Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Comunicación Segura y Eficiente de Datos de Dispositivos IoT Utilizando D2OTP

Autor: Adrián Rodríguez Serrano
Tutor(a): Miguel Jiménez Gañán

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Comunicación Segura y Eficiente de Datos de Dispositivos IoT Utilizando D2OTP

Junio 2025

Autor: Adrián Rodríguez Serrano
Tutor: Miguel Jiménez Gañán
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado se centra en el diseño e implementación de un sistema de comunicación seguro y eficiente entre dispositivos IoT, utilizando el protocolo D2OTP, que fue desarrollado específicamente para este proyecto. Su objetivo principal es demostrar que se puede crear una infraestructura de transmisión de datos que sea segura, robusta y ligera, ideal para entornos con recursos computacionales limitados y sin depender de conexiones externas a la nube. La propuesta combina microcontroladores asequibles con software libre, algoritmos criptográficos optimizados y una arquitectura distribuida que separa las funciones de los dispositivos, el servidor central y el gestor de claves.

En la práctica, se ha creado una red de sensores utilizando placas ESP32-C3 equipadas con sensores DHT22, que están conectadas a una red local a través de Wi-Fi. Cada dispositivo puede medir la temperatura y la humedad del ambiente, cifrar esos datos con una clave de un solo uso mediante el protocolo D2OTP, y enviarlos al servidor usando UDP. Además, el sistema incluye una comunicación bidireccional segura que permite confirmar la recepción de los datos, lo que aumenta la fiabilidad de la transmisión. Esta infraestructura también cuenta con un gestor de claves autónomo, encargado de generar, almacenar y distribuir las claves necesarias para cada uno de los componentes del sistema.

El diseño del protocolo D2OTP se basa en el principio del One-Time Pad (OTP) sincronizado mediante contadores, de modo que cada mensaje enviado utiliza una versión de la clave compartida. Para ello, se han creado mecanismos específicos para registrar nuevos dispositivos, actualizar claves, verificar la integridad mediante etiquetas criptográficas (TAGs) y establecer estrategias de recuperación ante desconexiones. Este enfoque ofrece un alto nivel de seguridad sin comprometer la eficiencia, especialmente en comparación con soluciones más pesadas como TLS, que son difíciles de implementar en dispositivos.

La implementación se ha realizado completamente en C++ (tanto el firmware como la librería criptográfica), Python (para el backend del servidor y la gestión de claves) y HTML/JavaScript (para la web de visualización en tiempo real). Este sistema es modular, escalable y fácil de replicar, y todos sus componentes han sido creados desde cero, sin depender de plataformas externas. Se ha puesto un énfasis especial en la eficiencia computacional, el consumo de memoria y la facilidad de depuración. La librería D2OTP, que se integra con `mbedTLS`, permite llevar a cabo operaciones de cifrado y descifrado de manera completamente autónoma en el dispositivo, incluso sin conexión a Internet.

Durante las pruebas, se verificaron todas las funcionalidades del sistema a través de simulaciones de escenarios reales: registro de nuevos dispositivos, envío periódico de mediciones, recepción de confirmaciones y validación del cifrado. También se evaluaron métricas clave como la latencia, el uso de recursos en el ESP32 y la respuesta del sistema en condiciones adversas. Los resultados muestran que el sistema opera de manera estable, segura y con una notable eficiencia.

Este proyecto presenta varias contribuciones importantes: una arquitectura de red IoT segura y práctica que no depende de recursos externos, un protocolo OTP diseñado específicamente para dispositivos embebidos, y una solución integral que funciona a la perfección en entornos educativos, industriales o domésticos. Todo esto se ha desarrollado con un fuerte enfoque en el software libre, la economía y la sostenibilidad técnica del sistema.

Finalmente, se han identificado varias líneas de trabajo futuras que podrían ayudar a escalar el sistema, como la adición de diferentes tipos de sensores. Así que, este TFG no solo cumple con los objetivos que se plantearon al principio, sino que también sienta las bases para el desarrollo de sistemas IoT que sean seguros y accesibles.

Abstract

This Final Degree Project focuses on the design and implementation of a secure and efficient communication system between IoT devices, using the D2OTP protocol, which was developed specifically for this project. Its main objective is to demonstrate that it is possible to create a data transmission infrastructure that is secure, robust and lightweight, ideal for environments with limited computational resources and without relying on external connections to the cloud. The proposal combines affordable microcontrollers with free software, optimised cryptographic algorithms and a distributed architecture that separates the functions of the devices, the central server and the key manager.

In practice, a network of sensors has been created using ESP32-C3 boards equipped with DHT22 sensors, which are connected to a local network via Wi-Fi. Each device can measure the temperature and humidity of the environment, encrypt that data with a one-time key using the D2OTP protocol, and send it to the server using UDP. In addition, the system includes secure two-way communication that allows confirmation of data reception, increasing the reliability of transmission. This infrastructure also has an autonomous key manager, responsible for generating, storing and distributing the keys required for each of the system components.

The design of the D2OTP protocol is based on the principle of the One-Time Pad (OTP) synchronised by counters, so that each message sent uses a version of the shared key. To this end, specific mechanisms have been created to register new devices, update keys, verify integrity using cryptographic tags (MACs) and establish recovery strategies in the event of disconnections. This approach offers a high level of security without compromising efficiency, especially when compared to heavier solutions such as TLS, which are difficult to implement on devices.

The implementation has been done entirely in C++ (both the firmware and the cryptographic library), Python (for the server backend and key management), and HTML/JavaScript (for the real-time visualisation web). This system is modular, scalable, and easy to replicate, and all its components have been created from scratch, without relying on external platforms. Special emphasis has been placed on computational efficiency, memory consumption and ease of debugging. The D2OTP library, which integrates with `mbedTLS`, allows encryption and decryption operations to be carried out completely autonomously on the device, even without an Internet connection.

During testing, all system functionalities were verified through simulations of real-world scenarios: registration of new devices, periodic sending of measurements, reception of confirmations, and encryption validation. Key metrics such as latency, resource usage on the ESP32, and system response under adverse conditions were also evaluated. The results show that the system operates stably, securely, and with remarkable efficiency.

This project makes several important contributions: a secure and practical IoT network architecture that does not depend on external resources, an OTP protocol designed specifically for embedded devices, and a comprehensive solution that works seamlessly in educational, industrial, or domestic environments. All of this has been developed with a strong focus on free software, economy, and the technical sustainability of the system.

Finally, several future lines of work have been identified that could help scale the system, such as adding different types of sensors. Thus, this TFG not only meets the objectives set out at the beginning, but also lays the foundations for the development of secure and accessible IoT systems.

Tabla de contenidos

1. Introducción **1**
1.1. Contexto y motivación 1
1.2. Objetivos del proyecto 2
1.3. Alcance y limitaciones 2
1.4. Metodología de trabajo 3
1.5. Estructura de la memoria 4

2. Fundamentos teóricos y Trabajos previos **7**
2.1. IoT y comunicaciones seguras: desafíos 7
2.2. Protocolos de seguridad existentes en IoT 8
2.3. Protocolo D2OTP: origen y fundamentos 9
2.3.1. Descripción general de D2OTP 9
2.3.2. Criptografía empleada 10
2.4. Tecnologías utilizadas en el proyecto 10
2.4.1. Hardware: ESP32 y sensor DHT22 10
2.4.2. Software embebido: entorno Arduino/ESP-IDF 11
2.4.3. Lenguajes y librerías 12
2.5. Comunicación en redes locales: UDP y concurrencia 12
2.6. Trabajos previos y aportación diferencial del TFG 13

3. Diseño del sistema **15**
3.1. Arquitectura general del sistema 15
3.1.1. Componentes del sistema 15
3.1.2. Diagrama de arquitectura 16
3.2. Modelo de comunicación y protocolo D2OTP adaptado 16
3.2.1. Formato de mensajes 16
3.2.2. Flujo de registro de un nuevo dispositivo 18
3.2.3. Flujo de envío de datos periódicos y confirmaciones 18
3.2.4. Mecanismo de One-Time Pad con contador 18
3.3. Gestión de identidades y claves 19
3.3.1. Estructuras de datos para credenciales 19
3.3.2. Políticas de generación de claves 19
3.3.3. Almacenamiento de claves en dispositivos y en servidor . . . 20
3.3.4. Distribución de claves inicial 20
3.4. Consideraciones de seguridad en el diseño 21
3.4.1. Amenazas consideradas 21

3.4.2. Medidas tomadas 22
3.4.3. Limitaciones conocidas 22
3.5. Diseño de la interfaz web para visualización en tiempo real 23
3.5.1. Arquitectura de la interfaz web 23
3.5.2. Diseño de la interfaz de usuario 23
3.5.3. Integración con el sistema IoT 24

4. Implementación y desarrollo 25
4.1. Firmware del dispositivo IoT 25
4.1.1. Configuración inicial 25
4.1.2. Lógica de registro 26
4.1.3. Lectura de sensores 26
4.1.4. Envío seguro de datos 26
4.1.5. Recepción de confirmaciones 27
4.2. Librería D2OTP 27
4.2.1. Funcionalidades ofrecidas 27
4.2.2. Integración con mbedTLS/OpenSSL 28
4.2.3. Validación de cifrado 28
4.3. Gestor de red 29
4.3.1. Estructura del programa 29
4.3.2. Menú del gestor 29
4.3.3. Comunicación segura con el servidor 30
4.3.4. Gestión de datos persistentes 30
4.4. Servidor central de datos 30
4.4.1. Inicialización 30
4.4.2. Recepción de datos UDP 31
4.4.3. Descifrado y verificación de mensajes 31
4.4.4. Almacenamiento de lecturas 31
4.4.5. Envío de respuesta/ACK a dispositivos 32
4.4.6. Peticiones al gestor en tiempo de ejecución 32
4.5. Aplicación web de monitorización 33
4.5.1. Servidor FastAPI 33
4.5.2. Página web 33
4.5.3. Lógica de actualización en la página 34
4.5.4. Ejemplo de visualización 34
4.6. Integración de componentes y ejecución 35
4.6.1. Puesta en marcha del sistema paso a paso 35
4.6.2. Interacciones entre componentes 36

5. Pruebas y Resultados 39
5.1. Entorno de pruebas 39
5.1.1. Disposición general del sistema 39
5.2. Pruebas funcionales 40
5.2.1. Registro de un dispositivo nuevo 41
5.2.2. Envío de datos periódico 42
5.2.3. Confirmación de servidor encendido 42
5.2.4. Confirmación de recepción 43
5.3. Pruebas de seguridad 43

5.3.1. Confidencialidad 43
5.3.2. Integridad 44
5.4. Rendimiento básico 44
5.4.1. Latencia de comunicación 44
5.4.2. Uso de recursos en el ESP32 45
5.5. Tabla resumen de pruebas 46
5.6. Evaluación general del sistema 47

6. Conclusiones y Líneas futuras **49**
6.1. Grado de consecución de objetivos 49
6.2. Aportaciones del proyecto 50
6.3. Líneas de trabajo futuro 50
6.4. Reflexión personal 51

7. Análisis del impacto **53**
7.1. Impacto social y ético 53
7.2. Impacto medioambiental 54
7.3. Impacto económico y técnico 54

Bibliografía **57**

Anexos **63**

A. Código fuente de la librería D2OTP **63**

B. Código representativo del servidor UDP **67**
B.1. Inicialización del servidor 67
B.2. Análisis de mensajes 67
B.3. Generación y envío del ACK cifrado 69

C. Código representativo del firmware **71**
C.1. Configuración inicial 71
C.2. Solicitud de clave 71
C.3. Lectura del sensor DHT22 72
C.4. Envío cifrado de datos y manejo del ACK 72
C.5. Persistencia de clave y contador en EEPROM 73

D. Código representativo del gestor de red **75**
D.1. Configuración inicial 75
D.2. Recuperación de claves desde fichero JSON 75
D.3. Respuesta al servidor o a la placa 76
D.4. Captura del menú del gestor 76

E. Captura de interfaz **77**

F. Archivo `dispositivos.json` **79**

G. Archivo `config_inicial_servidor.json` **81**

H. Archivo `datos.json` **83**

I. Captura Wireshark **85**

J. Turniting **87**

Índice de figuras

3.1. *Diagrama de arquitectura* 16
3.2. *Estructura mensaje solicitud de clave* 17
3.3. *Estructura mensaje respuesta con clave cifrada* 17
3.4. *Estructura mensaje envío medida* 17
3.5. *Estructura de los datos de un dispositivo en el gestor* 19
3.6. *Estructura de la configuración inicial del servidor* 20
3.7. *Estructura del fichero de claves del servidor* 20

4.1. *Estructura del fichero* `datos.json` *del servidor* 32

5.1. Mensaje consola servidor al recibir datos 42
5.2. Confirmación de servidor encendido 43
5.3. Confirmación ACK 43
5.4. Mensaje error con mensaje alterado 44
5.5. Uso de recursos de la ESP32 45

Capítulo 1

Introducción

1.1. Contexto y motivación

El Internet de las Cosas (IoT, por sus siglas en inglés) ha cambiado la forma en la que interactuamos con el mundo, permitiendo la conexión de dispositivos físicos a través de redes para recopilar y compartir datos. El IoT ha encontrado aplicaciones en diversos sectores, desde dispositivos domésticos inteligentes hasta sensores industriales, mejorando la eficiencia operativa y la toma de decisiones basadas en datos [1].

