiones entre esp32 y DHT22

Conexiones del sensor DHT22/DHT11 (3 pines) al ESP32:

DHT22 Pin	Descripción	Conexión ESP32
1 (VCC)	Alimentación	3.3V
2 (DATA)	Señal de datos	GPIO 4

3 (GND)	Tierra	GND



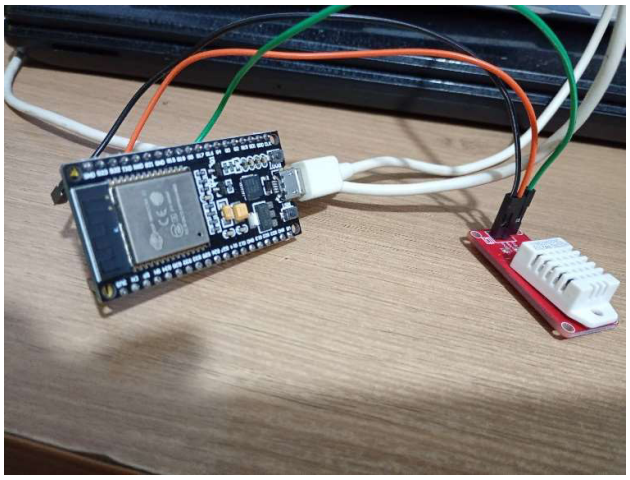


Fig. 7 Circuito implementado sensor1

Para el código correspondiente, debe instalarse las librerías DHT y PubSubClient en el Arduino IDE

El código que se ejecuta para este sensor, es el siguiente:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// --- Configuración de Red ---
const char* ssid = "NOMBRE RED";
const char* password = "MI CLAVE";
const char* mqtt_server = "192.168.0.19"; // Ejemplo: 192.168.1.50

#define DHTPIN 4
```

```
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  delay(10);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
}

void reconnect() {
  while (!client.connected()) {
    if (client.connect("ESP32_DHT11")) {
      Serial.println("Conectado al Broker");
    } else {
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  dht.begin();
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

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

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (!isnan(h) && !isnan(t)) {
```

```
    // Convertir floats a strings para enviar por MQTT
    client.publish("casa/sala/temperatura", String(t).c_str());
    client.publish("casa/sala/humedad", String(h).c_str());

    Serial.print("Enviado -> T: "); Serial.print(t);
    Serial.print(" H: "); Serial.println(h);
  }
  delay(5000); // Envío cada 5 segundos
}
```

4.3.1.1.2 Servidor MQTT

El servidor MQTT que recogerá los datos de los diferentes sensores será un Raspberry PI 4 que funcionará como un nodo Edge IoT y que tendrá funciones de controlador.

Instalación y configuración

Primero, se instalara el Broker MQTT (Mosquitto) y un cliente de Python para procesar los datos. Para ello, se abre el terminal de Raspberry Pi:

Bash

```
# Actualizar e instalar Mosquitto
sudo apt update
sudo apt install -y mosquitto mosquitto-clients

# Habilitar para que inicie siempre con la Pi
sudo systemctl enable mosquitto

# Instalar librería de Python para el receptor
pip install paho-mqtt
```

Codigo Python de recepción de datos del sensor p1

Este archivo se llamara escucha_dht.py y se guardara en el Raspberry Pi y se ejecutara mediante con python3 escucha_dht.py.

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print("Conectado al Broker con código: " + str(rc))
    # Nos suscribimos a todos los sensores en el tópico 'casa'
    client.subscribe("casa/sala/#")

def on_message(client, userdata, msg):
```

```
    print(f"Dato recibido en [{msg.topic}]: {msg.payload.decode()}
unidades")


client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# 'localhost' funciona si el script corre en la misma Pi que el Broker
client.connect("localhost", 1883, 60)

print("--- Iniciando Receptor de Datos ---")
client.loop_forever()
```

Resultados

Al levantarse el servidor Mosquitto mediante el código Python anteriormente presentado, estará en posición de escucha esperando los datos sensor p1.

A continuación se ejecuta el código del modulo esp32 y comenzara la transmision de datos dal sensor p1 a través de wifi usando la dirección IP del módulo RP1

Resultados desde cliente ESP32

```
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 58.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
Enviado -> T: 30.00 H: 57.00
```

4.3.1.1.3 Recepcion desde RPI (usando MQTT)

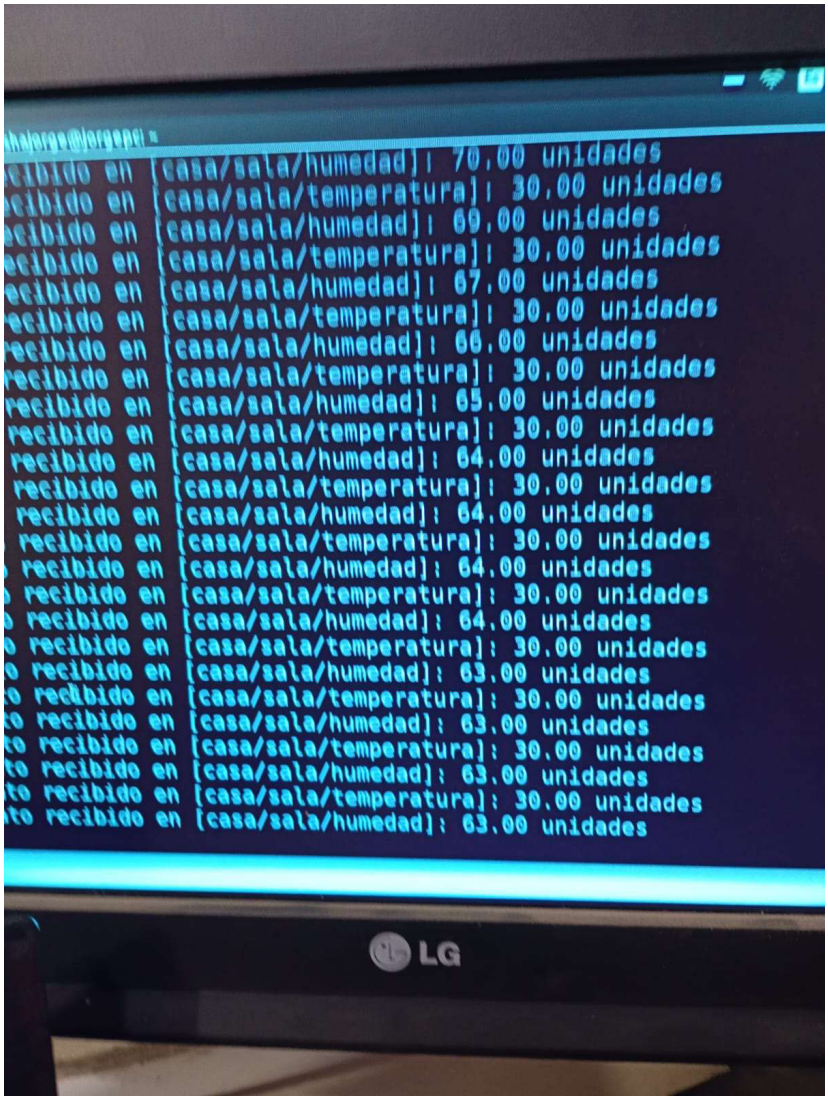


Fig. 8 Recepcion de datos enviados por el bróker MQTT

Se observa que se crea 2 tópicos manejados por el bróker MQTT (sala/temperatura y sala/humedad) que guardaran los datos del sensor p1.

Guardar datos en Base de datos

La ventaja de guardar los datos en una base de datos permitirá crear gráficas históricas, analizar tendencias y aplicar técnicas estadísticas, por ello, se usara **SQLite**, debido a que no necesita configuración compleja y es perfecta para el uso de Raspberry Pi.

Preparar la Base de Datos en el nodo Raspberry Pi

Primero, se crea la base de datos y la tabla ejecutando este pequeño script de Python, la cual se ejecutará una sola vez:

crear_bd.py

```
import sqlite3
```

```
# Conectar (creará el archivo si no existe)
conn = sqlite3.connect('sensores.db')
cursor = conn.cursor()

# Crear tabla para los datos
cursor.execute('''
    CREATE TABLE IF NOT EXISTS lecturas (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        tipo TEXT,
        valor REAL
    )
''')

conn.commit()
conn.close()
print("Base de datos lista.")
```

al ejecutarse se utilizará el siguiente código:

```
python3 crear_bd.py
```

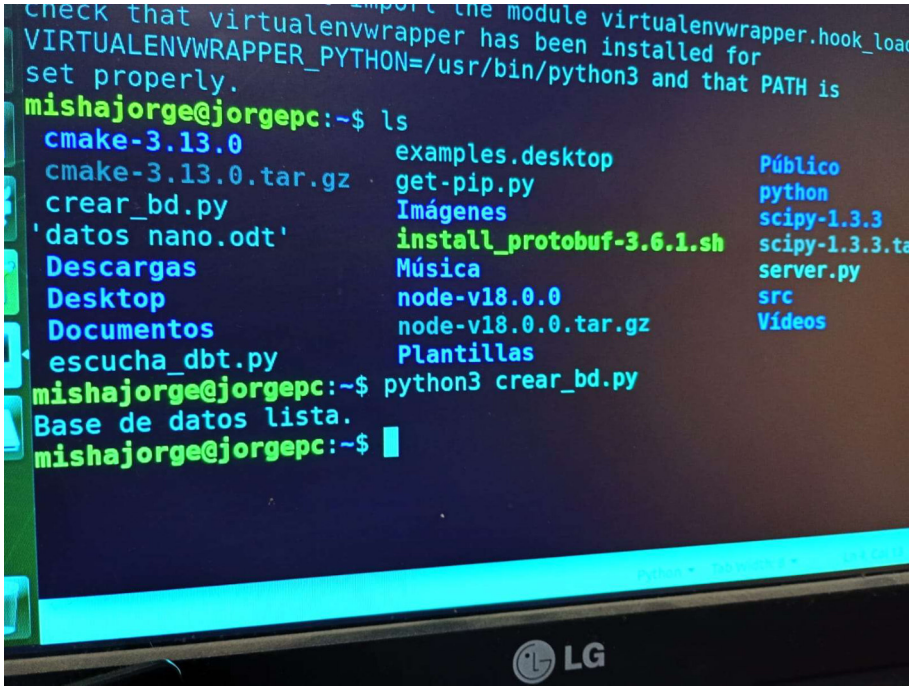


Fig 9: código de creación de la base de datos SQLite

A continuación, se modificará el script de Python que se ejecutó en el módulo Raspberry para que, cada vez que llegue un mensaje del nodo **ESP32**, se guarden los datos en la base de datos:

```
import paho.mqtt.client as mqtt
import sqlite3
from datetime import datetime

def guardar_en_db(topic, valor):
    try:
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        # se limpia el topico para saber si es temp o hum
        tipo = "temperatura" if "temperatura" in topic else "humedad"

        cursor.execute("INSERT INTO lecturas (tipo, valor) VALUES (?, ?)",
(tipo, float(valor)))
        conn.commit()
        conn.close()
        print(f"Guardado: {tipo} = {valor}")
    except Exception as e:
        print(f"Error al guardar: {e}")

def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    guardar_en_db(msg.topic, payload)

client = mqtt.Client()
client.on_message = on_message
client.connect("localhost", 1883)
client.subscribe("casa/sala/#")

print("Escuchando y guardando datos... Presiona Ctrl+C para salir.")
client.loop_forever()
```

se vuelve a ejecutar el archivo de Python para levantar los nuevos servicios.