Sin embargo, esta nueva conectividad masiva ha introducido muchos desafíos importantes en términos de seguridad. Muchos dispositivos IoT fueron programados sin las adecuadas consideraciones de seguridad, lo que los hace muy vulnerables a ataques cibernéticos. Según Statista, en 2024 había en funcionamiento 21 mil millones de dispositivos IoT en todo el mundo, cifra que se espera que cada año aumente hasta los 36.500 millones el año 2029 [2].

La gran variedad de dispositivos y protocolos, unida a la falta en estándares de seguridad, ha llevado a una superficie de ataque ampliada, donde los atacantes pueden aprovechar esas vulnerabilidades para acceder a las redes, robar datos o, incluso, interrumpir servicios. La autenticación de dispositivos y la gestión de las claves de éstos siguen siendo áreas críticas que requieren soluciones inmediatas robustas y escalables [3] [4].

En este contexto, el protocolo D2OTP (Drive-to-One-Time Password) nace como una solución prometedora para mejorar la seguridad en las comunicaciones entre dispositivos IoT. Este protocolo reduce el riesgo de ataques de repetición y suplantación de identidad gracias a sus contraseñas de un solo uso para autenticar a los dispositivos y cifrar las comunicaciones.

Este proyecto se centra en la implementación de una solución segura y eficiente para la comunicación de dispositivos IoT utilizando D2OTP, abordando los desafíos mencionados y contribuyendo a la mejora de la seguridad en entornos IoT.

1.2. Objetivos del proyecto

El objetivo principal de este Trabajo de Fin de Grado es diseñar, implementar y evaluar una solución segura y eficiente basada en el protocolo D2OTP para el problema actual en la comunicación entre dispositivos IoT. Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos:

1. **Análisis de requisitos y estudio del estado del arte**: Investigar las necesidades actuales de seguridad en los entornos IoT y analizar las soluciones existentes, prestando especial atención a los protocolos de autenticación y cifrado.
2. **Diseño de la arquitectura del sistema**: Definir una arquitectura que integre el protocolo D2OTP entre dispositivos IoT, teniendo en cuenta aspectos importantes como la gestión de claves, la interoperabilidad y la escalabilidad.
3. **Implementación de la solución**: Desarrollar un software para permitir la comunicación segura y eficiente entre dispositivos IoT utilizando D2OTP, incluyendo, entre otros aspectos, la generación y validación de contraseñas de un solo uso.
4. **Evaluación y validación**: Realizar pruebas para evaluar la seguridad, la eficiencia y el rendimiento de la solución creada, comparándola, en términos de consumo de recursos, latencia y resistencia a ataques, con otras soluciones ya existentes.
5. **Documentación y análisis de resultados**: Documentar el proceso de desarrollo junto con las pruebas, los resultados obtenidos y las lecciones aprendidas, proporcionando una base para futuras mejoras y adaptaciones de la solución.

Al término del cumplimiento de estos objetivos se espera contribuir al futuro desarrollo de soluciones mejoradas aún más seguras y eficientes en el ámbito del IOT, afrontando desafíos decisivos y promoviendo a la adopción de prácticas de seguridad robustas en dispositivos conectados.

1.3. Alcance y limitaciones

Este Trabajo de Fin de Grado se centra como hemos mencionado en el diseño, implementación y evaluación de una solución de comunicación segura y eficiente entre dispositivos IoT mediante D2OTP. El proyecto comprende desde la conceptualización de la arquitectura hasta la comprobación de su funcionamiento en un entorno controlado.

Alcance

- **Diseño de la arquitectura del sistema**: Se define una estructura que integra el protocolo D2OTP en la comunicación entre dispositivos IoT, consi-

derando aspectos importantes como, entre otras, la interoperabilidad y la gestión de claves.

- **Desarrollo de prototipos funcionales**: Se utilizan dispositivos IoT que utilizan este protocolo para establecer comunicaciones seguras entre ellos, incluyendo, por ejemplo, la generación y validación de contraseñas de un solo uso.
- **Evaluación del sistema**: Se realizan pruebas controladas para analizar el rendimiento, la eficiencia y la seguridad de la solución desarrollada, utilizando métricas como la latencia y el consumo de recursos.
- **Documentación detallada**: Se elabora una memoria en la que se describe el desarrollo llevado a cabo, los resultados obtenidos y lecciones aprendidas, creando una base para futuras investigaciones.

Limitaciones

- **Entorno de pruebas controlado**: Las evaluaciones se realizan en un entorno controlado y simulado, lo que no refleja todas las condiciones de un entorno real de producción.
- **Escalabilidad**: Aunque la escalabilidad es una de las partes consideradas en el diseño, esta implementación se limita a un número muy reducido de dispositivos, por lo que no se llega a evaluar su comportamiento en redes de gran escala con gran número de dispositivos.
- **Compatibilidad con otros protocolos**: El proyecto está centrado únicamente en el protocolo D2OTP, sin llegar a estudiar la integración o comparación con otros protocolos de seguridad ya existentes en el ámbito IoT.
- **Restricciones de tiempo y recursos**: Al tratarse de un proyecto académico, el proyecto está sujeto a limitaciones de recursos y de tiempo, lo que puede afectar a la profundización de ciertas áreas de investigación.

1.4. Metodología de trabajo

La metodología adoptada en este proyecto combina unos enfoques cualitativos y cuantitativos para abordar íntegramente todos los objetivos que se han planteado. Se estructura en las siguientes fases [5]:

1. Revisión bibliográfica

Se realiza una investigación en profundidad de distintas fuentes tanto técnicas como académicas para comprender el funcionamiento del protocolo D2OTP y el estado actual de la seguridad en IoT. Esta fase proporciona el marco teórico necesario para el desarrollo del proyecto.

2. Diseño del sistema

Teniendo en cuenta la información recopilada, se diseña una arquitectura que integra el protocolo D2OTP en la comunicación entre distintos dispositivos IoT. Se consideran aspectos como la autenticación, la eficiencia en la transmisión de datos y la gestión de claves de manera segura.

3. Desarrollo e implementación

Se utilizan prototipos funcionales de dispositivos IoT a los que se les implementa D2OTP para establecer comunicaciones seguras y eficientes. Esta fase incluye la programación de firmware, la configuración de redes y la implementación de mecanismos de autenticación.

4. Pruebas y evaluación

Se realizan pruebas controladas para evaluar tanto la seguridad como el rendimiento y la eficiencia de la solución implementada. Se utilizan métricas como el consumo de recursos y la latencia para analizar el comportamiento del sistema.

5. Documentación y análisis

Se elabora la memoria donde se documenta todo el proceso de desarrollo, incluyendo los desafíos afrontados y las soluciones adoptadas. También se incluyen y se analizan los resultados obtenidos y se extraen conclusiones que pueden servir de base de cara a futuras investigaciones en el ámbito de la seguridad en IoT.

Esta metodología permite abordar de manera sistemática y rigurosa los objetivos planteados para el proyecto, asegurando la validez y fiabilidad de los resultados obtenidos.

1.5. Estructura de la memoria

Esta memoria se ha estructurado de una forma coherente y lógica para reflajar, con claridad, el proceso seguido durante el desarrollo del proyecto, facilitando así la comprensión de los objetivos, la metodología seguida, la implementación y los resultados conseguidos. Cada capítulo se centra en aspectos concretos del trabajo, lo que permite al lector seguir el hilo conductor de la investigación y comprender las decisiones tomadas en cada etapa.

A continuación se describen, brevemente, los contenidos de cada capítulo:

- **Capítulo 1: Introducción**
  Se presenta el contexto y la motivación del proyecto, los objetivos establecidos, el alcance y las limitaciones que se han identificado, la metodología de trabajo adoptada y la estructura general de la memoria.
- **Capítulo 2: Fundamentos teóricos y Trabajos previos**
  Se lleva a cabo una revisión del estado actual en el ámbito de la seguridad

en el Internet de las Cosas (IoT), donde se analizan las principales amenazas y soluciones existentes. Además, se profundiza en los fundamentos teóricos del protocolo D2OTP y su aplicabilidad en entornos IoT.

- **Capítulo 3: Diseño del sistema**
  Se detalla la arquitectura de la solución propuesta, incluyendo la descripción de todos los componentes hardware y software, así como la integración del protocolo D2OTP en la comunicación entre dispositivos IoT.
- **Capítulo 4: Implementación y desarrollo**
  Se describe cuál es el proceso para implementar la solución, tratando aspectos técnicos relacionados con la programación de los dispositivos, la configuración de la red y la integración de todos los módulos del sistema.
- **Capítulo 5: Pruebas y Resultados**
  Se presentan los resultados obtenidos tras la realización de todas las pruebas realizadas, valorando tanto la seguridad como el rendimiento y eficiencia de la solución propuesta. Además, se incluyen comparaciones con otras soluciones donde se discuten las ventajas y las limitaciones observadas.
- **Capítulo 6: Conclusiones y Líneas futuras**
  Se resumen las conclusiones del trabajo realizado, subrayando las contribuciones y los logros conseguidos. Asimismo, se proponen posibles futuras líneas de trabajo que podrían ampliar, o incluso mejorar, la solución desarrollada.
- **Capítulo 7: Análisis de impacto**
  Se analiza el impacto del proyecto desde una perspectiva amplia, considerando sus efectos en los ámbitos social y ético, medioambiental y económico-técnico. Esta sección tiene como objetivo situar el proyecto dentro de un marco de sostenibilidad, responsabilidad social y viabilidad tecnológica, identificando los beneficios, los riesgos y las oportunidades que surgen de su implementación en situaciones reales.
- **Bibliografía**
  Se recopilan todas las fuentes consultadas y citadas a lo largo de la memoria, siguiendo un formato de citación adecuado y coherente.
- **Anexos**
  Se incluyen documentos complementarios, como fragmentos de código y esquemas, que aportan a la memoria información adicional relevante.

Esta estructura se ha diseñado para proporcionar, de una manera ordenada, una visión clara y detallada del proyecto permitiendo así una comprensión profunda de todos los aspectos teóricos y técnicos abordados.

Capítulo 2

Fundamentos teóricos y Trabajos previos

2.1. IoT y comunicaciones seguras: desafíos

El Internet de las Cosas (IoT, por sus siglas en inglés) ha crecido de manera exponencial en los últimos años, integrándose en diferentes áreas como la industria, el transporte, la salud y los hogares inteligentes. Este crecimiento ha traído consigo, aparte de muchos beneficios, nuevas vulnerabilidades y desafíos en el ámbito de la seguridad [1].

Uno de los grandes retos que enfrentamos es la débil autenticación de los dispositivos IoT. Muchos de estos aparatos utilizan contraseñas por defecto o, directamente, no cuentan con mecanismos de autenticación robustos, lo que los hace muy vulnerables a accesos no controlados. Además, la falta de cifrado en las comunicaciones es una preocupación importante ya que, como muchos de estos dispositivos transmiten mensajes en claro, se expone la información sensible a posibles interceptaciones [6].