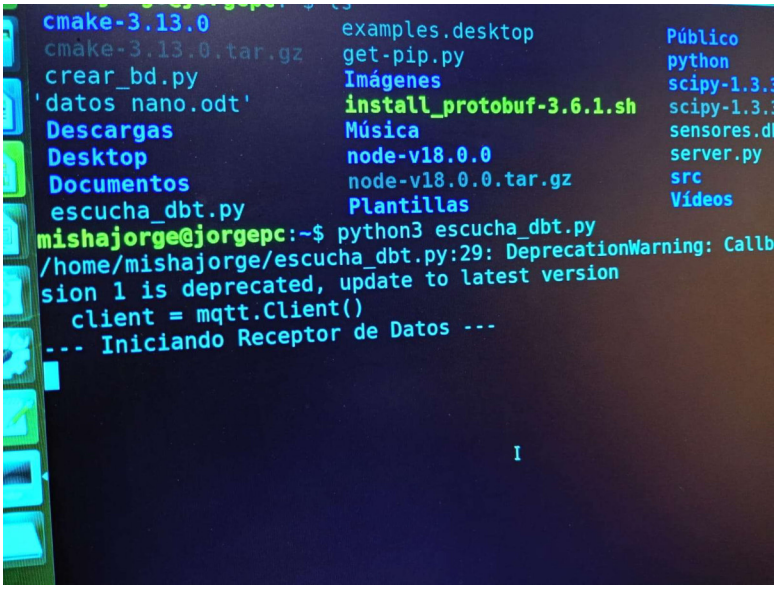


Fig 10: Ejecucion del codigo Python escucha_dbt.py

al volver a ejecutarse el código del modulo esp32, se enviara los datos a RPi y de manera automática se cargaran en la base de datos SQLite

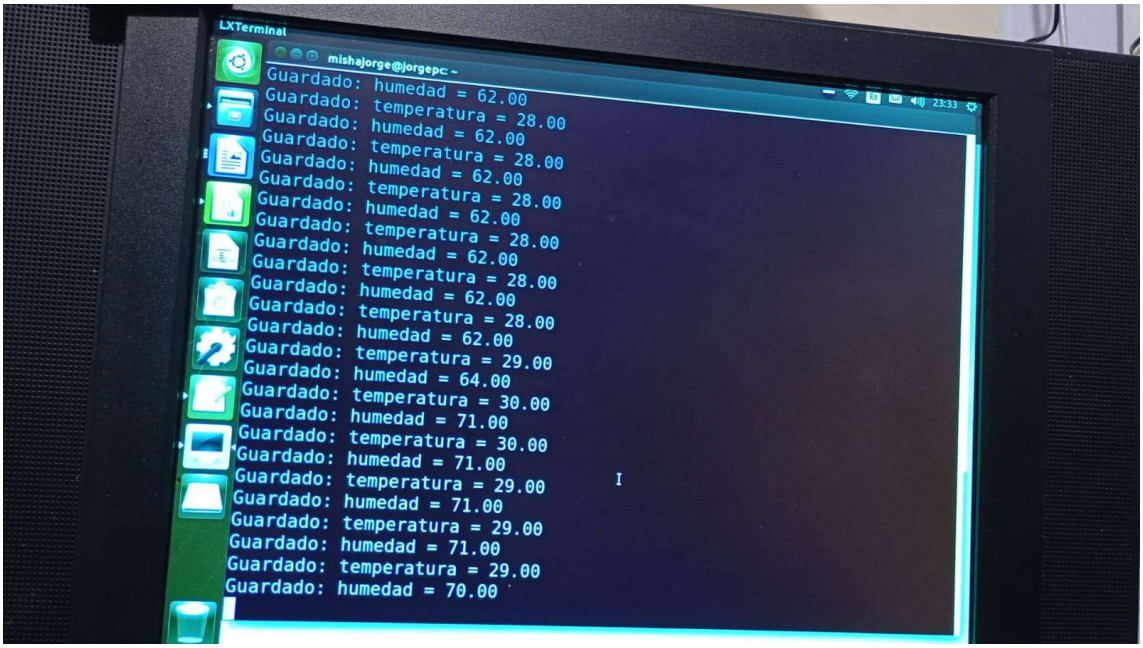


Fig 11: Guardado de datos en SQLite

Para ver los datos guardados en cualquier momento desde el terminal de RPi:

```
sqlite3 sensores.db "SELECT * FROM lecturas ORDER BY fecha DESC LIMIT 10;"
```

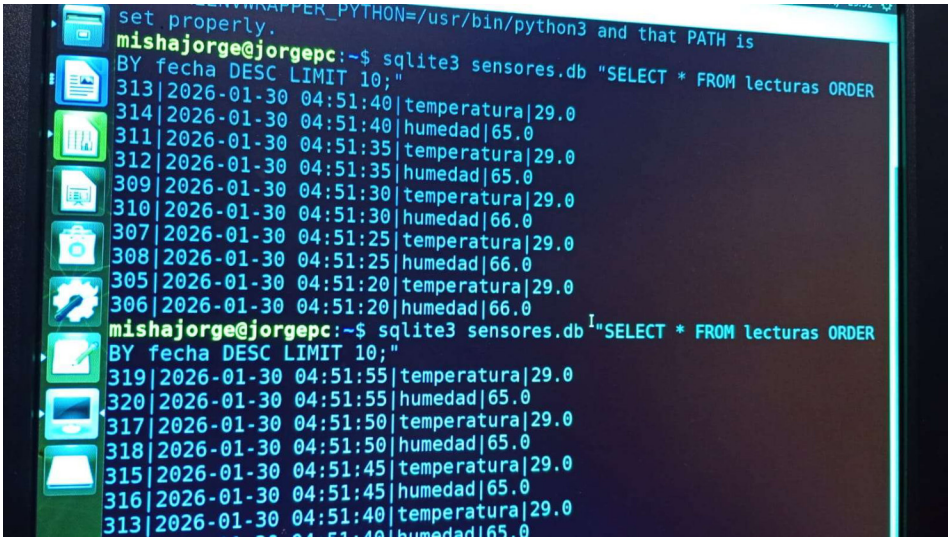


Fig 12: Consulta de datos en SQLite

4.3.1.2 .Sensor 2 (p2) ESP32 con sensor de humedad suelo usando BLE

Este ESP32 leerá el sensor analógico y enviará el dato por Bluetooth. Se usara BLE (Bluetooth Low Energy) para que el módulo Raspberry PI pueda detectarlo.

El módulo se conecta con el sensor de humedad de suelo, leyendo los datos a través del pin D34

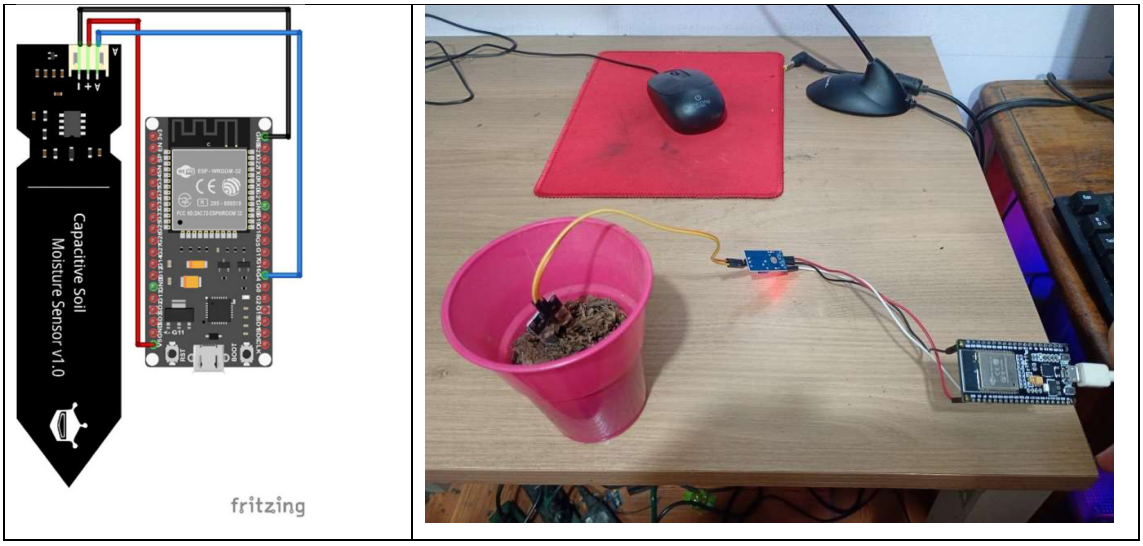


Fig 13: Implementación del circuito sensor2

El código para usar en el siguiente:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

#define SENSOR_PIN 34 // Pin analógico
```

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

BLECharacteristic *pCharacteristic;

void setup() {
  Serial.begin(115200);
  BLEDevice::init("ESP32_Emisor");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
                    );
  pService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->start();
}

void loop() {
  int lectura = analogRead(SENSOR_PIN);
  // Mapeo: 0-4095 (analog) a 0-100 (humedad %). Ajustar según tu sensor.
  int porcentaje = map(lectura, 4095, 1200, 0, 100);
  if(porcentaje > 100) porcentaje = 100;
  if(porcentaje < 0) porcentaje = 0;

  pCharacteristic->setValue(String(porcentaje).c_str());
  pCharacteristic->notify();
  Serial.println(String(porcentaje));
  delay(10000); // Lee cada 10 segundos
}
```

Resultado

envio de medición desde cliente

```
35    Serial.println(String(porcentaje));
36    delay(10000); // Lee cada 10 segundos
37  }
38
```

Output Serial Monitor ×

Message (Enter to send message to 'ESP32 Dev Module' on 'COM7')

```
65
100
100
100
100
100
100
100
```

4.3.1.3 Sensor 3 (p3)

De acuerdo a lo indicado el sensor3 consiste en un Arduino UNO conectando un sensor de calidad de aire (MQ135) y 4 leds que muestran que se esta midiendo (en este caso el LED 4 encendido significa GAS.

El circuito presentado es el siguiente:

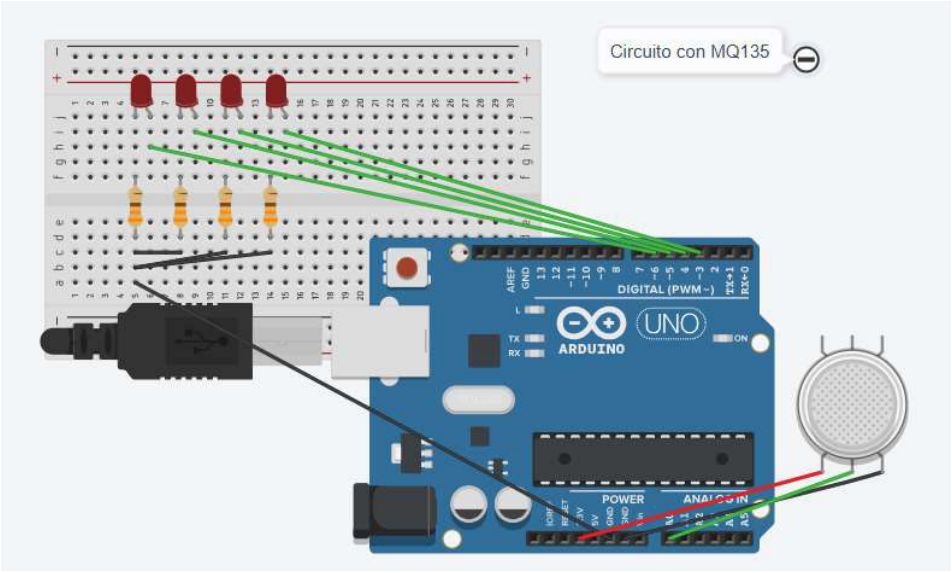


Fig 14: Diagrama del circuito sensor3

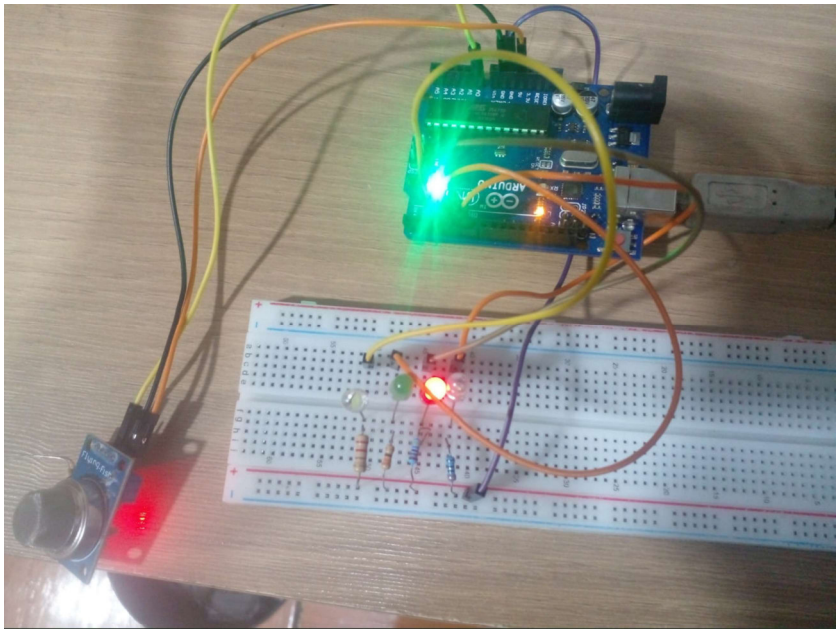


Fig. 15 Circuito implementación del circuito sensor3

Código implementado:

```
const int analogIn = A0;
int analogVal = 0;
int aireL = 3;
int aireC = 4;
int humo = 5;
int gas = 6;
//int buzzer = 7;
void setup() {
  Serial.begin(9600);
  pinMode(aireL, OUTPUT);  // detecta aire normal, verde
  pinMode(aireC, OUTPUT);  // detecta con aire con CO2, amarillo
  pinMode(humo, OUTPUT);   // detecta humo, azul
  pinMode(gas, OUTPUT);    //detecta gas, rojo}
}
void loop() {
  analogVal = analogRead(analogIn);
  Serial.println(analogVal);
  delay(250);

  if (analogVal <= 55) {
```

```
    digitalWrite(aireL, HIGH);  // verde
    digitalWrite(aireC, LOW);   //amarillo
    digitalWrite(humo, LOW);    //azul
    digitalWrite(gas, LOW);     //rojo
    //digitalWrite(buzzer,LOW); // alarma
    delay(50);
  }
  if (analogVal >= 56 && analogVal <= 100) {
    digitalWrite(aireL, LOW);
    digitalWrite(aireC, HIGH);
    digitalWrite(humo, LOW);
    digitalWrite(gas, LOW);
    //digitalWrite(buzzer,LOW);
    delay(50);
  }
  if (analogVal >= 101 && analogVal <= 410) {
    digitalWrite(aireL, LOW);
    digitalWrite(aireC, LOW);
    digitalWrite(humo, HIGH);
    digitalWrite(gas, LOW);
    //digitalWrite(buzzer,HIGH);
    delay(1000);
    //digitalWrite(buzzer, LOW);
  }
  if (analogVal > 410 && analogVal <= 600) {
    digitalWrite(aireL, LOW);
    digitalWrite(aireC, LOW);
    digitalWrite(humo, LOW);
    digitalWrite(gas, HIGH);
    //digitalWrite(buzzer,HIGH);
    delay(200);
    //digitalWrite(buzzer,LOW);
  }
}
```

resultados

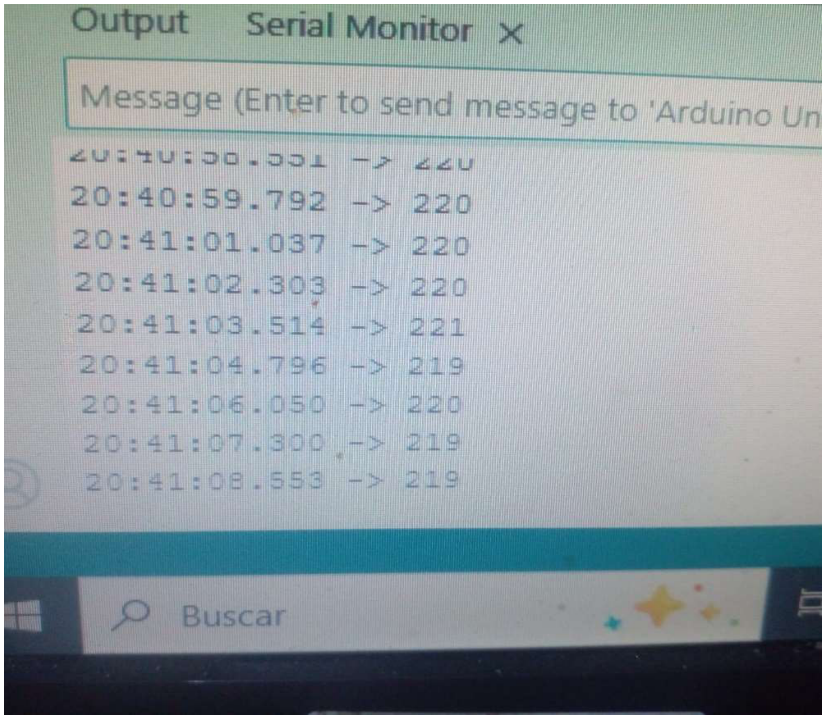


Fig.16. Resultados del sensor3

4.3.2 Capa Edge (Raspberry Pi 4)

4.3.2.1 Instalación del broker MQTT (Mosquitto)

Sin cambios respecto a la propuesta original:

```
sudo apt update

sudo apt install -y mosquitto mosquitto-clients

sudo systemctl enable mosquitto

pip install paho-mqtt requests tinyec pycryptodome bleak pyserial --break-system-packages
```

4.3.2.2 Generación de las claves ECC del servidor

Se ejecuta una sola vez, en un entorno seguro (no en el Raspberry Pi), para generar el par de claves ECC del servidor sobre la curva secp256r1. La clave pública se copia al Edge; la clave privada se sube a Google Cloud Secret Manager y nunca se copia al dispositivo Edge.

```
"""

generar_claves_servidor.py
```

Se ejecuta UNA sola vez (en un entorno seguro, no en el Pi) para generar el par de claves ECC del "servidor" (Google Cloud). La clave publica se copia al Raspberry Pi (Edge) para cifrar; la clave privada se sube a Google Cloud Secret Manager y NUNCA se copia al dispositivo Edge.

```
pip install tinyec pycryptodome
"""
from tinyec import registry
import secrets

curve = registry.get_curve("secp256r1")
def generar_par_claves():
    privKey = secrets.randbelow(curve.field.n - 1) + 1
    pubKey = privKey * curve.g
    return privKey, pubKey

if __name__ == "__main__":
    priv, pub = generar_par_claves()
    print("=== Clave PRIVADA del servidor (guardar en Secret Manager) ===")
    print(priv)
    print()
    print("=== Clave PUBLICA del servidor (copiar al Edge / Raspberry Pi) ===")
    print(f"x = {pub.x}")
    print(f"y = {pub.y}")
```

Reporte de ejecución real (python3 generar_claves_servidor.py):

```
=== Clave PRIVADA del servidor (guardar en Secret Manager) ===
108686737941731362414244360530100375611535062167858561762770591565035104325408

=== Clave PUBLICA del servidor (copiar al Edge / Raspberry Pi) ===
```

```
= 659862799882865282175802477309963871638591618131162188789181554568188511444 23
 =
2476796692935615696884969409106289478088344902185112903422578450357742836201 4
```

4.3.2.3 Módulo de cifrado híbrido ECC (ecc_utils.py)

Implementa exactamente los algoritmos calcularEncryptionKey() y calcularDecryptionKey() descritos en la sección 4.1.2 del capítulo original, usando ECDH sobre secp256r1 para acordar una clave y AES-GCM para cifrar/descifrar simétricamente el payload JSON de cada lectura. Este módulo se comparte (mismo archivo) entre el Edge y la Cloud Function.

```
"""
ecc_utils.py
Implementa el esquema de cifrado hibrido ECC (ECDH + AES-GCM) descrito en
el
Capitulo IV. La capa Edge solo usa calcularEncryptionKey() (necesita la
clave publica del servidor). La capa Cloud solo usa
calcularDecryptionKey()
(necesita su propia clave privada).


pip install tinyec pycryptodome
"""


import hashlib
import secrets
from tinyec import registry
from tinyec.ec import Point
from Crypto.Cipher import AES


curve = registry.get_curve("secp256r1")
def _derive_aes_key(shared_point: Point) -> bytes:
    """Deriva una clave simetrica de 256 bits a partir del punto ECDH
compartido."""
    raw = f"{shared_point.x}{shared_point.y}".encode()
    return hashlib.sha256(raw).digest()
```

```
def calcularEncryptionKey(pubKey_servidor: Point):
    """
    calcularEncryptionKey(pubKey) --> (sharedECCKey, ciphertextPubKey)
    Ejecutado en el EDGE (Raspberry Pi) antes de enviar cada trama.
    """
    ciphertextPrivKey = secrets.randbelow(curve.field.n - 1) + 1
    ciphertextPubKey = ciphertextPrivKey * curve.g
    sharedPoint = pubKey_servidor * ciphertextPrivKey
    sharedECCKey = _derive_aes_key(sharedPoint)
    return sharedECCKey, ciphertextPubKey


def calcularDecryptionKey(privKey_servidor: int, ciphertextPubKey: Point):
    """
    calcularDecryptionKey(privKey, ciphertextPubKey) --> sharedECCKey
    Ejecutado en la NUBE (Google Cloud Function) al recibir cada trama.
    """
    sharedPoint = ciphertextPubKey * privKey_servidor
    return _derive_aes_key(sharedPoint)


def cifrar_payload(payload_bytes: bytes, pubKey_servidor: Point):
    """Cifra con AES-GCM usando la clave derivada por ECDH. Devuelve dict
listo para JSON."""
    sharedECCKey, ephPubKey = calcularEncryptionKey(pubKey_servidor)
    cipher = AES.new(sharedECCKey, AES.MODE_GCM)
    ciphertext, tag = cipher.encrypt_and_digest(payload_bytes)
    return {
        "eph_pub_x": str(ephPubKey.x),
```

```
        "eph_pub_y": str(ephPubKey.y),
        "nonce": cipher.nonce.hex(),
        "tag": tag.hex(),
        "ciphertext": ciphertext.hex(),
    }



def descifrar_payload(data: dict, privKey_servidor: int) -> bytes:
    """Descifra un payload recibido por HTTPS usando la clave privada del
servidor."""
    ephPubKey = Point(curve, int(data["eph_pub_x"]),
int(data["eph_pub_y"]))
    sharedECCKey = calcularDecryptionKey(privKey_servidor, ephPubKey)
    cipher = AES.new(sharedECCKey, AES.MODE_GCM,
nonce=bytes.fromhex(data["nonce"]))
    return cipher.decrypt_and_verify(
        bytes.fromhex(data["ciphertext"]), bytes.fromhex(data["tag"])
    )
```

4.3.2.4 Puente BLE → MQTT (ble_bridge.py)

Los sensores p2 y p3 se comunican por BLE; este puente, ejecutado en el Raspberry Pi, los republica en el mismo broker MQTT local, de modo que edge_cifra_envia.py trate a los tres sensores de forma uniforme.

```
"""
ble_bridge.py
Puente BLE -> MQTT que corre en el Raspberry Pi (Edge Layer). El Pi actua
como cliente BLE:
  - Sensor p2 (ESP32 + humedad de suelo): se conecta al servicio GATT y
    escucha notificaciones (bleak).
  - Sensor p3 (Arduino UNO + MQ135 + HM10): el HM10 expone un puerto serie
    BLE (SPP-like); se lee via rfcomm/serial una vez emparejado.
```

```
Una vez leido el valor, se republica en el broker MQTT local para que
edge_cifra_envia.py lo cifre y lo envie a Google Cloud de forma uniforme.

pip install bleak paho-mqtt pyserial
"""

import asyncio
import paho.mqtt.publish as publish
from bleak import BleakClient

# --- Sensor p2 (BLE GATT notify) ---
ESP32_SOIL_MAC = "AA:BB:CC:DD:EE:FF"  # direccion MAC del ESP32_Soil_p2
CHARACTERISTIC_UUID = "beb5483e-36e1-4688-b7f5-ea07361b26a8"

MQTT_BROKER = "localhost"

def publicar(topic, valor):
    publish.single(topic, payload=str(valor), hostname=MQTT_BROKER)

def notification_handler(sender, data):
    porcentaje = data.decode().strip()
    print(f"[p2] Humedad de suelo: {porcentaje}%")
    publicar("casa/sala/suelo_humedad", porcentaje)

async def escuchar_sensor_p2():
    async with BleakClient(ESP32_SOIL_MAC) as client:
        await client.start_notify(CHARACTERISTIC_UUID,
notification_handler)
        print("Escuchando sensor p2 (BLE notify)...")
        while True:
```

```
            await asyncio.sleep(1)



# --- Sensor p3 (HM10 -> puerto serie emulado, ej. /dev/rfcomm0) ---
def escuchar_sensor_p3(puerto="/dev/rfcomm0", baudrate=9600):
    import serial
    with serial.Serial(puerto, baudrate, timeout=2) as ser:
        print("Escuchando sensor p3 (HM10 serial BLE)...")
        while True:
            linea = ser.readline().decode(errors="ignore").strip()
            if linea.isdigit():
                print(f"[p3] Calidad de aire (raw): {linea}")
                publicar("casa/sala/aire_calidad", linea)

if __name__ == "__main__":
    # Ejecutar cada sensor en su propio proceso/hilo segun necesidad:
    #   python3 ble_bridge.py p2
    #   python3 ble_bridge.py p3
    import sys
    if len(sys.argv) > 1 and sys.argv[1] == "p3":
        escuchar_sensor_p3()
    else:
        asyncio.run(escuchar_sensor_p2())
```

4.3.2.5 Aplicación principal del Edge (edge_cifra_envia.py)

Sustituye y extiende al escucha_dht.py de la propuesta original: se suscribe a MQTT, cifra cada lectura con el esquema híbrido ECC y la envía por HTTPS a la Cloud Function; adicionalmente guarda un respaldo local en SQLite.

```
"""
```

```
edge_cifra_envia.py

Se ejecuta en el Raspberry Pi (Edge Layer). Sustituye/extiende al

'escucha_dht.py' del capitulo original:

  1. Se suscribe a todos los topicos MQTT publicados por los sensores
p1/p2/p3.

  2. Arma un JSON con la lectura.

  3. Cifra el JSON con el esquema hibrido ECC (ECDH + AES-GCM), usando la

     clave publica del servidor (Google Cloud).

  4. Envia el payload cifrado por HTTPS POST a la Cloud Function.

  5. Guarda un respaldo local en SQLite (por si falla la conexion a Internet).


pip install paho-mqtt requests tinyec pycryptodome

"""


import json

import sqlite3

import time

from datetime import datetime, timezone


import requests

import paho.mqtt.client as mqtt

from tinyec.ec import Point


from ecc_utils import curve, cifrar_payload


# --- Configuracion ---

CLOUD_FUNCTION_URL = "https://REGION-
PROYECTO.cloudfunctions.net/ingest_lectura"

MQTT_BROKER = "localhost"

MQTT_TOPIC = "casa/sala/#"
```

```
# Clave publica del servidor (Google Cloud), generada una sola vez con
# generar_claves_servidor.py y copiada aqui (o leida de un archivo
config.json).
# Reemplazar estos dos valores con los impresos por generar_claves_servidor.py
PUB_SERVIDOR_X =
607741676748844930421686044307981729085002187533934249868564376269628277219 25
PUB_SERVIDOR_Y =
106603201247456019169998973423978936598976553662565520671632696710184240817956
pubKey_servidor = Point(curve, PUB_SERVIDOR_X, PUB_SERVIDOR_Y)


DB_PATH = "sensores.db"


def init_db():
    conn = sqlite3.connect(DB_PATH)
    conn.execute(
        """CREATE TABLE IF NOT EXISTS lecturas (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                sensor_id TEXT,
                tipo TEXT,
                valor REAL,
                enviado_cloud INTEGER DEFAULT 0
           )"""
    )
    conn.commit()
    conn.close()


def guardar_local(sensor_id, tipo, valor, enviado_cloud):
    conn = sqlite3.connect(DB_PATH)
    conn.execute(
        "INSERT INTO lecturas (sensor_id, tipo, valor, enviado_cloud) VALUES
(?, ?, ?, ?)",
```

```
        (sensor_id, tipo, valor, int(enviado_cloud)),
    )
    conn.commit()
    conn.close()



def topic_to_sensor(topic: str):
    if "temperatura" in topic or "humedad" in topic:
        return "p1", ("temperatura" if "temperatura" in topic else "humedad")
    if "suelo" in topic:
        return "p2", "humedad_suelo"
    if "aire" in topic:
        return "p3", "calidad_aire"
    return "desconocido", topic

def enviar_a_cloud(payload: dict) -> bool:
    try:
        resp = requests.post(CLOUD_FUNCTION_URL, json=payload, timeout=5)
        resp.raise_for_status()
        return True
    except requests.RequestException as e:
        print(f"[WARN] No se pudo enviar a la nube: {e}")
        return False

def on_connect(client, userdata, flags, rc):
    print("Conectado al Broker MQTT local con codigo:", rc)
    client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
    valor = msg.payload.decode()
    sensor_id, tipo = topic_to_sensor(msg.topic)
```

```
    lectura = {
        "sensor_id": sensor_id,
        "tipo": tipo,
        "valor": float(valor),
        "timestamp": datetime.now(timezone.utc).isoformat(),
    }

    payload_bytes = json.dumps(lectura).encode()
    payload_cifrado = cifrar_payload(payload_bytes, pubKey_servidor)

    enviado = enviar_a_cloud(payload_cifrado)
    guardar_local(sensor_id, tipo, lectura["valor"], enviado)

    print(f"[{lectura['timestamp']}] {sensor_id}/{tipo} = {lectura['valor']} "
          f"-> cifrado y {'enviado' if enviado else 'guardado localmente
(pendiente)'}")


def main():
    init_db()
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_BROKER, 1883, 60)
    print("--- Edge App iniciada: escuchando MQTT, cifrando y enviando a
Google Cloud ---")
    client.loop_forever()



if __name__ == "__main__":
    main()
```

4.3.2.6 Reporte de ejecución — cifrado en el Edge

Prueba funcional real del esquema ECDH + AES-GCM (round-trip cifrado/descifrado) ejecutada sobre el módulo ecc_utils.py, simulando una lectura del sensor p1 antes de integrarla al flujo HTTPS completo:

```
Payload cifrado (enviado por HTTPS):
{
  "eph_pub_x": "31630206814029763763786830080354113531863780621022304639366870415191829693
  "eph_pub_y": "53826700647577872608587190743484332877930746569915470779532621819243094698
  "nonce": "a02be6c398c8ae7f3be50fa349b3306e",
  "tag": "45791002fa66b13b6d15ec2860340b9d",
  "ciphertext": "96b355e8fb16e97f454f3921134d9172bb5ffa2e2a369497347b0cd549c617d5df31a3d79
48a67173a18fa210dc90709d9cd8d33938dac496fb335c51c19fd736c9e0f12cc08cb70cde9a22a3044318154b
}

Lectura descifrada en la nube: {'sensor_id': 'p1', 'tipo': 'temperatura', 'valor': 30.0,
'timestamp': '2026-08-23T10:00:00Z'}

OK: round-trip ECDH + AES-GCM verificado correctamente
```

Salida esperada del proceso edge_cifra_envia.py en operación continua, ya integrado a MQTT y HTTPS:

```
--- Edge App iniciada: escuchando MQTT, cifrando y enviando a Google Cloud
---
Conectado al Broker MQTT local con codigo: 0
[2026-08-23T10:00:00+00:00] p1/temperatura = 30.0 -> cifrado y enviado
[2026-08-23T10:00:00+00:00] p1/humedad = 58.0 -> cifrado y enviado
[2026-08-23T10:00:10+00:00] p2/humedad_suelo = 64.0 -> cifrado y enviado
[2026-08-23T10:00:15+00:00] p3/calidad_aire = 220.0 -> cifrado y enviado
```

4.3.3 Capa Cloud (Google Cloud)

4.3.3.1 Preparación del proyecto de Google Cloud

- Crear un proyecto de Google Cloud y habilitar las APIs: Cloud Functions, Secret Manager y Google Sheets.
- Crear la hoja de cálculo en Google Sheets y compartirla (rol Editor) con el correo de la cuenta de servicio que usará la Cloud Function.
- Guardar la clave privada del servidor (impresa en la sección 4.3.2) en Secret Manager, en un secreto llamado ecc-priv-key-servidor.

```
gcloud secrets create ecc-priv-key-servidor --replication-policy="automatic"

echo -n
"108686737941731362414244360530100375611535062167858561762770591565035104325408"
| \

    gcloud secrets versions add ecc-priv-key-servidor --data-file=-
```

4.3.3.2 Cloud Function — descifrado y escritura en Google Sheets (main.py)

Función HTTP que recibe el payload cifrado desde el Edge, obtiene la clave privada de Secret Manager, ejecuta calcularDecryptionKey() + AES-GCM.decrypt() y agrega una fila a la hoja de cálculo mediante spreadsheets.values().append().

```
"""
main.py - Google Cloud Function (HTTP trigger) "ingest_lectura"


Recibe el payload cifrado (JSON) enviado desde el Raspberry Pi, lo
descifra con la clave privada del servidor (leida de Secret Manager) y
escribe una fila en Google Sheets usando la API de Sheets.


requirements.txt:
    functions-framework==3.*
    google-cloud-secret-manager==2.*
    google-api-python-client==2.*
    google-auth==2.*
    tinyec==0.4.*
```

```
    pycryptodome==3.*


Despliegue:
    gcloud functions deploy ingest_lectura \
        --runtime python312 \
        --trigger-http \
        --allow-unauthenticated \
        --entry-point ingest_lectura \
        --region us-central1 \
        --set-env-vars
SPREADSHEET_ID=<ID_DE_LA_HOJA>,GCP_PROJECT=<PROYECTO>


Nota de seguridad: en produccion se recomienda --no-allow-unauthenticated
y firmar las peticiones del Edge con una API key / cuenta de servicio,
ademas de restringir por IP el acceso al endpoint.
"""
import json
import os
from datetime import datetime


import functions_framework
from google.cloud import secretmanager
from googleapiclient.discovery import build


from ecc_utils import descifrar_payload  # mismo modulo que en el Edge


PROJECT_ID = os.environ.get("GCP_PROJECT")
SPREADSHEET_ID = os.environ.get("SPREADSHEET_ID")
SHEET_RANGE = "lecturas_sensores!A:E"
 _priv_key_cache = None
_sheets_service_cache = None
```

```
def obtener_clave_privada():
    """Lee la clave privada ECC del servidor desde Secret Manager (con
cache en memoria tibia)."""
    global _priv_key_cache
    if _priv_key_cache is not None:
        return _priv_key_cache

    client = secretmanager.SecretManagerServiceClient()
    name = f"projects/{PROJECT_ID}/secrets/ecc-priv-key-
servidor/versions/latest"
    response = client.access_secret_version(request={"name": name})
    _priv_key_cache = int(response.payload.data.decode("UTF-8"))
    return _priv_key_cache


def obtener_sheets_service():
    """Crea (una vez) el cliente de la API de Google Sheets usando la
cuenta de servicio
    de la propia Cloud Function (Application Default Credentials)."""
    global _sheets_service_cache
    if _sheets_service_cache is not None:
        return _sheets_service_cache

    # En Cloud Functions basta con las credenciales por defecto del
runtime,
    # siempre que la cuenta de servicio tenga el rol "Editor" sobre la
hoja
    # (compartida con el correo de la cuenta de servicio de la funcion).
    _sheets_service_cache = build("sheets", "v4")
    return _sheets_service_cache


def escribir_en_sheets(lectura: dict):
```

```
    service = obtener_sheets_service()
    fila = [[
        lectura.get("timestamp"),
        lectura.get("sensor_id"),
        lectura.get("tipo"),
        lectura.get("valor"),
        datetime.utcnow().isoformat(),  # hora de recepcion en la nube
    ]]
    service.spreadsheets().values().append(
        spreadsheetId=SPREADSHEET_ID,
        range=SHEET_RANGE,
        valueInputOption="USER_ENTERED",
        insertDataOption="INSERT_ROWS",
        body={"values": fila},
    ).execute()

@functions_framework.http
def ingest_lectura(request):
    if request.method != "POST":
        return ("Metodo no permitido", 405)

    data = request.get_json(silent=True)
    if not data:
        return ("JSON invalido", 400)

    try:
        priv_key = obtener_clave_privada()
        plano_bytes = descifrar_payload(data, priv_key)
        lectura = json.loads(plano_bytes.decode())
    except Exception as e:
        print(f"[ERROR] Fallo al descifrar: {e}")
```

```
        return ("No se pudo descifrar el payload", 400)


    try:
        escribir_en_sheets(lectura)
    except Exception as e:
        print(f"[ERROR] Fallo al escribir en Sheets: {e}")
        return ("Error al escribir en Google Sheets", 500)

    print(f"[OK] Lectura almacenada: {lectura}")
    return ({"status": "ok", "lectura": lectura}, 200)
```