La vulnerabilidad del firmware y del software es un tema realmente crítico. La rapidez en los ciclos de desarrollo y la presión por mantener un bajo presupuesto hacen que muchos dispositivos IoT lleguen al mercado sin haber pasado unas pruebas de seguridad exhaustivas, dejando puertas abiertas a posibles ataques. Además, la dificultad para actualizar y parchear estos dispositivos no hace más que empeorar la situación, ya que muchos de ellos no cuentan con mecanismos eficientes para recibir actualizaciones de seguridad [7].

La segmentación insuficiente de la red también conlleva riesgos. Si no existe la separación adecuada entre los dispositivos IoT y otros sistemas críticos, una vulnerabilidad en un solo dispositivo puede comprometer toda la red [7] [8].

Estos desafíos resaltan la importancia de crear o implementar soluciones de seguridad que sea específicas para el entorno del IoT, teniendo en cuenta las limitaciones y características únicas de estos dispositivos.

2.2. Protocolos de seguridad existentes en IoT

Para enfrentar los retos de seguridad en el IoT, se han ido desarrollando varios protocolos que tienen como objetivo proteger la confidencialidad, la integridad y la autenticación de las comunicaciones entre dispositivos. A continuación, se presentan algunos de los más relevantes:

TLS/DTLS (Transport Layer Security / Datagram TLS)

TLS es un protocolo muy utilizado para asegurar las comunicaciones en redes que funcionan con TCP, proporcionando cifrado y autenticación de extremo a extremo. Por otro lado, DTLS adapta TLS para funcionar sobre protocolos de transporte que no son orientados a conexión, como UDP, lo que lo hace más adecuado para aplicaciones en tiempo real y dispositivos con recursos limitados [9].

IPsec (Internet Protocol Security)

IPsec es un conjunto de protocolos que asegura la red a través del cifrado y autenticación de paquetes IP. Es especialmente seguro para crear redes privadas virtuales (VPN) y para proteger las comunicaciones entre redes o dispositivos que están a distancia [10].

MQTT (Message Queuing Telemetry Transport)

MQTT es un protocolo de mensajería ligero que se basa en el modelo de publicador / subscriptor muy utilizado en aplicaciones de IoT. Aunque por sí solo no ofrece mecanismos de seguridad, se puede combinar con TLS/SSL para cifrar las comunicaciones y asegurar la autenticación de los dispositivos [11].

CoAP (Constrained Application Protocol)

CoAP es un protocolo creado especialmente para dispositivos con recursos limitados que permite una comunicación eficiente en redes IoT. Al igual que MQTT, se puede completar con mecanismos de seguridad como DTLS para proteger las comunicaciones [12].

OSCORE (Object Security for Constrained RESTful Environments)

OSCORE es un protocolo creado para ofrecer seguridad de extremo a extremo en entornos de IoT con recursos limitados. Se fundamenta en el protocolo CoAP y emplea una criptografía basada en objetos para proteger los mensajes, lo que lo hace eficiente en el uso de recursos y perfecto para dispositivos con capacidades restringidas [13].

Cada uno de estos protocolos tiene sus propias ventajas y desventajas, dependiendo del contexto en el que se apliquen, los recursos disponibles y los requisitos de seguridad específicos. Elegir el protocolo correcto es fundamental para asegurar que las comunicaciones en entornos IoT sean seguras y eficientes.

2.3. Protocolo D2OTP: origen y fundamentos

El protocolo D2OTP (Drive-to-One-Time Password) surge como una respuesta innovadora a las crecientes demandas de seguridad en las comunicaciones entre dispositivos IoT. Desarrollado y registrado bajo la patente internacional WO2022018310, D2OTP se presenta como una solución que asegura la integridad, la confidencialidad y la autenticación mutua en las comunicaciones entre dispositivos, sin depender de infraestructuras centralizadas ni de certificados digitales [14].

La principal razón por la que se creó D2OTP es para superar las limitaciones de los protocolos de seguridad tradicionales en el mundo del IoT, donde hay una gran variedad de dispositivos que suelen tener recursos limitados. Al utilizar un sistema que genera contraseñas de un solo uso (OTP) para cada mensaje, D2OTP añade una capa extra de seguridad que ayuda a prevenir ataques comunes como el phishing, los ataques de intermediario (Man-in-the-Middle) y la reutilización de claves [14].

Además, D2OTP se alinea con las normativas y estándares internacionales de seguridad, como las directrices de NIST para la ciberseguridad en IoT, la Ley de Ciberresiliencia de la Unión Europea y el US Cyber Trust Mark, facilitando su adopción en diversos sectores industriales y comerciales [15].

2.3.1. Descripción general de D2OTP

D2OTP es un protocolo de seguridad que funciona en la capa de aplicación, diseñado para proteger las comunicaciones entre dispositivos IoT. Lo hace generando y utilizando contraseñas de un solo uso (OTP) para cada mensaje que se intercambia. Estas contraseñas se generan de forma autónoma en cada dispositivo, empleando una clave base que se establece durante el emparejamiento inicial y un valor aleatorio único para cada comunicación [14].

El funcionamiento de D2OTP se basa en los siguientes principios:

- **Cifrado de extremo a extremo**: Cada mensaje se cifra utilizando una clave OTP que es única, asegurando que únicamente el destinatario previsto sea el que pueda descifrar y acceder al contenido [14].
- **Autenticación mutua**: Ambos dispositivos que participan en la comunicación se aseguran de verificar la identidad del otro, garantizando que la información solo se intercambia entre entidades autorizadas.
- **No repudio**: Como se utilizan claves OTP únicas y autenticación mutua, se garantiza que ninguna de las partes pueda rechazar su participación en la comunicación.
- **Independencia de la red**: D2OTP no depende ni de las infraestructuras centralizadas ni de las características de la red subyacente, lo que permite su implementación en entornos con conectividad limitada o intermitente.

Esta arquitectura descentralizada y ligera hace que D2OTP sea una opción ideal

para dispositivos IoT que tienen recursos limitados, como sensores, actuadores y microcontroladores, donde la eficiencia y la seguridad son aspectos clave.

2.3.2. Criptografía empleada

D2OTP utiliza técnicas de criptografía simétrica ligera para asegurar las comunicaciones sin sacrificar el rendimiento de los dispositivos IoT. En particular, hace uso de algoritmos de cifrado simétrico eficientes, como el Estándar de Encriptación Avanzada (AES), que proporciona un alto nivel de seguridad mientras consume pocos recursos computacionales [16].

La creación de contraseñas de un solo uso (OTP) se lleva a cabo utilizando funciones hash criptográficas seguras. Estas funciones utilizan la clave base compartida que se estableció en el emparejamiento inicial combinada con un valor único aleatorio para cada mensaje. Este método asegura que cada clave OTP sea única e irrepetible, lo que ayuda a prevenir ataques de repetición y suplantación de identidad.

Además, al eliminar la necesidad de certificados digitales y de infraestructuras de clave pública (PKI), D2OTP simplifica la gestión de claves en entornos IoT, lo que reduce tanto la complejidad como los costos. Esto hace que su implementación sea mucho más accesible en una amplia gama de dispositivos y aplicaciones [17].

En resumen, la criptografía simétrica ligera combinada con la generación automática de claves OTP hace de D2OTP una solución tanto eficaz como eficiente para asegurar las comunicaciones en el mundo del IoT.

2.4. Tecnologías utilizadas en el proyecto

La implementación del sistema de comuniciación segura que utiliza el protocolo D2OTP se ha realizado mediante una mezcla de tecnologías de hardware y software, lo que permite una integración eficiente y segura en entornos de IoT. A continuación, se describen los componentes y herramientas elegidos para el desarrollo del proyecto.

2.4.1. Hardware: ESP32 y sensor DHT22

ESP32

El microcontrolador ESP32, desarrollado por Espressif Systems, es una opción económica y de bajo consumo energético que combina conectividad Wi-Fi y Bluetooth, lo que lo convierte en una excelente elección para aplicaciones de IoT. Entre sus características más destacadas se incluyen:

- Procesador de doble núcleo Xtensa LX6 de 32 bits, con una frecuencia de hasta 240 MHz.
- 520 KiB de SRAM y soporte para memoria flash externa.

- Conectividad Wi-Fi 802.11 b/g/n y Bluetooth v4.2 BR/EDR y BLE.
- Amplia gama de interfaces periféricas, incluyendo UART, SPI, I^2C, ADC, DAC y PWM.
- Capacidades de seguridad avanzadas, como arranque seguro, cifrado de flash y aceleradores criptográficos para AES, SHA-2, RSA y ECC.

Estas características hacen del ESP32 una plataforma increíblemente versátil y poderosa para desarrollar dispositivos IoT que sean tanto seguros como eficientes [18].

Sensor DHT22

El sensor DHT22 (también conocido como AM2302) es un sensor digital de temperatura y humedad que ofrece una alta precisión y estabilidad a largo plazo. Entre sus especificaciones encontramos:

- Rango de medición de temperatura: -40°C a 80°C, con una precisión de ±0,5°C.
- Rango de medición de humedad: 0 % a 100 % RH, con una precisión de ±2 % RH.
- Voltaje de operación: 3.3V a 6V DC.
- Salida de señal digital a través de un bus de un solo hilo.

Su bajo consumo de energía y su fácil integración lo hacen ideal para aplicaciones de monitoreo ambiental en sistemas IoT [19].

2.4.2. Software embebido: entorno Arduino/ESP-IDF

Arduino IDE

El entorno de desarrollo Arduino IDE es una plataforma sencilla y accesible para la programación de microcontroladores, como el ESP32. Su amplia comunidad y la gran cantidad de bibliotecas disponibles, hace sencillo el desarrollo rápido de prototipos y la implementación de funciones en dispositivos IoT [20].

ESP-IDF

El Espressif IoT Development Frameword (ESP-IDF) es el entorno de desarrollo oficial para los microcontroladores ESP32. Proporciona un conjunto de bibliotecas y herramientas que facilitan la creación de aplicaciones IoT avanzadas, permitiendo un control minucioso del hardware y la implementación de características de seguridad sólidas. ESP-IDF es especialmente útil para proyectos que necesitan un nivel alto de personalización y optimización del rendimiento [21].

2.4.3. Lenguajes y librerías

Lenguajes de programación

El proyecto se ha desarrollado principalmente con el lenguaje de programación C++, el cual es totalmente compatible tanto con el entono de Arduino como con ESP-IDF. Estos lenguajes ofrecen un control preciso del hardware y una gestión eficiente de los recursos, lo cual es fundamental en dispositivos con capacidades limitadas, como los que se utilizan en entornos IoT.

Librerías utilizadas

Para facilitar la implementación de las funcionalidades específicas de este proyecto y garantizar la compatibilidad con los componentes seleccionados, se han utilizado varias librerías, entre las que destacan [22]:

- **DHT**: Proporciona funciones para la lectura de datos de temperatura y humedad desde el sensor DHT22.
- **WiFi**: Permite la conexión del ESP32 a redes Wi-Fi, facilitando la transmisión de datos a través de internet.
- **ESP-IDF components**: Incluye módulos para la gestión de tareas, control de periféricos, manejo de eventos y funciones de seguridad, entre otros.
- **EEPROM**: Permite guardar en la memoria permanente (memoria EEPROM) los datos y almacenarlos incluso tras desconectar la alimentación de la placa del microprocesador.

La fusión de estas tecnologías de hardware y software ha dado lugar a una solución de comunicación segura y eficiente para los dispositivos IoT, cumpliendo con todos los objetivos y requisitos que se han establecido en este proyecto.

2.5. Comunicación en redes locales: UDP y concurrencia

En el mundo del Internet de las Cosas (IoT), es crucial que los dispositivos se comuniquen de manera eficiente y confiable. Aquí es donde el protocolo UDP (User Datagram Protocol) brilla como una opción ideal, gracias a su simplicidad y a su bajo consumo de recursos. UDP es un protocolo de transporte que permite enviar datagramas sin tener que establecer una conexión previa, lo que lo convierte en la elección perfecta para aplicaciones que necesitan baja latencia y pueden tolerar la pérdida de algunos paquetes [23].

Una de las grandes ventajas de UDP en redes locales es su habilidad para llevar a cabo transmisiones multicast y broadcast. Esto significa que un dispositivo puede enviar un mensaje a varios receptores al mismo tiempo. Es especialmente útil en situaciones donde se necesita descubrir dispositivos en la red o enviar actualizaciones a múltiples nodos de una sola vez [24].

Sin embargo, la naturaleza sin conexión de UDP trae consigo algunos desafíos, como la falta de garantías en la entrega de paquetes y la posibilidad de que los

mensajes lleguen desordenados. Para abordar estos inconvenientes es bastante común implementar mecanismos de control en la capa de aplicación, como la verificación de la integridad de los datos y la gestión de retransmisiones en caso de que se pierdan los paquetes [23].

En cuanto a la concurrencia, en sistemas IoT, es bastante común que varios dispositivos intenten comunicarse al mismo tiempo. Esto puede provocar colisiones y congestamientos en la red. Para enfrentar este reto, se han propuesto varias estrategias, como el uso de protocolos de acceso al medio eficientes y la implementación de técnicas de programación concurrente que ayuden a gestionar múltiples flujos de manera efectiva.

En este proyecto, se ha decidido utilizar UDP para la comunicación entre dispositivos, gracias a su bajo consumo de recursos y su idoneidad en entornos con limitaciones. Además, se han incorporado mecanismos en la capa de aplicación para asegurar que los mensajes lleguen íntegros y en el orden correcto, así como estrategias de programación concurrente para manejar, de una manera eficiente, múltiples flujos de datos.

2.6. Trabajos previos y aportación diferencial del TFG

La seguridad en las comunicaciones IoT ha sido un tema de gran interés en los últimos años, con numerosos estudios e investigaciones dedicadas a ello. Se han propuesto diferentes protocolos y enfoques para enfrentar los desafíos que surgen en estos entornos, como la autenticación de dispositivos, la protección de la confidencialidad de los datos y la garantía de la integridad de las comunicaciones.

Entre los trabajos previos más destacados, se encuentran aquellos que han investigado el uso de protocolos de seguridad ligeros como DTLS (Datagram Transport Layer Security) y OSCORE (Object Security for Constrained RESTful Environments). Estos protocolos ofrecen mecanismos de cifrado y autenticación que son ideales para dispositivos con recursos limitados [9] [13].

El protocolo D2OTP (Drive-to-One-Time Password) se presenta como una solución innovadora en este contexto, ya que ofrece un mecanismo de autenticación mutua y cifrado de extremo a extremo basado en contraseñas de un solo uso, sin necesidad de depender de infraestructuras de clave pública (PKI). Sin embargo, la implementación práctica de D2OTP en dispositivos IoT y su evaluación en entornos reales todavía no ha sido explorada en profundidad [25].

La principal aportación de este Trabajo de Fin de Grado se centra en la implementación y evaluación de una solución de comunicación segura basada en D2OTP para dispositivos IoT reales, utilizando microcontroladores ESP32 y sensores DHT22. Se ha creado un sistema completo que integra el protocolo D2OTP en la comunicación entre dispositivos. También se han establecido mecanismos para la gestión de claves y se ha evaluado tanto el rendimiento como la seguridad de la solución en un entorno controlado.

Además, se han explorado aspectos prácticos sobre cómo implementar D2OTP

en dispositivos con recursos limitados, centrándose en la optimización del uso de memoria y en una gestión eficiente de la energía. Los resultados obtenidos ofrecen una perspectiva valiosa sobre la viabilidad y efectividad de D2OTP en entornos IoT, estableciendo así las bases para futuras investigaciones y desarrollos en este campo.

Capítulo 3

Diseño del sistema

3.1. Arquitectura general del sistema

El sistema desarrollado en este proyecto tiene como objetivo principal crear una comunicación segura y eficiente entre dispositivos IoT. Para lograrlo, utilizamos el protocolo D2OTP, que asegura la confidencialidad, integridad y autenticación de los datos que se transmiten. La arquitectura propuesta se basa en una estructura modular, lo que facilita la escalabilidad, el mantenimiento y la integración de nuevos componentes en el futuro.

3.1.1. Componentes del sistema

La arquitectura del sistema se compone de los siguientes elementos principales:

- **Dispositivos IoT (nodos sensores)**: Estos dispositivos son los mencionados microcontroladores ESP32, que brindan conectividad Wi-Fi y un buen poder de procesamiento para aplicaciones de IoT. Cada nodo sensor cuenta un sensor DHT22 para la medición de la temperatura y la humedad ambiental. Los datos recopilados se cifran utilizando el protocolo D2OTP antes de ser transmitidos al servidor.
- **Servidor Central**: Es el encargado de recibir, descifrar y almacenar los datos enviados por los nodos sensores. Además, proporciona una interfaz web para la visualización en tiempo real de las mediciones.
- **Gestor de red**: Es el encargado de almacenar las claves tanto del servidor como de los dispositivos, y enviárselas a estos cuando sea necesario. Es, también, dónde se registran todos los dispositivos.
- **Interfaz web**: Permite ver a tiempo real los datos que recogen los sensores, lo que hace más fácil el monitoreo y análisis de las condiciones ambientales.
- **Red de comunicación**: La comunicación entre los nodos sensores y el servidor se lleva a cabo a través de un red Wi-Fi local, empleando el protocolo UDP para transmitir los datos cifrados. Esta elección se hace para reducir al mínimo la latencia y el overhead en la comunicación.

3.1.2. Diagrama de arquitectura

El diagrama de arquitectura del sistema ilustra cómo interactúan los distintos componentes descritos anteriormente. En este diagrama, se puede ver como los nodos sensores recogen datos del entorno, los cifran usando D2OTP y los envían al servidor central a través de la red Wi-Fi. Luego, el servidor descifra esos datos, los almacena, y los muestra en la interfaz web para que los usuarios puedan visualizarlos.

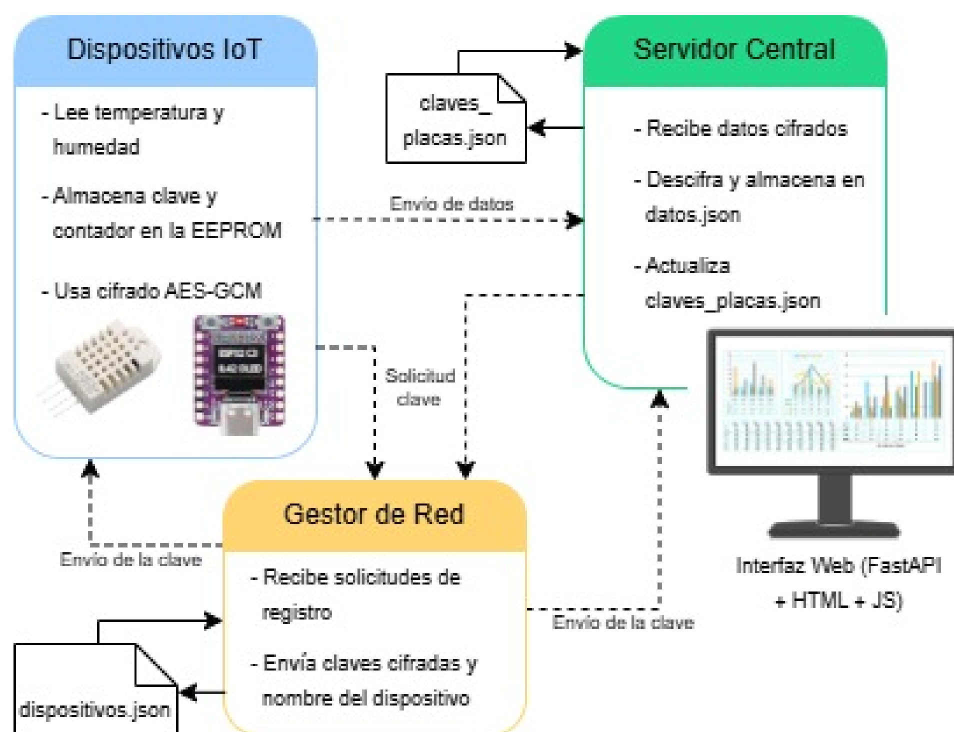


Figura 3.1: Diagrama de arquitectura

3.2. Modelo de comunicación y protocolo D2OTP adaptado

El sistema implementado en este proyecto se basa en una arquitectura cliente-servidor, completada por un gestor de claves externo. Este modelo asegura la integración del protocolo D2OTP, que está diseñado específicamente para entornos IoT, a través de la aplicación de tres actores clave: los dispositivos (las placas IoT conectadas a los sensores), el servidor central y el gestor de claves. La comunicación entre ellos sigue flujos bien definidos y se organiza en mensajes JSON que contienen datos cifrado, metadatos y solicitudes de clave.

3.2.1. Formato de mensajes

Los mensajes que se interceptan están estructurados en formato JSON, lo que facilita su análisis y permite una interacción más fluida entre los distintos com-

ponentes. Dependiendo de la función del mensaje, la estructura puede cambiar. A continuación, se presentan los tipos principales:

- **Detección de servidor disponible**: El dispositivo IoT, antes de enviar cualquier medición al servidor, le envía un mensaje (*ping*) y espera la respuesta del servidor. Si este le responde *pong* quiere decir que el servidor está encendido a la espera de recibir datos.
- **Solicitud de clave**: Este mensaje se envía cuando bien el servidor o bien el dispositivo le piden al gestor su clave, ya que es la primera vez que se encienden o, por algún motivo, la han perdido. Se detalla en el ejemplo la solicitud del servidor al gestor.

```
{
  "tipo": "solicitud_clave_servidor",
  "id_origen": "ID_SERVIDOR",
  "nonce": "NONCE"
}
```

Figura 3.2: Estructura mensaje solicitud de clave

- **Respuesta con clave cifrada**: El gestor responde al emisor del mensaje de solicitud enviando la clave cifrada.

```
{
  "clave_cifrada": "CLAVE_CIFRADA_HEX",
  "tag": "TAG_GENERADO"
}
```

Figura 3.3: Estructura mensaje respuesta con clave cifrada

- **Mensaje del dispositivo al servidor**: El dispositivo, después de que el sensor recoja la medición ambiental, cifra los datos y envía el siguiente mensaje:

```
{
  "id": "ID_DISPOSITIVO",
  "contador": "CONTADOR_ALMACENADO_EEPROM",
  "payload": "DATOS_CIFRADOS",
  "tag": "TAG_GENERADO"
}
```

Figura 3.4: Estructura mensaje envío medida

- **Confirmación servidor**: El servidor le confirma al dispositivo que ha recibido su medición contestándole con un *OK*.

3.2.2. Flujo de registro de un nuevo dispositivo

El proceso de alta de un nuevo dispositivo lo inicia el usuario desde el gestor:

1. El usuario introduce en el menú del gestor el ID del nuevo dispositivo a añadir y el nombre con el que quiere que sea visible.
2. El gestor genera, de manera aleatoria, una clave de 32 bytes (64 caracteres hexadecimales) (`Cs` para el servidor o `Cpx` para las placas), y se la asocia al ID del dispositivo.
3. El gestor almacena toda esta información (nombre, ID y clave) en un archivo llamado `dispositivos.json`.

3.2.3. Flujo de envío de datos periódicos y confirmaciones

Este es el flujo que se repite de forma periódica y continua durante la operación normal del sistema:

1. El dispositivo recopila una medición (temperatura y humedad) a través del sensor.
2. Se cifra el mensaje utilizando el algoritmo OTP con el contador interno del dispositivo y su clave privada `Cpx` almacenada en la EEPROM.
3. El dispositivo envía al servidor el JSON con el mensaje cifrado.
4. Si el servidor no tiene la clave del dispositivo IoT, se la solicita al gestor con su clave privada (`Cs`).
5. El gestor devuelve `Cpx` cifrada con `Cs` y el servidor la descifra y la guarda, junto con el nombre del dispositivo que también iba en el mensaje, en `claves_placas.json`.
6. El servidor ya puede descifrar el mensaje recibido del dispositivo con la clave `Cpx`, comprueba la validez del nonce y el tag, y almacena el dato en `datos.json`.
7. El servidor envía un ACK como respuesta para que el dispositivo sepa que el dato fue correctamente almacenado.

Este flujo asegura una seguridad criptográfica fuerte.