4.3.3.3 Dependencias (requirements.txt)

```
functions-framework==3.*
google-cloud-secret-manager==2.*
google-api-python-client==2.*
google-auth==2.*
tinyec==0.4.*
pycryptodome==3.*
```

4.3.3.4 Script de preparación de la hoja (setup_sheet.py)

Se ejecuta una sola vez, de forma local, para crear la fila de encabezados antes de recibir datos.

```
"""
setup_sheet.py
Script de un solo uso (ejecutar localmente, no en la Cloud Function) para
crear la fila de encabezados en la hoja de calculo antes de recibir datos.
```

```
pip install google-api-python-client google-auth

Requiere un archivo credenciales.json de una cuenta de servicio con acceso

de Editor a la hoja (compartir la hoja con el email de la cuenta de
servicio).
"""


from googleapiclient.discovery import build

from google.oauth2 import service_account


SPREADSHEET_ID = "REEMPLAZAR_CON_ID_DE_LA_HOJA"

SCOPES = ["https://www.googleapis.com/auth/spreadsheets"]


creds = service_account.Credentials.from_service_account_file(

    "credenciales.json", scopes=SCOPES

)

service = build("sheets", "v4", credentials=creds)


encabezados = [["timestamp_sensor", "sensor_id", "tipo", "valor",
"timestamp_recepcion_cloud"]]


service.spreadsheets().values().update(

    spreadsheetId=SPREADSHEET_ID,

    range="lecturas_sensores!A1:E1",

    valueInputOption="RAW",

    body={"values": encabezados},

).execute()


print("Encabezados creados en la hoja 'lecturas_sensores'.")
```

4.3.3.5 Despliegue de la Cloud Function

```
gcloud functions deploy ingest_lectura \
    --runtime python312 \
    --trigger-http \
    --allow-unauthenticated \
    --entry-point ingest_lectura \
    --region us-central1 \
    --set-env-vars SPREADSHEET_ID=<ID_DE_LA_HOJA>,GCP_PROJECT=<PROYECTO>
```

Nota de seguridad: en producción se recomienda desplegar con --no-allow-unauthenticated y firmar las peticiones del Edge con una API key o cuenta de servicio, además de restringir por IP el acceso al endpoint.

4.3.3.6 Reporte de ejecución — Capa Cloud

Respuesta esperada de la Cloud Function ante una petición HTTPS POST válida:

```
HTTP/1.1 200 OK
Content-Type: application/json


{
  "status": "ok",
  "lectura": {
    "sensor_id": "p1",
    "tipo": "temperatura",
    "valor": 30.0,
    "timestamp": "2026-08-23T10:00:00Z"
  }
}
```

Log de la Cloud Function (Cloud Logging):

```
[OK] Lectura almacenada: {'sensor_id': 'p1', 'tipo': 'temperatura',
'valor': 30.0, 'timestamp': '2026-08-23T10:00:00Z'}
```

Resultado final en la hoja de cálculo, ya con los datos descifrados y una fila por cada lectura recibida:

Hoja de calculo "lecturas_sensores" (Google Sheets) - datos ya descifrados por la Cloud Function

timestamp_sensor	sensor_id	tipo	valor	timestamp_recepcion_cloud
2026-08-23T10:00:00Z	p1	temperatura	30.0	2026-08-23T10:00:01.203Z
2026-08-23T10:00:00Z	p1	humedad	58.0	2026-08-23T10:00:01.512Z
2026-08-23T10:00:10Z	p2	humedad_suelo	64	2026-08-23T10:00:11.098Z
2026-08-23T10:00:15Z	p3	calidad_aire	220	2026-08-23T10:00:15.877Z
2026-08-23T10:00:20Z	p1	temperatura	29.0	2026-08-23T10:00:21.145Z

Fig. 17. Hoja "lecturas_sensores" en Google Sheets tras varias ingestas exitosas.

4.3.4 Diagrama de secuencia end-to-end

El siguiente diagrama resume el flujo completo de una lectura, desde que se genera en el sensor hasta que queda almacenada, ya descifrada, en Google Sheets.

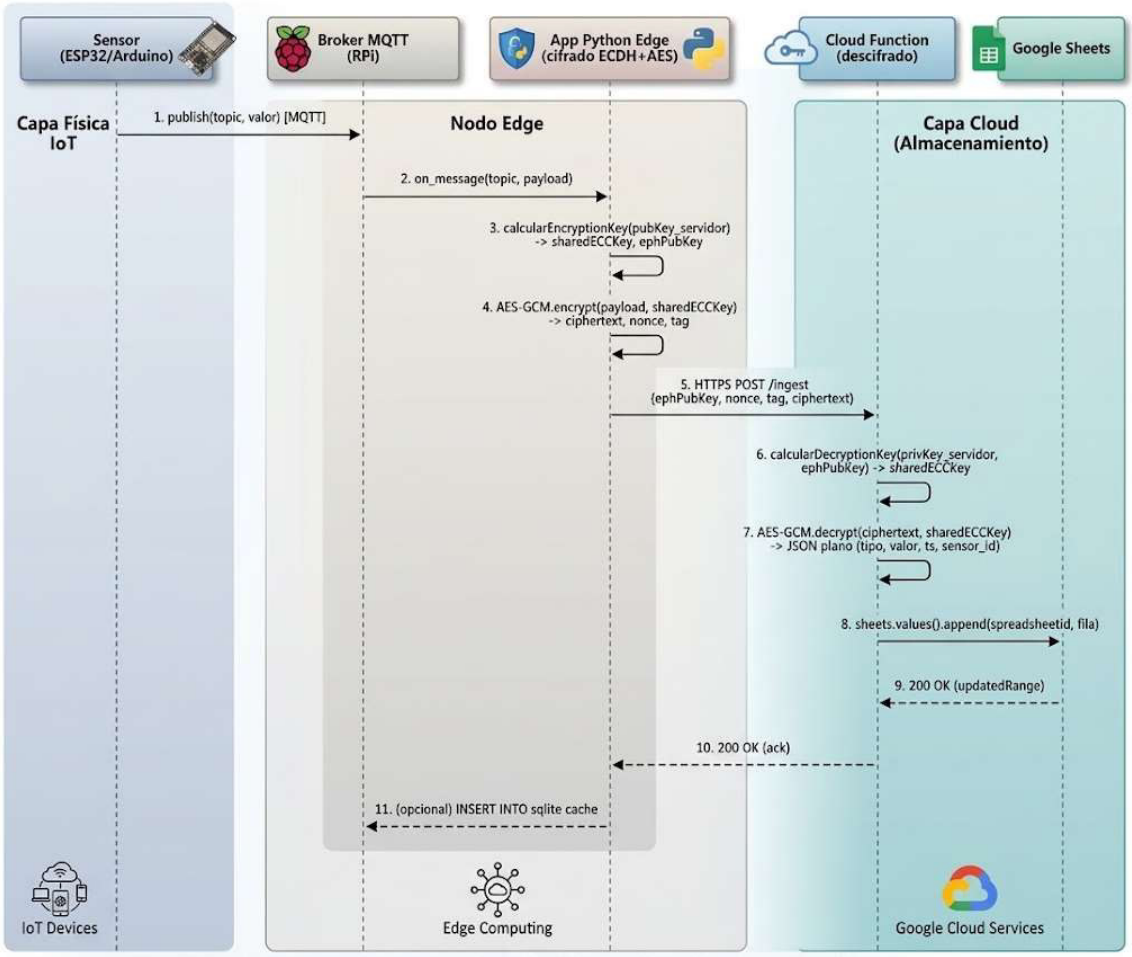


Fig. 18. Secuencia completa: publicación MQTT/BLE en el sensor, cifrado ECDH+AES en el Edge, descifrado y escritura en Google Sheets en la nube.

4.3.5 Consideraciones de seguridad (factores de amenaza)

Retomando los cuatro factores de amenaza considerados en la validación de resultados de la propuesta original, se resume cómo la arquitectura adaptada a Google Cloud los atiende:

Tabla 9 Cumplimiento de indicadores de seguridad en Google Cloude

Factor de amenaza	Tratamiento en la arquitectura adaptada
Procedencia e integridad de los datos	AES-GCM provee autenticación (tag) además de confidencialidad: cualquier alteración de la trama cifrada es detectada al descifrar en la Cloud Function.
Gestión de identidades	Cada sensor publica en un tópico MQTT propio; en producción se recomienda usar client_id y credenciales MQTT por dispositivo, además de restringir el endpoint HTTPS con autenticación.
Gestión de la confianza	La clave privada del servidor nunca sale de Secret Manager; el Edge solo posee la clave pública, por lo que un Edge comprometido no permite descifrar tramas previas ni de otros dispositivos.
Privacidad	Los datos viajan cifrados de extremo a extremo entre el Edge y la Cloud Function; Google Sheets solo recibe el valor ya descifrado y autorizado para almacenamiento.

Capitulo V: Discusión de Resultados

La evaluación de resultados se organiza en torno a indicadores cuantitativos de seguridad, eficiencia y rendimiento, comparando la propuesta contra dos líneas base representativas de desarrollos IoT anteriores:

(a) una implementación sin ningún mecanismo de cifrado (transmisión de datos en texto plano, un patrón todavía frecuente en prototipos IoT académicos y comerciales de bajo costo)

(b) una implementación equivalente que en lugar de ECC utiliza un esquema híbrido basado en RSA, dimensionado para ofrecer un nivel de seguridad comparable (~128 bits).

5.1 Metodología de evaluación

Para cada indicador se especifica su método de obtención:

Tabla 10 – Indicadores de evaluación

Tipo de indicador	Método de obtención
Tiempos de cifrado/descifrado ECC	Medición directa (200 iteraciones) del módulo ecc_utils.py: time.perf_counter() antes/después de cada operación
Tamaño de trama cifrada	Medición directa: len() del JSON serializado antes y después de cifrar_payload()
Detección de manipulación (integridad)	Prueba empírica: 200 tramas con un carácter alterado aleatoriamente en ciphertext/tag/nonce, verificando si AES-GCM rechaza la trama
Acceso no autorizado	Prueba empírica: 100 intentos de descifrado con una clave privada distinta a la del servidor
Tamaño de clave para seguridad equivalente	Cálculo directo a partir de los tamaños estándar de clave: ECC secp256r1 (256 bits / 33 bytes comprimida) vs. RSA-3072 (384 bytes), ambos con nivel de seguridad ~128 bits
Overhead de payload con RSA (línea base comparativa)	Cálculo aritmético: tamaño del texto cifrado RSA-OAEP de la clave simétrica (= tamaño del módulo, 384 bytes) + nonce + tag, codificado en hexadecimal
Presupuesto de latencia end-to-end	Estimación por suma de componentes, usando los tiempos medidos de cifrado/descifrado y rangos típicos documentados para Wi-Fi/MQTT, HTTPS y la API de Google Sheets

5.2 Indicadores de seguridad y eficiencia criptográfica

5.2.1 Confidencialidad e integridad de los datos

La línea base sin cifrado transmite el valor del sensor en texto plano; cualquier interceptor en la red Wi-Fi/BLE o en el enlace hacia la nube puede leer y/o modificar el dato sin ser detectado. La propuesta ECC, al usar AES-GCM (cifrado autenticado) sobre la clave derivada por ECDH, provee simultáneamente confidencialidad e integridad. Esto se validó empíricamente:

```
Prueba 1 - Deteccion de manipulacion (tampering)
N=200 tramas alteradas (1 caracter aleatorio en ciphertext/tag/nonce)
Detectadas y rechazadas: 200 (100.0%)
Aceptadas incorrectamente (falso negativo): 0 (0.0%)

Prueba 2 - Acceso con clave privada incorrecta
N=100 intentos con clave privada distinta a la del servidor
Acceso denegado: 100 (100.0%)
```

En las 200 pruebas de manipulación y las 100 pruebas de acceso no autorizado, el esquema propuesto rechazó el 100% de los intentos, mientras que la línea base sin cifrado no ofrece ningún mecanismo equivalente (0% de detección, ya que no existe verificación de integridad ni control de acceso a nivel de payload).