3.2.4. Mecanismo de One-Time Pad con contador

El cifrado se basa en una versión adaptada de OTP, donde el método de cifrado es:

```
mbedtls_gcm_crypt_and_tag(&ctx, MBEDTLS_GCM_ENCRYPT, input_len,
    iv, 12, nullptr, 0, input, output, tag_len, tag);
```

Esta construcción garantiza que:

- Cada mensaje esté cifrado con una clave distinta gracias al contador (IV) que se incrementad después de cada envío.

- El servidor pueda regenerar la clave si dispone del valor del contador y de la clave privada.
- La integridad se valide con un tag generado mediante AES-GCM.

El uso del contador como elemento de unicidad evita ataques de repetición, al igual que la validación cruzada del servidor contra mensajes previamente recibidos.

3.3. Gestión de identidades y claves

La gestión de entidades y claves es un aspecto fundamental en la arquitectura del sistema desarrollado, especialmente en entornos IoT donde la seguridad y la eficiencia son cruciales. En este proyecto, se ha implementado un enfoque centralizado a través de un gestor de claves, que funciona como autoridad de confianza para la generación, almacenamiento y distribución de las claves criptográficas. Este gestor no solo facilita la administración de identidades, sino que también asegura la integridad y confidencialidad de las comunicaciones entre el servidor y los dispositivos.

3.3.1. Estructuras de datos para credenciales

Para manejar y representar las credenciales de los dispositivos, se ha diseñado una estructura de datos que es tanto eficiente como segura. Cada dispositivo cuenta con un identificador único (`ID`) y se le asigna una clave privada (`Cx`) que se genera de manera segura. La estructura de datos que utilizamos en el gestor de claves (almacenadas en `dispositivos.json`) es la siguiente:

```
{
  "ID_DISPOSITIVO"{
    "clave": "CLAVE_DISPOSITIVO",
    "nombre": "NOMBRE_VISIBLE_DISPOSITIVO"
  }
}
```

Figura 3.5: Estructura de los datos de un dispositivo en el gestor

Esta estructura hace que la gestión de las credenciales sea mucho más eficiente, simplificando operaciones como buscar, actualizar y revocar claves.

3.3.2. Políticas de generación de claves

La generación de claves se lleva a cabo bajo rigurosas políticas para asegurar tanto su seguridad como su singularidad. Las claves privadas de los los dispositivos y del servidor se generan mediante un generador de números aleatorios que es criptográficamente seguro, lo que garantiza que cada clave sea única y no predecible. Además, se aplican las siguientes políticas:

- **Longitud de clave fuerte**: Las claves tienen una longitud de 256 bits, es decir, 32 bytes, lo que proporciona un alto nivel de seguridad.
- **Generación byte a byte con entropía uniforme**: Se generan 32 valores aleatorios independientes, lo que cumple con los estándares criptográficos actuales [26] y proporciona una longitud de búsqueda de 2^{256}, inarbordable con medios computacionales actuales. Además, gracias al uso de `uniform_int_distribution<>dis(0, 255);` se garantiza una distribución uniforme, es decir, cada byte tiene igual probabilidad de ser cualquier valor posible.

Estas políticas aseguran que las claves utilizadas en el sistema mantengan un nivel de seguridad adecuado durante todo su ciclo de vida [27].

3.3.3. Almacenamiento de claves en dispositivos y en servidor

El almacenamiento de seguro de claves es fundamental para prevenir accesos no autorizados. En los dispositivos, las claves se almacenan en áreas de memoria protegidas, lo que es la EEPROM.

En el servidor, se almacena tanto su clave como la de los dispositivos de las cuales ya lo ha ido pidiendo. El servidor guarda su clave (`Cs`) en un fichero llamado `config_inicial_servidor.json` con la siguiente estructura:

```
{
  "clave_compartida": "CLAVE_SERVIDOR_CS",
  "id_servidor": "ID_SERVIDOR"
}
```

Figura 3.6: Estructura de la configuración inicial del servidor

Además, en el archivo `claves_placas.json` el servidor almacena de forma segura las claves de los dispositivos (`Cpx`) para no tener que pedírselas de nuevo al servidor y así ser más eficiente. La estructura del fichero es:

```
{
  "ID_DISPOSITIVO"{
    "clave": "CLAVE_DISPOSITIVO",
    "nombre": "NOMBRE_VISIBLE_DISPOSITIVO"
  }
}
```

Figura 3.7: Estructura del fichero de claves del servidor

3.3.4. Distribución de claves inicial

El proceso de distribución de claves inicial es crítico para establecer una base segura en las comunicaciones del sistema. El flujo depende de si es el servidor

o los dispositivos quien se está registrando. A continuación, se detalla el flujo para ambos casos:

Servidor

1. El servidor, la primera vez que se enciende, está vacío, no contiene ningún tipo de información.
2. Cuando se ha encendido por primera vez, se le pide al usuario que introduzca el ID del servidor, que debe coincidir con el que se escribió cuando se le dio de alta en el gestor.
3. Con ese ID, el servidor le pide al gestor su clave.
4. El gestor mira si tiene el ID registrado en `dispositivos.json` y, si lo tiene, envía la clave junto con el nombre.
5. El servidor recibe esta información, la descifra y la almacena en el archivo `config_inicial_servidor.json`.

Dispositivos

1. El dispositivo, la primera vez que se enciende, está vacío, no tiene en la EEPROM ningún tipo de información.
2. Cuando se ha encendido por primera vez, mira cuál es su ID (lo hace mediante una función) y le pide al gestor su clave.
3. El gestor, si tiene la ID guardada en `dispositivos.json`, le envía al dispositivo su clave (`Cpx`) junto con el nombre con el que será visible.
4. El dispositivo recibe la información y guarda su clave en la EEPROM.

Estos procesos aseguran que tanto el servidor como cada dispositivo tengan una clave única y que todas las partes involucradas estén sincronizadas para establecer comunicaciones seguras desde el inicio.

3.4. Consideraciones de seguridad en el diseño

La seguridad es un aspecto clave en el diseño de sistemas IoT, especialmente cuando se trata de manejar datos sensibles y asegurar la integridad y confidencialidad de la información que se transmite. En este proyecto, se ha identificado y abordado varias amenazas potenciales, implementando las medidas de mitigación adecuadas. Sin embargo, también se reconocen ciertas limitaciones que vienen con el entorno y con los recursos disponibles.

3.4.1. Amenazas consideradas

Durante el proceso de diseño, se han tenido en cuenta las siguientes amenazas relevantes:

- **Interceptación de datos en tránsito**: La posibilidad de que un atacante capture la información que se transmite entre dispositivos y el servidor poniendo en riesgo la confidencialidad de esos datos.
- **Compromiso de las claves**: El peligro de que las claves criptográficas sean expuestas o robadas, lo que permitiría a un atacante descifrar comunicaciones o crear mensajes falsos.
- **Ataques de denegación de servicio (DoS)**: Intentos de saturar el servidor o los dispositivos con tráfico malicioso, interrumpiendo así el funcionamiento normal del sistema.

3.4.2. Medidas tomadas

Para mitigar las amenazas identificadas, se han implantado las siguientes medidas de seguridad:

- **Cifrado de datos**: Se utiliza el protocolo D2OTP, que emplea un esquema de cifrado de un solo uso (One-Time Pad) gracias al uso de un contador, asegurando así que cada mensaje sea único y confidencial.
- **Autenticación y validación de mensajes**: Cada mensaje incluye un identificador de dispositivo, permitiendo al dispositivo verificar la autenticidad del emisor. Además, se incluye un tag de autenticación generado automáticamente por el modo GCM, lo que permite al receptor verificar la autenticidad e integridad de los datos recibidos, impidiendo así la aceptación de mensajes manipulados.
- **Gestión segura de las claves**: Las claves criptográficas se generan de forma segura y se almacenan en áreas seguras del servidor y de los dispositivos. Además, se implementa un gestor de claves que facilita la distribución y rotación de las mismas.

3.4.3. Limitaciones conocidas

A pesar de las medidas implementadas, se reconocen las siguientes limitaciones:

- **Recursos limitados de los dispositivos**: Los dispositivos IoT utilizados tienen limitaciones en su capacidad de procesamiento y almacenamiento. Esto hace que sea complicado implementar algoritmos de seguridad más avanzados o guardar grandes cantidades de datos.
- **Dependencia de la seguridad física**: La seguridad del sistema también depende de cómo se protegen físicamente los dispositivos. Si un atacante logra acceder físicamente a uno de ellos, podría robar información sensible o incluso alterar su funcionamiento.
- **Actualizaciones de firmware**: La forma en que se implementan los mecanismos para actualizar el firmware de manera segura es bastante limitada, lo que puede complicar la tarea de corregir vulnerabilidades que se descubren después de que se ha realizado el despliegue.

- **Escalabilidad del sistema**: A medida que se añaden nuevos dispositivos al sistema, la gestión de claves y la capacidad del servidor para procesar mensajes pueden convertirse en cuellos de botella si no se implementan estrategias de escalado adecuadas.

Estas evaluaciones destacan la necesidad de una evaluación continua de la seguridad del sistema y la implementación de mejoras conforme evolucionen las amenazas y las tecnologías disponibles.

3.5. Diseño de la interfaz web para visualización en tiempo real

La interfaz web desarrollada en este proyecto tiene como objetivo principal ofrecer una visualización en tiempo real de las lecturas de temperatura y humedad que recogen los sensores DHT22 conectados a los dispositivos ESP32. Gracias a esta interfaz, los usuarios pueden monitorear las condiciones ambientales de forma eficiente y accesible desde la web, sin necesidad de instalar software adicional.

3.5.1. Arquitectura de la interfaz web

La arquitectura de la interfaz web se basa en un modelo cliente-servidor. En este sistema, el servidor central se encarga de recopilar y procesar los datos que envían los dispositivos, presentándolos al cliente a través de una página web dinámica. Para llevar a cabo esta tarea, se utilizan tecnologías como Node.js y Express.js, que permiten gestionar múltiples conexiones al mismo tiempo, garantizando así una experiencia de usuario fluida.

Para mantener los datos actualizados en tiempo real, se emplea un mecanismo de consulta periódica (polling) a través de JavaScript. Cada pocos segundos, el cliente envía una solicitud HTTP al servidor para obtener las últimas medidas almacenadas. Aunque esta solución es más simple que las técnicas basadas en eventos, resulta suficiente y efectiva para el número limitado de dispositivos y usuarios que se espera en el sistema.

3.5.2. Diseño de la interfaz de usuario

La interfaz de usuario (UI) ha sido diseñada con un enfoque en la usabilidad y la accesibilidad, lo que garantiza que sea intuitiva y fácil de navegar. Se utilizan HTML5, CSS3 y JavaScript para desarrollar una página web responsiva que se ajusta a diferentes tamaños de pantalla y dispositivos.

En la página principal, se pueden ver las lecturas actuales de temperatura y humedad en formato numérico. Se incluye un botón de historial que nos permite ver, los datos de las 10 últimas mediciones. Además de una tabla con estas mediciones, se emplean gráficos de líneas para mostrar como varían estas variables (temperatura y humedad) a lo largo del tiempo, utilizando bibliotecas como Chart.js para hacer la implementación más sencilla. Además, se han añadido

indicadores visuales que avisan al usuario cuando los valores superan ciertos límites predefinidos o cuando un dato ha quedado desactualizado, lo que permite identificar rápidamente cualquier condición anómala.

3.5.3. Integración con el sistema IoT

La interfaz web está profundamente conectada con el sistema IoT desarrollado en este proyecto. Los dispositivos ESP32, que cuentan con sensores DHT22, envían de manera regular las lecturas de temperatura y humedad al servidor central a través del protocolo HTTP y utilizando mensajes en formato JSON. El servidor se encarga de procesar esos datos y almacenarlos en un fichero (`datos.json`), desde donde se recuperan y se muestran en la interfaz web.

Esta integración permite visualizar en tiempo real las condiciones ambientales, lo que facilita la toma de decisiones informadas y la detección temprana de posibles problemas. Además, la arquitectura modular del sistema hace que sea fácil expandirlo, permitiendo la adición de nuevos sensores o funcionalidades en el futuro.

Capítulo 4

Implementación y desarrollo

4.1. Firmware del dispositivo IoT

El firmware que se ha desarrollado para el dispositivo IoT, que corre sobre una placa ESP32-C3, es el corazón del sistema distribuido plantado. Su principal objetivo es gestionar el registro inicial del dispositivo, recoger datos ambientales a través de un sensor DHT22, cifrarlos con una clave OTP segura que proporciona el gestor de red, y enviarlos al servidor central usando UDP. Además, cuenta con mecanismos para asegurar la persistencia, controlar errores y actualizar el estado a través de una pantalla integrada, lo que mejora la robustez del sistema y la trazabilidad de los eventos.

4.1.1. Configuración inicial

Durante la fase de inicialización, la placa realiza la configuración de los periféricos esenciales:

- **Red Wi-Fi**: Se establece la conexión a la red mediante las credenciales definidas en `config_inicial_placas.h`, pero es importante saber que, por soporte de la placa, solo se pueden usar WI-Fi en la banda de 2.4 GHz [28].
- **Pantalla OLED**: Mediante la biblioteca `U8g2lib.h`, se inicializa la pantalla de 128x64 píxeles conectada a la placa por I2C.
- **Sensor DHT22**: Se activa la lectura mediante la librería `DHT.h`, asignando el pin digital al cual está conectado.
- **EEPROM**: Se reserva un espacio de 128 bytes donde se almacenan la clave base (`Cpx`) y el contador asociado al cifrado.

Además, se calculan las direcciones IP del gestor y del servidor mediante las constantes definidas, y se inicia el servicio `WifiUDP` para permitir tanto el envío como la recepción de datagramas.

4.1.2. Lógica de registro

La primera acción lógica que se lleva a cabo después de la configuración es verificar si el dispositivo ya tiene una clave guardada en la EEPROM. Si no la tiene (por ejemplo, tras el primer arranque o un reinicio de memoria), se inicia un proceso de alta seguro con el gestor.

El procedimiento es el siguiente:

1. Se genera un `nonce` aleatorio de 16 bytes.
2. Se construye un mensaje JSON del tipo `solicitud_clave_placa` que incluye el identificador único de la placa y el nonce generado.
3. Este mensaje se envía al gestor vía UDP.
4. El gestor responde con un mensaje cifrado que contiene la clave base (`Cpx`) y un tag de autenticación.
5. La placa descifra este mensaje localmente gracias a la ayuda de la clase `D2OTP`, verificando la integridad del contenido mediante el nonce original.
6. Una vez validada, la clave OTP se almacena en la EEPROM y se inicializa el contador a cero.

Este procedimiento garantiza que la clave no se transmite en claro y que no puede ser ni reutilizada ni manipulada, preservando las propiedades del esquema OTP.

4.1.3. Lectura de sensores

Las lecturas periódicas se llevan a cabo utilizando el sensor DHT22, conectado a la placa ESP32. En cada ciclo de operación, el dispositivo:

- Lee la temperatura y la humedad ambiental
- Verifica que los valores leídos no sean ni nulos ni extremos.
- Muestra los datos en la pantalla OLED de la placa, hasta que se hace otra lectura y se sobrescribe.

La frecuencia de lectura está configurada para mantener un equilibrio entre la actualización en tiempo real y el consumo energético.

4.1.4. Envío seguro de datos

La transmisión al servidor se lleva a cabo a través de datagramas UDP, pero protegidos mediante una variante adaptada del protocolo D2OTP. El flujo que se sigue es el siguiente:

1. Se construye un nueva clave para cada envío, utilizando la clave base (`Cpx`) y el contador actual.
2. Se genera el mensaje que contiene la temperatura y la humedad recibidas del sensor.

3. El contenido del mensaje se cifra con la clave OTP generada mediante AES-GCM.
4. Se adjunta tanto el `ID` del dispositivo, el `contador` y el `tag` de autenticación y se envía todo el paquete al servidor.

Este enfoque garantiza:

- **Confidencialidad**: Los datos viajan cifrados.
- **Autenticidad e integridad**: cualquier alteración del mensaje es detectable gracias al `tag`.
- **Unicidad criptográfica**: Cada cifrado es único por el uso del contador, que actúa como IV.

El contador se incrementa tras cada envío y se guarda en la EEPROM para persistencia tras reinicios.

4.1.5. Recepción de confirmaciones

El servidor, después de procesar y validar el mensaje, devuelve una respuesta del tipo `ACK` que indica la recepción correcta de los datos. El firmware está preparado para:

- Escuchar una respuesta durante un tiempo definido.
- Verificar que se trata de una respuesta del tipo esperado.
- Mostrar en la pantalla un check visual de confirmación.
- Si no se recibe el `ACK` en tiempo, el envío se considera fallido, pero no se retrocede el contador para evitar vulnerabilidades criptográficas.

Este mecanismo proporciona un canal unidireccional seguro, robusto y eficiente, adecuado para sistemas embebidos de bajo consumo.

4.2. Librería D2OTP

La librería D2OTP ofrece las herramientas criptográficas necesarias para asegurar la confidencialidad, integridad y autenticidad de las comunicaciones entre los dispositivos Iot, el servidor y el gestor. Esto se logra mediante un esquema de cifrado de un solo uso (One-Time Pad) que se refuerza con un contador y cifrado simétrico autenticado.

Su diseño modular facilita la integración tanto en entornos embebidos, como el ESP32 utilizando mbedtls, como en servidores que se basan en OpenSSL, todo con interfaces uniformes para las operaciones de cifrado y descifrado.

4.2.1. Funcionalidades ofrecidas

Esta librería ofrece los siguientes métodos esenciales:

- `calcularSHA256`: Genera el hash de SHA-256 de un input binario. Este método es esencial para derivar claves a partir de contraseñas u otros elementos identificadores.
- `encrypt`: Realiza el cifrado de un mensaje binario utilizando el algoritmo AES-256-GCM, asegurándose de emplear un nonce único como IV (vector de inicialización) para cada operación. Además, devuelve el `tag` de autenticación que permite verificar la integridad del mensaje en su destino.
- `decrypt`: descifra un mensaje cifrado con AES-256-GCM, validando el `tag` recibido para comprobar que no ha habido ningún tipo de manipulación de los datos.

Internamente, todos los métodos tienen en cuenta las diferencias entre plataformas: utilizan `mbedtls` en las placas ESP32 y OpenSSL en los entornos del servidor. Esto permite que el mismo código base se reutilice en ambas plataformas sin sacrificar ni la compatibilidad ni la seguridad.

4.2.2. Integración con mbedTLS/OpenSSL

Uno de los puntos fuertes del diseño de la librería D2OTP es su portabilidad. Para garantizar su uso tanto en dispositivos de bajo nivel como en servidores, se implementan directivas de compilación condicional:

- En las plataformas ESP32, se integra con `mbedtls`, la solución más extendida en sistemas embebidos. Aquí se utiliza `mbedtls_gcm_crypt_and_tag` para el cifrado y `mbedtls_gcm_auth_decrypt` para el descifrado autenticado.
- En plataformas Linux (donde, en este proyecto, se aloja tanto el gestor como el servidor), se apoya en `OpenSSL`, utilizando `EVP_aes_256_gcm()` con sus correspondientes funciones de inicialización, actualización y finalización (`EVP_EncryptInit_ex`, `EVP_EncryptUpdate`, etc.).

Este enfoque garantiza el uso de cifrado autenticado, resistente frente a ataques de replay y manipulación.

4.2.3. Validación de cifrado

Durante la operación del sistema, el cifrado es válido de extremo a extremo:

- El dispositivo cifra el payload JSON de los datos de los sensores con una clave OTP derivada de la clave base, utilizando como IV un `contador` o un `nonce` aleatorio.
- El gestor, al enviar la clave, también cifra el paquete de respuesta. La placa verifica mediante el `tag` la integridad del mensaje.
- El servidor, tras recibir un mensaje cifrado desde una placa, solicita al gestor la clave correspondiente `Cpx`. Esta clave la usa para descifrar el contenido y verificar la autenticidad del `tag`.

El sistema está diseñado de tal manera que la clave nunca se envía en texto claro. El nonce funciona como una sal única para cada operación, mientras que el contador garantiza que cada mensaje cifrado sea único, lo cual es fundamental en cualquier esquema OTP.

Además, se llevan a cabo validaciones explícitas de los resultados que devuelven `mbedtls` o `OpenSSL`, y se aborta la operación si el descifrado no pasa la verificación de autenticidad (tag inválido).

4.3. Gestor de red

El componente del gestor de red funciona como un intermediario seguro, encargado de proteger y distribuir las claves criptográficas que son esenciales para establecer canales de comunicación cifrada entre los dispositivos IoT y el servidor central. Su papel es crucial dentro del protocolo D2OTP, ya que garantiza una clara separación de responsabilidades y asegura que las claves nunca se expongan directamente durante su transmisión.

El gestor se desarrolla en el lenguaje C++ y utiliza la biblioteca `nlohmann::json` para manejar configuraciones persistentes en formato JSON, además de emplear sockets UPD pra comunicarse con las placas y el servidor.

4.3.1. Estructura del programa

El gestor mantiene un archivo llamado `dispositivos.json` donde se almacenan tanto las claves base asignadas a cada entidad (servidor y placas) como los nombres visibles que tendrán, por ejemplo, en la interfaz web, indexadas por su identificador único (`id_dispositivo`).

Además el programa define funciones para cargar, guardar y generar claves de manera segura y atómica mediante la exclusión mutua (`std::mutex`) y el control de concurrencia.

El gestor está diseñado para escuchar en el puerto 5000/UDP y atiende dos tipos de solicitudes: aquellas provenientes de dispositivos IoT que desean su clave, y peticiones del servidor cuando requiere claves de terceros para poder descifrar los mensajes entrantes.

4.3.2. Menú del gestor

Cuando se enciende el gestor, nos sale un menú con distintas operaciones a realizar:

1. **Listar dispositivos**: Al entrar en esta opción, aparece una lista con el ID y el nombre los dispositivos ya registrados en `dispositivos.json`
2. **Añadir dispositivo**: Si se selecciona esta opción, se pide al usuario que introduzca el nuevo ID y el nombre descriptivo visible para almacenarlo en `dispositivos.json`

3. **Eliminar dispositivo**: Al entrar, se pide al usuario que introduzca en ID, y, si este existe en `dispositivos.json`, se eliminará.
4. **Salir**: Al seleccionar esta opción, se cierra el gestor, impidiendo la llegada de más solicitudes de clave

Este menú permite tener al usuario un control de todos los dispositivos registrados en el gestor.

4.3.3. Comunicación segura con el servidor

El servidor central puede solicitar al gestor la clave asociada a cualquier dispositivo registrado. El flujo se mantiene consistente: se envía un nonce, y el gestor responde con la clave cifrada y autenticada. Esta operación se lleva a cabo cuando el servidor necesita descifrar un mensaje de una placa.

Este mecanismo evita la necesidad de mantener desde un principio todas las claves en el propio servidor, reduciendo así la superficie de ataque.

4.3.4. Gestión de datos persistentes

El archivo `dispositivos.json` actúa como base de datos persistente al gestor. Toda modificación (alta o baja) se realiza sobre esta estructura, la cual se serializa y desearializa utilizando `nlohmann:json`.

Además, para evitar condiciones de carrera entre operaciones concurrentes (por ejemplo, dos solicitudes simultáneas), se emplea un `std::mutex` (`mtx_jsn`) para sincronizar el acceso de lectura y de escritura al fichero.

4.4. Servidor central de datos

El servidor central funciona como el corazón del sistema, recolectando, verificando y almacenando las mediciones periódicas que envían los dispositivos IoT conectados en red. También se asegura de que la comunicación sea segura, utilizando claves distribuidas a través del gestor y aplicando funciones criptográficas con la librería D2OTP.

Este servidor ha sido desarrollado en el lenguaje de programación C++ y emplea sockets UDP para recibir mensajes, `nlohmann::json` para manejar datos estructurados y `mbedtls` (a través de la librería D2OTP) para el descifrado y la autenticación.

4.4.1. Inicialización

El servidor inicia su proceso verificando si ya tiene guardada su configuración criptográfica, es decir, su clave base `Cs` y su identificador, a través del archivo `config_inicial_servidor.json`. Si este archivo no está presente, el servidor solicita de manera segura al gestor la clave, utilizando un nonce aleatorio y recibiendo la clave cifrada con AES-GCM.

Una vez que obtiene la clave, la almacena en memoria en el archivo mencionado y la convierte a formato hexadecimal para su uso futuro. Esta configuración permite que el servidor se autentique como una entidad segura dentro del esquema de D2OTP.