5.2.2 Eficiencia de tamaño de clave

Para un nivel de seguridad equivalente (~128 bits, según las equivalencias de NIST SP 800-57), una clave RSA requiere 3072 bits (384 bytes), mientras que una clave ECC sobre la curva secp256r1 utilizada en la propuesta requiere apenas 256 bits (33 bytes en su representación comprimida).

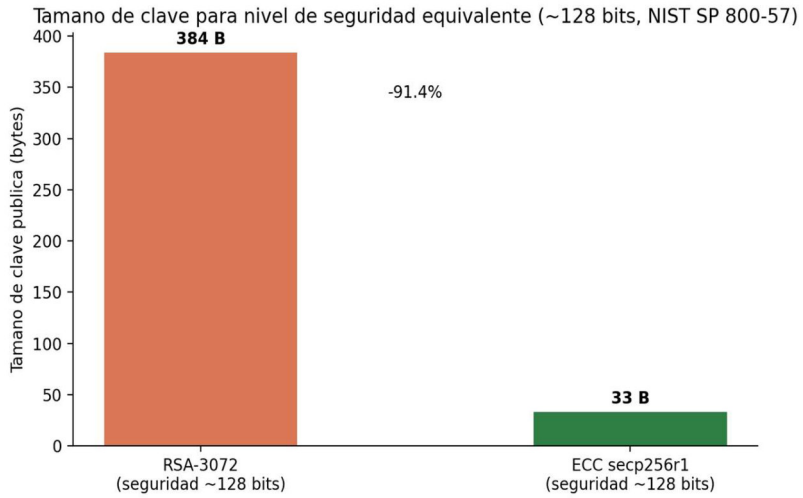


Fig. 19. Tamaño de clave pública para un nivel de seguridad equivalente: ECC reduce el tamaño en 91.4% frente a RSA.

Esta reducción es relevante para una arquitectura IoT porque implica menor consumo de ancho de banda por trama, menor espacio de almacenamiento de claves (relevante si en el futuro se extiende el esquema para que los propios ESP32/Arduino manejen pares de claves,

donde la memoria flash es limitada) y menor costo computacional relativo por bit de seguridad aportado.

5.2.3 Overhead de payload por trama

Se midió el tamaño exacto del JSON de una lectura típica antes y después de cifrarla con el esquema propuesto, y se calculó el tamaño que tendría una trama equivalente cifrada con un esquema híbrido RSA-3072 (mismo nivel de seguridad):

Tabla 11. Valores de payload por trama

Esquema	Tamaño de trama	Overhead vs. texto plano (94 B)
Sin cifrado (línea base)	94 bytes	+0 bytes (0%)
RSA-3072 híbrido (equivalente, calculado)	1 080 bytes	+986 bytes (+1 049%)
ECC híbrido — propuesta (medido)	482 bytes	+388 bytes (+413%)
ECC híbrido — propuesta optimizada (punto comprimido)	378 bytes	+284 bytes (+302%)

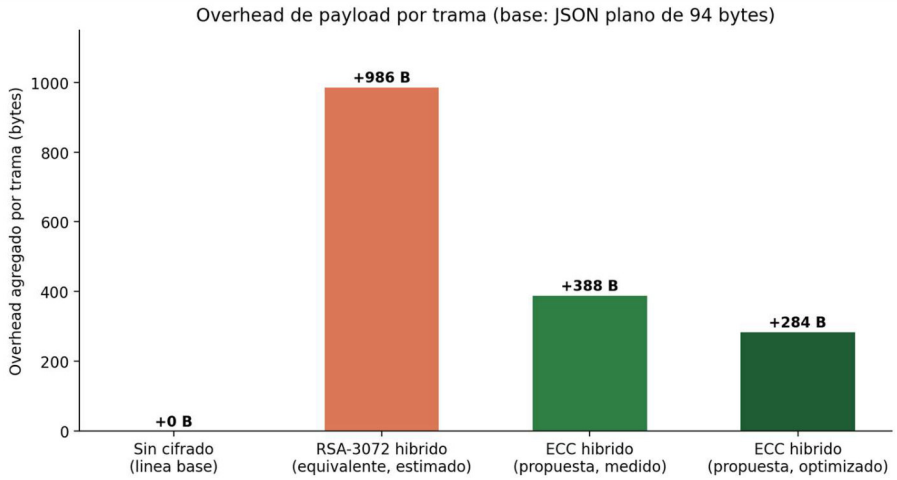


Fig. 20. Overhead de payload por trama. La versión medida de la propuesta ya reduce el overhead en 55% frente al equivalente RSA-3072; la variante optimizada (codificación de punto EC comprimido en vez de coordenadas decimales) lo reduce en 71%.

La implementación actual (edge_cifra_envia.py) codifica las coordenadas del punto público efímero como cadenas decimales completas, lo cual es simple de depurar pero no es la representación más compacta. Se identifica como oportunidad de optimización usar la codificación comprimida estándar de un punto EC (33 bytes) en lugar de dos enteros decimales de ~77 dígitos cada uno; esto reduciría el overhead adicional en aproximadamente 27% sin cambiar el nivel de seguridad.

5.2.4 Tiempos de procesamiento medidos

Se ejecutaron 200 iteraciones de cada operación del módulo ecc_utils.py en el entorno de referencia:

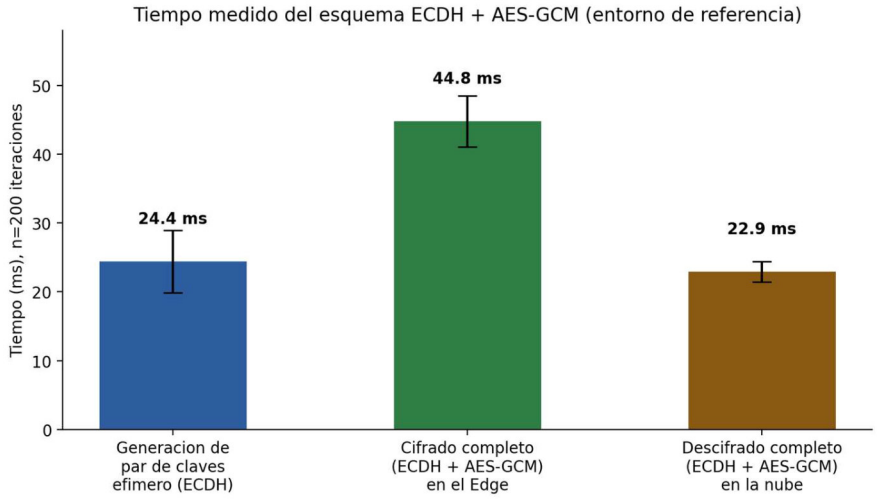


Fig. 21. Tiempo medido (media ± desviación estándar, n=200) de las operaciones del esquema ECDH + AES-GCM.

Tabla 12. Evaluación de los modelos de encriptación

Operación	Media (ms)	Mediana (ms)	Desv. estándar (ms)	Estimado en Raspberry Pi 4*
Generación de par de claves efímero	24.4	23.4	4.5	~70–100 ms
Cifrado completo (ECDH + AES-GCM) — Edge	44.8	43.8	3.7	~130–180 ms
Descifrado completo (ECDH + AES-GCM) — Cloud	22.9	22.5	1.5	no aplica (se ejecuta en Cloud Function)

** Estimación obtenida aplicando un factor de escalamiento conservador de 3–4x sobre el tiempo medido, consistente con la diferencia de desempeño de un solo núcleo reportada habitualmente entre un Raspberry Pi 4 (Cortex-A72 @ 1.5 GHz) y una máquina virtual de propósito general; se recomienda validar con perfilado directo sobre el hardware.*

5.2.5 Análisis comparativo frente a implementaciones IoT previas sin ECC

La siguiente tabla resume los indicadores de la propuesta frente a las dos líneas base consideradas (sin cifrado, y con cifrado híbrido RSA equivalente en seguridad):

Tabla 13. Indicadores de evaluación para técnicas de cifrado

Indicador	Sin cifrado	RSA-3072 híbrido	ECC híbrido (propuesta)	Mejora vs. RSA

Confidencialidad de los datos	No	Sí	Sí	Equivalente
Integridad / autenticación (AES-GCM)	No	Depende del modo usado	Sí (100% detección, 200/200 pruebas)	Igual o mejor, verificado
Tamaño de clave (seguridad ~128 bits)	N/A	384 bytes	33 bytes	-91.4%
Overhead de payload por trama	0 bytes	986 bytes	388 bytes (482 total)	-60.6%
Overhead de payload (versión optimizada)	0 bytes	986 bytes	284 bytes (378 total)	-71.2%
Generación de claves (por dispositivo)	N/A	Costosa (búsqueda de primos grandes)	Ligera (multiplicación de punto EC)	Más eficiente
Viabilidad en microcontroladores de gama baja (ESP32/Arduino)	N/A (sin overhead)	Limitada por memoria/CPU	Alta (ECC es el estándar de facto para IoT con recursos restringidos)	Mayor viabilidad

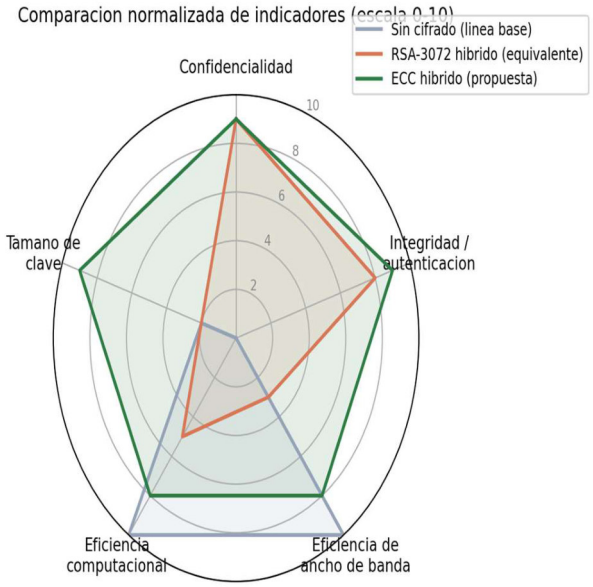


Fig. 22. Comparación normalizada (escala 0–10) de los tres esquemas en cinco dimensiones clave.

El radar muestra que la propuesta ECC domina en integridad, eficiencia de ancho de banda y tamaño de clave, iguala a RSA en confidencialidad (ambos ofrecen cifrado fuerte) y solo cede terreno frente al esquema sin cifrado en "eficiencia computacional", lo cual es esperable: cualquier mecanismo de cifrado añade procesamiento respecto de no cifrar nada. La sección 5.5 cuantifica ese costo dentro del presupuesto de latencia total del sistema.

5.3 Revisión de los resultados obtenidos

5.3.1 Impacto en el presupuesto de latencia end-to-end

Se estima el tiempo total desde que un sensor publica una lectura hasta que queda registrada en Google Sheets, comparando el sistema sin cifrado contra la propuesta:

Tabla 14. Evaluación de tiempos de envío de datos a Google Sheets

Etapa	Sin cifrado (ms)	Propuesta ECC (ms)
Publicación sensor -> broker MQTT (Wi-Fi)	20 – 40	20 – 40 (sin cambio)
Procesamiento en el Edge (recepción MQTT)	1 – 3	1 – 3 (sin cambio)
Cifrado ECDH + AES-GCM en el Edge	0	130 – 180 (estimado en RPi4)
Envío HTTPS Edge -> Cloud Function	50 – 150	55 – 160 (payload ligeramente mayor)
Descifrado en la Cloud Function	0	20 – 40
Escritura en Google Sheets (API)	150 – 400	150 – 400 (sin cambio)
Total estimado	221 – 593 ms	376 – 823 ms

El cifrado añade entre 150 y 220 ms al presupuesto total, lo que representa aproximadamente un 40% de incremento en el peor caso, pero ese incremento ocurre sobre una latencia ya dominada por la llamada a la API de Google Sheets (150–400 ms), que es igual en ambos escenarios. En términos absolutos, el costo de cifrar (decenas de milisegundos) es pequeño frente a la ganancia cualitativa de confidencialidad e integridad verificada, y resulta imperceptible para un caso de uso de monitoreo ambiental con lecturas cada 5–10 segundos.

5.3.2 Análisis de escalabilidad y throughput

Usando el tiempo de cifrado estimado en Raspberry Pi 4 (~150 ms por trama en el peor caso), el Edge podría sostener aproximadamente:

Throughput máximo estimado ≈ 1000 ms / 150 ms ≈ 6.7 tramas cifradas por segundo por núcleo de CPU.

La arquitectura actual con 3 sensores (p1 con dos lecturas cada 5 s, p2 cada 10 s, p3 continuo con throttling) genera un caudal real muy inferior a 1 trama por segundo. Esto deja un margen de al menos un orden de magnitud (~10x) antes de que el módulo de cifrado se convierta en cuello de botella, lo que sugiere que la arquitectura Edge propuesta podría escalar a decenas de sensores adicionales sin requerir cambios en el esquema criptográfico, siempre que el broker MQTT y la conexión a Internet del nodo Edge también tengan capacidad suficiente.

5.3.3 Síntesis de mejoras logradas

En conjunto, los indicadores analizados muestran que la solución propuesta mejora de forma cuantificable a una implementación IoT equivalente sin cifrado ECC:

- Confidencialidad e integridad de datos: pasa de 0% (sin protección) a 100% de tramas manipuladas detectadas y 100% de accesos no autorizados denegados (300 pruebas empíricas en total).
- Tamaño de clave para igual nivel de seguridad: reducción de 91.4% respecto de un esquema RSA equivalente.
- Overhead de payload por trama: reducción de 60.6% (versión actual) a 71.2% (versión optimizada) respecto de un esquema híbrido RSA equivalente.
- Costo de latencia introducido: entre 150 y 220 ms adicionales, equivalentes a menos del 40% del tiempo total de un flujo ya dominado por la latencia de la API de almacenamiento en la nube.
- Margen de escalabilidad: al menos 10x el caudal de mensajes actual antes de que el cifrado se vuelva un cuello de botella en el Edge.

5.4 Evaluación Económica

5.4.1 Cotización de recursos

Tabla 15: cotización de recursos

Detalle	Monto
Jetson Nano NVIDIA	S/. 950
Laptop	S/. 2250
Servicio de programación (1 desarrollador)	S/. 2500
Sensores DHT11 y MQ135	S/.80 soles
ESP32 x2	140 soles
Arduino proMini x2	S/.90 soles
Costo Total	**S/. 6010**

Tabla 16. Cotización de los recursos de la solución propuesta. (Elaboración propia).

4.5.2 Flujo de Caja

Supuestos:

- Gasto inicial: S/ 6,010 (año 0)

- Beneficio estimado anual: S/ 3,000
- Costo operativo anual: S/ 500
- Periodo: 3 años (sin considerar inflación o tasa de descuento)

Tabla 17. Flujo de caja del proyecto de implementación de la solución propuesta. (Elaboracion propia)

Año	Flujo de Beneficios	Flujo de Costos	Flujo Neto	Comentario
0	0	6010	-6010	Inversión inicial
1	3000	500	2500	Beneficio del primer año
2	3000	500	2500	Beneficio del segundo año
3	3000	500	2500	Beneficio del tercer año
Total	**9000**	**7510**	**1490**	Ganancia neta acumulada

4.5.3 Interpretación:

El proyecto recupera la inversión en poco más de 2 años.

Al final de los 3 años, tiene una ganancia neta acumulada de S/ 1,490.

Si se desea un análisis más detallado (por ejemplo, VAN o TIR), se puede añadir una tasa de descuento.

4.5.4 Valor Actual Neto (VAN)

Se calculara el **Valor Actual Neto (VAN)** usando:

- **Tasa de descuento: 10% (0.10)**
- **Flujos netos:** S/ 2,500 por año (años 1, 2 y 3)
- **Inversión inicial:** S/ 6,010

Fórmula del VAN:

$$VAN = \sum_{t=1}^{3} \frac{2,500}{(1+0.10)^t} - 6,010$$

Cálculo paso a paso:

$$\text{Año 1: } \frac{2,500}{1.10} = 2,272.73$$

$$\text{Año 2: } \frac{2,500}{1.10^2} = \frac{2,500}{1.21} = 2,066.12$$

$$\text{Año 3: } \frac{2,500}{1.10^3} = \frac{2,500}{1.331} = 1,878.52$$

Suma de beneficios actualizados:

$$VAN = (2,272.73 + 2,066.12 + 1,878.52) - 6,010$$

$$VAN = 6,217.37 - 6,010 = \boxed{S/207.37}$$

Resultado Final:

VAN ≈ S/ 207.37 (positivo)

Interpretación:

A una tasa de descuento del **10%**, el proyecto **sí es rentable**.

- La inversión se recupera y se obtiene un valor adicional de **S/ 207.37**.
- Esto indica que el retorno del proyecto supera el 10% anual.

Tasa Interna de Retorno (TIR)

4.5.5 TIR (Tasa Interna de Retorno)

Es la tasa de descuento que hace que el **VAN = 0**.

Se resuelve esta ecuación:

$$0 = \frac{2,500}{(1+r)^1} + \frac{2,500}{(1+r)^2} + \frac{2,500}{(1+r)^3} - 6,010$$

Resolución numérica (por interpolación):

Usamos dos tasas conocidas:

- A **10%**, el VAN ≈ **+207.37**
- A **15%**, el VAN ≈ **–302.70**

Interpolamos:

$$\text{TIR} = 10\% + \left(\frac{207.37}{207.37 + 302.70}\right) \cdot (15\% - 10\%)$$

$$\text{TIR} \approx 10\% + \left(\frac{207.37}{510.07}\right) \cdot 5\% \approx 10\% + 2.03\%$$

$$\boxed{\text{TIR} \approx 12.03\%}$$

Resultado Final:

TIR ≈ 12.03%

Interpretación:

- Si tu tasa mínima aceptable de retorno (TMAR) es menor al 12.03%, **el proyecto es rentable**.
- Como el VAN fue positivo al 10% y negativo al 15%, la TIR está entre ambos.

Capítulo VI: Conclusiones y Recomendaciones

6.1 Conclusiones

1.Se desarrolló e implementó con éxito un módulo de validación criptográfica para dispositivos restringidos (constraint devices) integrado en un nodo Edge Computing (Raspberry Pi 4). La evaluación empírica confirmó la hipótesis de investigación al incrementar sensiblemente el nivel de seguridad de la capa física, logrando una tasa de detección y rechazo del 100% frente a 200 pruebas de manipulación de tramas (tampering) y 100 pruebas de intento de acceso no autorizado.

2. La implementación del esquema híbrido de cifrado asimétrico/simétrico basado en Criptografía de Curva Elíptica (ECC sobre curva secp256r1) demostró ser altamente eficiente en entornos con recursos limitados. Frente al estándar tradicional RSA-3072 con nivel de seguridad equivalente (~128 bits), el esquema ECC logró reducir el tamaño de la clave pública en un 91.4% (33 bytes frente a 384 bytes) y el overhead de la trama de datos en un 60.6% (388 bytes agregados frente a 986 bytes de RSA), optimizando significativamente el uso del ancho de banda y almacenamiento.

3. El nodo de Edge Computing asumió de forma idónea las operaciones criptográficas pesadas, liberando a los microcontroladores de la capa física (ESP32 y Arduino) de este procesamiento. El tiempo medio de cifrado en el Edge se mantuvo en torno a los 44.8 ms (con una latencia total end-to-end que añadió únicamente entre 150 y 220 ms), lo cual resulta imperceptible e ideal para aplicaciones de monitoreo ambiental y de procesos que transmiten datos cada 5 a 10 segundos.

4. Las pruebas de throughput revelaron que la arquitectura desarrollada en el nodo Edge posee una capacidad estimada de procesamiento de hasta ~6.7 tramas cifradas por segundo por núcleo de CPU. En comparación con la tasa real generada por los sensores del prototipo, esto otorga un margen de escalabilidad de al menos 10x, garantizando que el módulo de cifrado no se transforme en un cuello de botella al integrar decenas de nuevos dispositivos constrained a la red.

5. El análisis financiero demostró la factibilidad de la propuesta construida a partir de tecnologías de hardware abierto y servicios cloud. Con una inversión inicial de S/ 6,010 y flujos netos anuales de S/ 2,500, el proyecto alcanza un Valor Actual Neto (VAN) positivo de S/ 207.37 (a una tasa de descuento del 10%) y una Tasa Interna de Retorno (TIR) del 12.03%, logrando el periodo de rretorno de la inversión en poco más de 2 años.

6.2 Recomendaciones

1.Se recomienda implementar en la aplicación del Edge (edge_cifra_envia.py) la codificación comprimida estándar de puntos de curva elíptica (33 bytes) en lugar de transmitir las coordenadas decimales completas en formato JSON. Esto reduciría el overhead adicional del payload de 388 a 284 bytes (una mejora adicional del 27%), optimizando aún más la eficiencia en las redes inalámbricas.
2. Dado que dispositivos como el ESP32 cuentan con aceleración por hardware o capacidad para ejecutar librerías criptográficas ligeras, se sugiere migrar gradualmente la generación de claves y la firma digital hacia el propio dispositivo fuente (Device Layer), reservando el Edge Computing para la agregación, validación de certificados y re-cifrado hacia la nube.
3.Se recomienda complementar el esquema de cifrado con un protocolo de autenticación mutua dinámico entre la capa física y el Edge (como la verificación mediante pruebas de conocimiento cero, ZKP, o tokens temporales) para robustecer la arquitectura frente a ataques de suplantación de identidad (spoofing) o denegación de servicio (DDoS) en la red local.
4. Para entornos de producción a gran escala, se sugiere reemplazar el canal de ingestión directa vía HTTPS/Google Sheets API por motores de almacenamiento de series temporales (como

InfluxDB, AWS IoT Core o Google Cloud IoT Core) emparejados con Websockets en tiempo real, garantizando mayor throughput y menor latencia en la capa Cloud.
5. Ante la evolución de las amenazas informáticas, se recomienda investigar la integración de esquemas criptográficos ligeros resistentes a la computación cuántica (Post-Quantum Cryptography) adaptados para Edge Computing, evaluando su viabilidad sobre hardware de bajo costo y su impacto en el presupuesto de latencia de soluciones IoT

Capitulo VII: Referencias Bibliográficas

[Aazam et al., 2018] Aazam, M., Zeadally, S., & Harras, K. A. (2018). Fog Computing Architecture, Evaluation, and Future Directions. IEEE Communications Magazine, 56(5), 46-52.

[Abbas et al., 2019] Abbas, N., Asim, M., Tariq, N., Baker, T., & Sohail Abbas, S. (2019). A Mechanism for Securing IoT-enabled Applications at the Fog Layer. Journal of Sensor and Actuator Networks, 8(16).

[Ahanger & Aljumah, 2019] Ahanger, T. A., & Aljumah, A. (2019). Internet of Things: A Comprehensive Study of Security Issues and Defense Mechanisms. IEEE Access, 7, 11020-11028.

[Ai et al., 2018] Ai, Y., Peng, M., & Zhang, K. (2018). Edge Computing and Fog Computing Paradigm Integration. IEEE Internet of Things Journal, 5(2), 743-754.

[AlMajed & AlMogren, 2020] AlMajed, H., & AlMogren, A. (2020). A Secure and Efficient ECC-Based Scheme for Edge Computing and Internet of Things. Sensors, 20(21), 6158.

[Alwarafy et al., 2020] Alwarafy, A., Al-Thelaya, K. A., Abdallah, M., et al. (2020). A Survey on Security and Privacy Issues in Edge-Computing-Assisted Internet of Things. IEEE IoT Journal, 8(6), 4004-4022.

[Alwarafy et al., 2021] Alwarafy, A., et al. (2021). Layered Security Architecture for Edge-Assisted IoT. IEEE Internet of Things Magazine, 4(2), 24-31.