4.4.2. Recepción de datos UDP

El servidor escucha en el puerto 6000/UDP. La recepción de datos se implementa mediante un hilo dedicado que permanece a la espera de mensajes JSON estructurados con los campos:

- `id`: Identificador del dispositivo que envía.
- `contador`: Contador utilizado como IV en el proceso del cifrado del mensaje.
- `payload`: Mensaje cifrado.
- `tag`: Código de autenticación generado.

En el caso de que sea una de las placas las que envía el mensaje, y:

- `clave_cifrada`: Clave base `Cs` que viene cifrada.
- `tag`: Código de autenticación generado.

En el caso de la respuesta del gestor.

Una vez recibido un mensaje, se lanza un hilo independiente que se encarga de su validación y procesamiento, asegurando así la concurrencia y capacidad de escalar a múltiples dispositivos simultáneos.

4.4.3. Descifrado y verificación de mensajes

Antes de descifrar cualquier mensaje, el servidor primero revisa su caché local (`claves_placas.json`) para buscar la clave base del dispositivo emisor (`Cpx`). Si no la tiene, solicita esa clave al gestor, asegurándose de que esté cifrada y autenticada mediante un nonce.

Una vez que tiene la clave en memoria, utiliza la función `descifrarMensaje` de la librería D2OTP, que emplea AES-256-GCM para garantizar tanto la confidencialidad como la integridad del mensaje. Si la verificación del tag falla, el mensaje se descarta. Si el resultado ha sido correctamente descifrado, se le envía un mensaje de confirmación (`ACK`) a la placa emisora.

4.4.4. Almacenamiento de lecturas

El servidor guarda las diez últimas lecturas en el archivo `datos.json`, junto con el `id_dispositivo` y el `nombre`, la `hora` de la medida y los valores de los sensores. La estructura de ese archivo, suponiendo que en el historial ya hay una lectura guardada es:

```
{
  "dispositivos"{
    "ID_DISPOSITIVO":{
        "actual":{
            "hora": "HORA_ACTUAL",
            "humedad": "VALOR_HUMEDAD",
            "temperatura": "VALOR_TEMPERATURA"
        },
        "historial":[
        {
           "hora": "HORA_DATO",
           "humedad": "VALOR_HUMEDAD",
           "temperatura": "VALOR_TEMPERATURA"
        }
        ],
        "nombre": "NOMBRE_VISIBLE_DISPOSITIVO"
    }
  }
}
```

Figura 4.1: Estructura del fichero `datos.json` *del servidor*

Para asegurar que todo funcione sin problemas y evitar problemas de concurrencia, el acceso al archivo se maneja a través de `std::mutex`. Esto garantiza que no haya dos hilos escribiendo al mismo tiempo. Las nuevas entradas se añaden al final del array del JSON, lo que facilita su análisis posterior en su visualización web o en herramientas externas.

4.4.5. Envío de respuesta/ACK a dispositivos

Tras el procedimiento exitoso de una lectura, el servidor genera una respuesta simple de comunicación en texto plano (`"OK"`) que se devuelve al dispositivo utilizando la dirección IP y puerto del emisor original, completando el ciclo de comunicación segura.

Este ACK sirve también como mecanismo de sincronización y retroalimentación, permitiendo a la placa emisora confirmar que sus datos han sido recibidos y aceptados.

4.4.6. Peticiones al gestor en tiempo de ejecución

En el momento en el que el servidor recibe por primera vez un mensaje de un nuevo dispositivo (no presente en su caché local), ejecuta automáticamente una solicitud al gestor mediante el protocolo definido:

1. El servidor genera un nuevo nonce aleatorio.
2. Envía al gestor un JSON cifrado con su clave base (`Cs`) y el nonce utilizado como IV con el tipo `"solicitud_clave"` y el `id_placa`.
3. Recibe una mensaje cifrado, junto con un tag de autenticidad.

4. El servidor lo descifra utilizando su clave `Cs` y obtiene la clave `Cpx` de la placa.
5. La almacena en `claves_placas.json`.

Este mecanismo permite incorporar nuevos dispositivos de forma segura y dinámica sin la necesidad de la intervención manual.

4.5. Aplicación web de monitorización

Para hacer más fácil el seguimiento en tiempo real de las mediciones que registran los dispositivos IoT, se ha creado una interfaz web accesible desde cualquier navegador. Esta aplicación permite ver las lecturas que recibe el servidor, organizadas por dispositivo, y también consultar el historial de medidas a través de gráficos y tablas detalladas.

4.5.1. Servidor FastAPI

El backend de la aplicación web está construido con FastAPI, un framework moderno y eficiente basado en Python 3. La estructura es minimalista pero funcional:

- El archivo `main.py` define un servidor HTTP que expone dos rutas principales:
  - `/`: Sirve el archivo HTML principal (`index.html`), que actúa como interfaz del usuario.
  - `/datos`: Expone el contenido del archivo `datos.json` generado por el servidor de recepción, proporcionando las lecturas registras en formato JSON.

Este servidor se ejecuta localmente en el puerto 8080 mediante Uvicorn. Gracias al uso de FastAPI, se logra una respuesta rápida, se aprovecha la sincronía nativa y se cuenta con la compatibilidad para herramientas de documentación automática como Swagger UI, aunque en este proyecto, por razones de simplicidad, no se utiliza.

4.5.2. Página web

El frontend se compone de un único archivo HTML (`index.html`) acompañado por referencias externas a:

- **TailWindCSS**: Para un diseño visual moderno y adaptable (responsive), con sombras, gradientes y animaciones [29].
- **Chart.js**: Para la generación dinámica de gráficos de líneas que muestran temperatura y humedad en función del tiempo [30].

La interfaz incluye las siguientes secciones:

- Un encabezado que da título a la página

- Un panel principal donde cada tarjeta representa un dispositivo distinto.
- Al hacer click en el botón de "Ver Historial.en un dispositivo, se muestra una vista detallada con:
    - Una tabla cronológica de medidas.
    - Dos gráficos, uno para la temperatura y otro para la humedad.
    - Un botón para volver a la vista principal.

El diseño está adaptado para visualizarse correctamente en pantallas de escritorio, tablets y móviles, manteniendo un enfoque accesible y minimalista.

4.5.3. Lógica de actualización en la página

El script JavaScript embebido en el HTML es el que gestiona toda la lógica del frontend. Realiza peticiones periódicas al endpoint `/datos` para obtener la información más actual y actualizar dinámicamente las tarjetas y los gráficos.

Cada tarjeta contiene:

- El nombre visible del dispositivo
- La última lectura de temperatura y humedad junto con la hora de la medida.
- Un icono parpadeante en caso de que se lleven sin actualizar los datos de esa placa más de un minuto.

Cuando se selecciona un dispositivo, el historial de lecturas se reconstruye automáticamente en la tabla y en los gráficos. Esto facilita que el usuario pueda identificar tendencias o posibles anomalías sin complicaciones.

La aplicación actualiza sus datos en segundo plano, asique no es necesario tener que recargar la página, lo que brinda una experiencia más fluida y agradable al usuario.

4.5.4. Ejemplo de visualización

La interfaz está diseñada para soportar múltiples dispositivos en paralelo. Por ejemplo, si el servidor recibe lecturas del dispositivo `ESP32_Placa_1`, la tarjeta correspondiente mostrará tanto la última temperatura y humedad registradas como la hora de estas medidas, y al hacer click en el botón se desplegará:

- Un historial tabulado con marcas de tiempo
- Gráficos temporales generados dinámicamente mediante Chart.js.

Ver Anexo E, donde se muestra una captura tanto de la interfaz principal como del historial.

4.6. Integración de componentes y ejecución

La implementación completa del sistema se organiza en cuatro componentes clave: dispositivo IoT, un gestor de red, un servidor de recepción y una aplicación web para la visualización. Estos elementos trabajan juntos de manera sincronizada para crear un sistema seguro y eficiente para la transmisión de datos en entornos IoT. La integración adecuada de estos módulos ha sido fundamental en el desarrollo, garantizando que las diversas tecnologías empleadas, como C++, Python, FastAPI y HTML, funcionen en perfecta armonía.

4.6.1. Puesta en marcha del sistema paso a paso

A continuación se detalla el procedimiento recomendado para desplegar y ejecutar correctamente el sistema:

1. **Arranque del gestor de red**

   El gestor se ejecuta como primer componente, ya que es el encargado de centralizar las claves criptográficas. Al iniciarse:

   - Carga el archivo `dispositivos.json`, que contiene las claves de los dispositivos registrados.
   - Escucha peticiones por UDP en el puerto asignado, esperando solicitudes de alta de dispositivos o peticiones de claves por parte del servidor.

2. **Inicialización del servidor**

   El servidor central de recepción de datos debe inicarse en segundo lugar. Este:

   - Carga las configuraciones necesarias y estable una conexión UDP en el puerto configurado.
   - Se comunica con el gestor cuando necesita obtener o verificar claves, especialmente cuando recibe un mensaje cifrado por primera vez de un dispositivo.

3. **Encendido del dispositivo IoT**

   Cada dispositivo ESP32-C3 arranca su firmware configurando la red Wi-Fi, inicializando sensores y solicitando su clave base `Cpx` al gestor. El flujo que sigue es:

   - Si la EEPROM no contiene su clave, el dispositivo se la solicita al gestor.
   - Una vez que obtiene la clave, inicia un bucle periódico de lectura de medidas y envío de las mismas.
   - Cada mensaje va cifrado con D2OTP y enviado al servidor vía UDP.

4. **Ejecución de la aplicación web**

   Finalmente, se lanza la aplicación con el comando:

```
python3 main.py
```

Desde este momento, la interfaz web está disponible en la página web `http://localhost:8080`, mostrando la información contenida en el fichero `datos.json`, que es constantemente actualizado por el servidor.

Recomendación

Para asegurar la persistencia de datos entre sesiones, se recomienda no eliminar los archivos `datos.json` ni `dispositivos.json`, a menos que se desee reiniciar el sistema desde cero.

4.6.2. Interacciones entre componentes

El sistema implementa una arquitectura distribuida en la que cada componente tiene un rol definido, pero necesita colaborar con los demás para funcionar correctamente. A continuación se describe la interacción entre ellos:

- **Dispositivo ↔ Gestor**

  El dispositivo solicita su clave criptográfica (`Cpx`) al gestor cuando se inicializa. Esta comunicación se realiza vía UDP mediante mensajes JSON. El servidor responde con la clave en hexadecimal.

- **Servidor ↔ Gestor**

  Cuando el servidor necesita su propia clave `Cs` se la pide al gestor de forma muy parecida a la solicitud por parte de las placas. También, cuando el servidor recibe un mensaje cifrado, puede requerir la clave base (`Cpx`) del emisor. Para ellos solicita esa clave al gestor que le responderá con ella si la tiene registrada.

- **Dispositivo ↔ Servidor**

  La interacción principal de este proyecto. El dispositivo envía sus lecturas cifradas al servidor usando un esquema One-Time Pad con contador, incluyendo una etiqueta de autenticación. El servidor:

  - Obtiene la clave adecuada.
  - Descifra y valida el mensaje.
  - Almacena los datos en `datos.json`.
  - Envía un ACK al dispositivo para confirmar recepción.

- **Servidor ↔ Web**

  La comunicación se realiza mediante el archivo `datos.json`. La web accede a este recurso a través del endpoint `/datos`, actualizado constantemente por el servidor con los datos más recientes.

Esta arquitectura desacoplada permite escalar el sistema fácilmente, añadir más dispositivos o migrar componentes a distintas máquinas sin afectar al funcionamiento general.

Capítulo 5

Pruebas y Resultados

5.1. Entorno de pruebas

Con el fin de validar el correcto funcionamiento del sistema propuesto, se ha definido un entorno de pruebas controlado que simula un escenario realista para el despliegue de dispositivos IoT. En este entorno, todos los componentes del sistema operan de manera distribuida en una misma red local. Esto nos permite evaluar cómo se comporta el sistema en términos funcionales, su robustez criptográfica y el rendimiento de las comunicaciones seguras utilizando el protocolo D2OTP.