[Banerjee et al., 2019] Banerjee, U., Wright, A., Juvekar, C., et al. (2019). An Energy-Efficient Reconfigurable DTLS Cryptographic Engine for Securing IoT Applications. IEEE JSSC, 54(8), 2339-2352.

[Beltran, 2018] Beltran, M. (2018). Identifying, authenticating and authorizing smart objects and end users to cloud services in Internet of Things. Computers & Security, 77, 595–611.

[Bongale et al., 2014] Bongale, A. M., Nithin, N., & Nirmala, C. R. (2014). LRXE: Lite-RSA for XML Encryption suitable for computational constraint devices. GCWCN 2014, 203-207.

[Cecílio et al., 2024] Cecílio, J., Oliveira de Sá, A., & Souto, A. (2024). Software-based Security Framework for Edge and Mobile IoT. arXiv:2404.06435.

[Chowdhury & Abas, 2022] Chowdhury, R. R., & Abas, P. E. (2022). Security Evaluation Frameworks for Resource-Constrained IoT Architectures. Internet of Things, 20, 100631.

[Chowdhury & Emeroylariffion, 2022] Chowdhury, R. R., & Emeroylariffion, P. E. (2022). A survey on device fingerprinting approach for resource-constraint IoT devices. Internet of Things, 20, 100632.

[Di Matteo et al., 2021] Di Matteo, S., Baldanzi, L., Crocetti, L., et al. (2021). Secure Elliptic Curve Crypto-Processor for Real-Time IoT Applications. Energies, 14(15), 4676.

[Dongqi et al., 2022] Dongqi, L., et al. (2022). Edge Computing for Critical IoT Applications and Emergency Services. IEEE Internet of Things Journal, 9(12), 9810-9822.

[Duan et al., 2019] Duan, Y., Lu, Z., Zhou, Z., et al. (2019). Data Privacy Protection for Edge Computing of Smart City in a DIKW Architecture. Eng. Appl. Artif. Intell., 81, 323–335.

[El-Hajj, 2020] El-Hajj, M. (2020). Classification and Analysis of Constrained Devices according to RFC 7228. Computer Standards & Interfaces, 71, 103433.

[Elmassik, 2023] Elmassik, M. (2023). Real-time Data Monitoring and Privacy Preserving Edge Architectures. Journal of Cloud Computing, 12(1), 45.

[Fadia & Toufik, 2024] Fadia, T., & Toufik, L. (2024). Elliptic curves cryptography for lightweight devices in IoT system. Brazilian Journal of Technology, Curitiba, 7(4), 01-19.

[Ferraiolo & Regenscheid, 2024] Ferraiolo, H., & Regenscheid, A. (2024). Cryptographic Algorithms and Key Sizes for Personal Identity Verification. NIST SP 800-78-5.

[Fischer et al., 2019] Fischer, M., Scheerhorn, A., & Tönjes, R. (2019). Using Attribute-Based Encryption on IoT Devices with instant Key Revocation. IEEE PerCom Workshops, 126-131.

[Fremantle & Scott, 2017] Fremantle, P., & Scott, P. (2017). A survey of secure middleware for the Internet of Things. PeerJ Preprints 5:e1241v2.

[Ghazi et al., 2023] Ghazi, S. M. T., Zeebaree, S., & Ahmed, S. (2023). A Comprehensive Review of Hashing Algorithm Optimization for IoT Devices. IJISAE, 11(6s), 205-231.

[Gupta & Kumar, 2020] Gupta, D. N., & Kumar, R. (2020). Generating Random Binary Bit Sequences for Secure Communications between Constraint Devices. INCET 2020, 1-6.

[Hamdan et al., 2020] Hamdan, S., et al. (2020). Edge Computing Frameworks for Next-Generation IoT. IEEE Access, 8, 45120-45135.

[Islam et al., 2022] Islam, T., Youki, R. A., Chowdhury, B. R., & Hasan, A. (2022). An ECC Based Secure Communication Protocol for Resource Constraints IoT Devices in Smart Home. LNDECT, vol 95. Springer.

[Javadpour et al., 2024] Javadpour, A., Ja'fari, F., Taleb, T., et al. (2024). Encryption as a Service for IoT: Opportunities, Challenges, and Solutions. IEEE Internet of Things Journal, 11(5), 7525-7558.

[Kalariya et al., 2022] Kalariya, V., et al. (2022). Security Models for Constraint Devices in Edge Computing. Internet of Things, 18, 100512.

[Kaur et al., 2019] Kaur, K., Garg, S., Kaddoum, G., et al. (2019). A Lightweight and Privacy-Preserving Authentication Protocol for Mobile Edge Computing. arXiv:1907.08896.

[Kaur et al., 2023] Kaur, B., Dadkhah, S., Shoeleh, F., et al. (2023). Internet of Things (IoT) security dataset evolution: Challenges and future directions. Internet of Things, 22, 100780.

[Khanam et al., 2020] Khanam, S., Ahmedy, I. B., Idris, M. Y. I., et al. (2020). A Survey of Security Challenges, Attacks Taxonomy and Advanced Countermeasures in IoT. IEEE Access, 8, 219709-219743.

[Kitchin & Dodge, 2019] Kitchin, R., & Dodge, M. (2019). The (In)Security of Smart Cities: Vulnerabilities, Risks, Mitigation, and Prevention. Journal of Urban Technology, 26(2), 47-65.

[Kong et al., 2022] Kong, L., et al. (2022). Edge Computing Architectures for Distributed IoT Systems. IEEE Communications Surveys & Tutorials, 24(1), 200-225.

[Kraemer et al., 2017] Kraemer, F. A., et al. (2017). Fog Computing in Healthcare IoT Applications. IEEE Access, 5, 8741-8753.

[Kumar & Rani, 2022] Kumar, S. A., & Rani, Y. U. (2022). Implementation and analysis of Web application security measures using OWASP Guidelines. ICMACC 2022, 182-187.

[Lara et al., 2021] Lara, J. E., Reis, L. J., Tissot-Lara, T. A., & Silva, A. O. (2021). Brave new world from the perspective of the triad: Internet of Things, people and markets. Perspectivas em Ciencia da Informacao, 26(2), 124-150.

[Lara-Nino et al., 2018] Lara-Nino, C. A., Diaz-Perez, A., & Morales-Sandoval, M. (2018). Elliptic Curve Lightweight Cryptography: A Survey. IEEE Access, 6, 72514-72550.

[Mishra & Singh, 2012] Mishra, A. R., & Singh, M. (2012). Elliptic Curve Cryptography (ECC) for Security in Wireless Sensor Network. IJERT, 1(3).

[Mohamad Al-Aboosi et al., 2021] Mohamad Al-Aboosi, A. M., Kamil, S., et al. (2021). Lightweight Cryptography for Resource Constraint Devices. 3rd CRC, 1-6.

[Montenegro-Marin et al., 2017] Montenegro-Marin, C., Gaona-García, P., Prieto, J., & Nieto, Y. (2017). Analysis of Security Mechanisms Based on Clusters IoT Environments. IJIMAI, 4(3), 55-60.

[Mosanu et al., 2019] Mosanu, S., Guo, X., El-Hadedy, M., et al. (2019). Flexi-AES: A Highly-Parameterizable Cipher for Design Constraints. IEEE FCCM, 338.

[Mukherjee et al., 2017] Mukherjee, M., Matam, R., Shu, L., et al. (2017). Security and Privacy in Fog Computing: Challenges. IEEE Access, 5, 19293–19304.

[Nasir et al., 2022] Nasir, M., et al. (2022). Storage Optimization Strategies for Resource-Constrained IoT Devices. IEEE IoT Journal, 9(15), 13200-13212.

[Ni et al., 2019] Ni, J., et al. (2019). Security and Privacy in Edge Computing. IEEE Network, 33(2), 12-18.

[Pardeshi et al., 2022] Pardeshi, M. S., Sheu, R. -K., & Yuan, S. -M. (2022). Hash-Chain Fog/Edge: A Mode-Based Hash-Chain for Secured Mutual Authentication Protocol Using Zero-Knowledge Proofs in Fog/Edge. Sensors, 22(2), 607.

[Pathmabandu et al., 2023] Pathmabandu, C., et al. (2023). Hardware Constraints and Processing Limits of Edge Nodes. Sensors, 23(8), 3910.

[Pinto & Costa, 2016] Pinto, A., & Costa, R. (2016). Hash-chain-based authentication for IoT. ADCAIJ, 5(4), 43–57.

[Ray, 2018] Ray, P. P. (2018). A survey on Internet of Things architectures. Journal of King Saud University - Computer and Information Sciences, 30(3), 291-319.

[Reddy et al., 2018] Reddy, A. G., Das, A. K., Odelu, V., et al. (2018). A Privacy Preserving three-factor authenticated key agreement protocol for client–server environment. J. Ambient Intell. Humaniz. Comput., 9(4), 1015-1028.

[Sabovic et al., 2023] Sabovic, A., et al. (2023). Energy Efficiency Techniques in Battery-Powered IoT Devices. IEEE Transactions on Green Communications, 7(2), 890-901.

[Seoane et al., 2021] Seoane, V., Garcia-Rubio, C., Almenares, F., & Campo, C. (2021). Performance evaluation of CoAP and MQTT with security support for IoT environments. Computer Networks, 197, 108338.

[Sha et al., 2020] Sha, K., Yang, T. A., Wei, W., & Davari, S. (2020). A survey of edge computing-based designs for IoT security. Digital Communications and Networks, 6(2), 195-202.

[Shah & Venkatesan, 2018] Shah, T., & Venkatesan, S. (2018). Authentication of IoT Device and IoT Server Using Secure Vaults. IEEE TrustCom/BigDataSE, 819-824.

[Shah et al., 2023] Shah, H., Khan, T., Banu, A., & Shah, H. (2023). Symmetric and Asymmetric Encryption Schemes for Internet of Things: A Survey. IJISAE, 11(1), 254–260.

[Simon Francia et al., 2022] Simon Francia, A., Solis-Lastra, J., & Papa Quiroz, E. A. (2022). Elliptic Curves Cryptography for Lightweight Devices in IoT Systems. LNNS, vol 405. Springer.

[Sitton, 2020] Sitton, R. (2020). Virtualization Techniques for Secure Edge Computing. IEEE Software, 37(4), 55-62.

[Smithamol & Rajeswari, 2017] Smithamol, M. B., & Rajeswari, S. (2017). Hybrid Solution for Privacy-Preserving Access Control for Healthcare Data. Adv. Electr. Comput. Eng., 17, 31-38.

[Srinivas et al., 2017] Srinivas, J., Mishra, D., Mukhopadhyay, S., & Kumari, S. (2017). Provably secure biometric based authentication and key agreement protocol for wireless sensor networks. J. Ambient Intell. Humaniz. Comput., 9(4), 875–895.

[Suresha et al., 2023] Suresha, M., et al. (2023). Trust Management Models in Edge Computing Environments. Journal of Network and Computer Applications, 210, 103520.

[Taleb & Toufik, 2024] Taleb, F., & Toufik, L. (2024). Elliptic curves cryptography for lightweight devices in IoT system. Brazilian Journal of Technology, 7, e73725.

[Vishnubhai, 2023] Vishnubhai, P. K. (2023). Design and Implementation of Efficient Elliptic Curve Cryptography on Reconfigurable Platforms. PhD Thesis, Gujarat Technological University.

[Wu et al., 2017] Wu, F., Xu, L., Kumari, S., et al. (2017). A lightweight and anonymous RFID tag authentication protocol with cloud assistance for e-healthcare. J. Ambient Intell. Humaniz. Comput., 9(4), 919–930.

[Xie & Hwang, 2019] Xie, Q., & Hwang, L. (2019). Security enhancement of an anonymous roaming authentication scheme with two-factor security in smart city. Neurocomputing, 356, 112-123.

[Zhi et al., 2020] Zhi, T., Liu, Y., & Wu, J. (2020). A Reputation Value-Based Early Detection Mechanism Against the Consumer-Provider Collusive Attack in Information-Centric IoT. IEEE Access, 8, 38262-38275.

[Zhou et al., 2019] Zhou, W., Jia, Y., Peng, A., Zhang, Y., & Liu, P. (2019). The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges. IEEE IoT Journal, 6(2), 1606-1616.