5.1.1. Disposición general del sistema

Las infraestructura de pruebas se ha ejecutado íntegramente en una red Wi-Fi doméstica, con los siguientes elementos conectados a través de una red local configurada mediante un router convencional:

- **Dispositivo IoT (ESP32-C3)**:
  - Modelo: ESP32-C3 con pantalla OLED integrada.
  - Sensor conectado: DHT22 (AM2302) para medir la temperatura y la humedad ambiental.
  - Fuente de alimentación: conexión USB-C a un ordenador portátil.
  - Firmware cargado mediante el IDE de Arduino, empleando el archivo `__Placas_UDP.ino` y la red configuración de red especificada en `config_inicial_placas.h`.
- **Servidor central de recepción de datos**:
  - Lenguaje: C++ (compilado con `g++` en Ubuntu 22.04).
  - Archivo ejecutable: `servidor_completo`.
  - Escucha por UDP en el puerto 6000.

- Archivos utilizados:
  - claves_placas.json: almacenamiento temporal de las claves base (Cpx) por ID de dispositivo.
  - datos.json: almacenamiento de las lecturas descifradas.
  - config_inicial_servidor.json: almacenamiento de la clave y el ID del servidor.

- **Gestor de red**:
  - Lenguaje: C++ (compilado con g++ en Ubuntu 22.04).
  - Archivo ejecutable: gestor
  - Escucha por UDP en el puerto 5000.
  - Archivo crítico: dispositivos.json, que contiene los ID de los dispositivos autorizados, su clave base (Cpx) y la clave del propio servidor (Cs).

- **Aplicación web de visualización**:
  - Framework: FastAPI (Python 3.10).
  - Archivo principal: main.py.
  - Plantilla HTML: templates/index.html.
  - Endpoint relevante: /datos, que accede a datos.json para visualizar en tiempo real las últimas mediciones.

Nota

Todos los servicios se ejecutaron en una máquina virtual Ubuntu 22.04, excepto la placa ESP32-C3, que se comunicó del exterior mediante Wi-Fi.

5.2. Pruebas funcionales

Para validar la lógica funcional del sistema, se llevaron a cabo pruebas exhaustivas enfocadas en verificar los tres pilares fundamentales de la comunicación segura: el registro inicial correcto del dispositivo, el envío regular de datos cifrados y la recepción de confirmaciones (ACK) por parte del servidor. Estas pruebas aseguran que los diferentes módulos del sistema trabajen juntos de manera efectiva utilizando el protocolo D2OTP adaptado.

Las pruebas se realizaron siguiendo el entorno descrito en la sección 5.1, y los resultados se documentaron a través de la inspección directa de los registros en consola, el análisis de archivos generados y la visualización mediante la interfaz web.

Para realizar las pruebas, se registraron en el servidor todos los actores involucrados en el sistema (servidor y placas) con su ID y su nombre. El gestor, a la vez que se iban añadiendo, iba creando su clave y almacenando toda esta

información en `dispositivos.json`, el cual se puede ver su contenido en el Anexo F.

5.2.1. Registro de un dispositivo nuevo

Servidor

La primera prueba funcional consistió en verificar el flujo completo del registro inicial del servidor en el sistema. El objetivo era confirmar que podía recibir bien su clave y almacenarla, sin la intervención humana directa.

Pasos observados:

1. El servidor, cuando se enciende, intenta leer su configuración inicial.
2. Al no encontrar el archivo y no tener su clave, le pide al usuario que introduzca cuál es su ID (que debe ser el mismo con el que se registró en el gestor).
3. Cuando tiene ya su ID, crea un mensaje JSON con el campo `"tipo": "solicitud_clave_servidor"` y lo envía por UDP al gestor.
4. El gestor localiza el ID en `dispositivos.json`, y responde con la clave del servidor `Cs`.
5. El servidor almacena esta información (su clave y su ID) en el archivo `config_inicial_servidor.json`.

Para validar que todo funcionó correctamente, se comprobó si se creó el archivo `config_inicial_servidor.json`, del cual se puede ver una captura en el Anexo G.

Dispositivo

La siguiente prueba radicó en verificar el flujo completo del registro inicial de una placa en el sistema. El objetivo era confirmar que podía recibir bien su clave y almacenarla.

Pasos observados:

1. La placa, al iniciar por primera vez, intenta leer de la EEPROM su clave.
2. Al no encontrarla, construye un mensaje JSON con el campo `"tipo": "solicitud_clave_placa"` y su ID que la obtiene mediante una función.
3. El gestor busca el ID en `dispositivos.json`, y responde, si encuentra el ID, con la clave de la placa (`Cpx`).
4. La placa guarda esta información en la EEPROM y está lista para enviar datos.

Validaciones realizas:

- Se monitorizó la consola de la placa y del gestor, verificando los mensajes de solicitud y respuesta.

- El contenido de la EEPROM fue inspeccionado indirectamente al reiniciar la placa: en el segundo arranque ya no pidió la clave.

5.2.2. Envío de datos periódico

Una vez obtenida la clave, el dispositivo comenzó a enviar periódicamente lecturas de temperatura y humedad procedentes del sensor DHT22. Para ello, se empleó la lógica contenida en el `loop()` del archivo `__Placas_UDP.ino`.

Pasos observados:

1. Cada 30 segundos, el firmware toma una nueva lectura del sensor.
2. Se construye un paquete cifrado con D2OTP incluyendo:
   - El ID de la placa.
   - El contador utilizado para cifrar.
   - La temperatura y la humedad medidas.
   - El tag generado.
3. El mensaje se envía por UDP al servidor.

Validaciones realizadas:

- El servidor mostró en consola este mensaje:

```
📥 [ESP32_Placa_1] Temp: 29.8 °C,  Hum: 26.1 %
```

Figura 5.1: Mensaje consola servidor al recibir datos

- Las lecturas se almacenaron en el archivo `datos.json`.
- Web renderizando nuevas filas dinámicamente mediante JavaScript con `fetch()`.

Se puede ver una captura del archivo `datos.json` en el Anexo H y una captura de la interfaz web en el Anexo E.

5.2.3. Confirmación de servidor encendido

Antes de enviar ningún mensaje, la placa se asegura de que el servidor está disponible mediante el método `Ping-pong`.

Pasos observados:

1. La placa, al ir a enviar, le envía al servidor el texto plano en claro `ping`.
2. Si el servidor recibe ese mensaje, contesta con un `pong` para confirmar que está a la espera de la recepción de datos.
3. El dispositivo comprueba que el mensaje recibido es el esperado, si lo es (el servidor ha respondido `pong`) lo es, si no, vuelve a intentarlo cada segundo hasta que se conecte.

Para validar que se envió correctamente, en el monitor serial de la placa salió este mensaje:

✅ Servidor activo, conexión verificada.

Figura 5.2: Confirmación de servidor encendido

5.2.4. Confirmación de recepción

El protocolo establece que el servidor debe enviar una respuesta cada vez que recibe un mensaje cifrado de manera correcta. Este mensaje de confirmación (ACK) asegura que el dispositivo puede seguir con el envío.

Pasos observados:

1. El servidor, tras descifrar correctamente, genera un mensaje ACK.
2. Este ACK (que contiene el texto plano en claro `"OK"`) se envía a la placa.
3. La placa lo recibe y continúa con su proceso.

Para validar que se envió correctamente, en el monitor serial de la placa salió este mensaje:

📤 Medida enviada: Temp: 29.9 °C, Hum: 25.9 %
✅ ACK recibido del servidor

Figura 5.3: Confirmación ACK

5.3. Pruebas de seguridad

El diseño del sistema propuesto se enfoca en asegurar una comunicación que sea confidencial, íntegra y robusta entre los dispositivos IoT y el servidor central. Para verificar que estas garantías se cumplen en la práctica, se llevaron a cabo varias pruebas de seguridad que validan los principios fundamentales de la criptografía aplicada. En todos los casos, los resultados se analizaron tanto a través de registros en consola como mediante la inspección del tráfico de red (utilizando Wireshark) y el análisis del comportamiento interno de los componentes involucrados.

5.3.1. Confidencialidad

El sistema implementa cifrado simétrico con autenticación (AES-128-GCM), garantizando que los mensajes enviados entre dispositivos y el servidor no puedan ser leídos por terceros. Para comprobarlo, se realizaron pruebas prácticas de interceptación y análisis de tráfico.

Procedimiento:

1. Se lanzó Wireshark en la interfaz de la red de la máquina receptora.

2. Se capturaron varios paquetes UDP provenientes de los dispositivos.
3. Se analizó el contenido de dichos paquetes.

Resultados:

- Todos los paquetes presentaban un payload completamente cifrado, sin ningún texto legible.
- Se verificó que los nonces eran distintos en cada envío, eliminando la posibilidad de ataques de repetición.

Ver Anexo I, donde se encuentra una captura de la aplicación Wireshark.

5.3.2. Integridad

El protocolo utiliza un tag de autenticación que se genera a través de AES-GCM, lo que permite verificar que el mensaje no ha sido modificado durante su transmisión. Esta verificación asegura la integridad de los datos, incluso si el mensaje es interceptado y alterado en el camino.

Procedimiento:

1. Se capturó un mensaje con la aplicación Wireshark.
2. Se descargó el contenido, se alteró el payload y se le envió al servidor.
3. Al llegar el mensaje al servidor, este lo descartó ya que los tags no coincidían.

Como resultado, podemos ver como el servidor desechó el paquete recibido mediante el siguiente mensaje en la consola, verificando el funcionamiento:

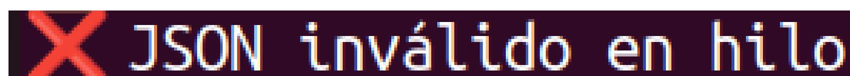


Figura 5.4: Mensaje error con mensaje alterado

5.4. Rendimiento básico

El sistema diseñado no solo debía cumplir con criterios de seguridad, sino también asegurar un comportamiento eficiente en entornos reales con dispositivos de recursos limitados como el ESP32. A continuación se presentan las pruebas realizadas para evaluar la latencia de comunicación y el uso de recursos del dispositivo durante la operación estándar del sistema.

5.4.1. Latencia de comunicación

Para medir el tiempo que pasa desde que un dispositivo IoT envía un dato hasta que recibe la confirmación del servidor (ACK), se modificó el código del firmware para registrar marcas de tiempo antes y después de la transmisión, añadiendo este fragmento de código:

```
unsigned long t_envio = millis();

//Lógica envío y recepción del ACK

unsigned long t_final = millis();

if (ackRecibido) {
    Serial.print("Latencia total: ");
    Serial.print(t_final - t_envio);
    Serial.println(" ms");
}
```

Se procedió a apuntar en la tabla adjunta el dato de las 10 últimas medidas, viendo un valor medio de 24.5 ms, con un máximo de 42 ms y un mínimo de 9, lo que nos demuestra un rendimiento más que adecuado para escenarios en tiempo real, como la monitorización ambiental.

Prueba	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Latencia (ms)	16	28	42	21	22	20	20	40	27	9

Cuadro 5.1: Tabla de latencia de envío

5.4.2. Uso de recursos en el ESP32

Dado que el ESP32 es un microcontrolador con recursos limitados, es esencial garantizar que la lógica del protocolo y el cifrado no superen sus capacidades.

Procedimiento:

1. Se utilizó la función `esp_get_free_heap_size()` para monitorizar el uso de la RAM.

2. Se midió el tamaño disponible tanto antes del cifrado, como después y tras el envío.

Podemos ver en la figura adjunta los valores de almacenamiento disponibles, que se mantienen muy regulares, lo que nos demuestra que el sistema es suficientemente ligero como para ejecutarse de forma estable en la ESP32 sin bloqueos.

```
📦 Heap antes de cifrado: 196436
📦 Heap después de cifrado: 195032
📤 Medida enviada: Temp: 29.8 °C,   Hum: 25.2 %
📦 Heap después del envío: 194684
✅ ACK recibido del servidor
```

Figura 5.5: Uso de recursos de la ESP32

5.5. Tabla resumen de pruebas

Con el fin de ofrecer una visión clara y concisa de los resultados experimentales obtenidos, a continuación se presenta una tabla resumen que reúne las pruebas clave realizadas sobre el sistema, organizadas según su tipo: funcional, seguridad y rendimiento.

Categoría	**Prueba realizada**	**Resultado esperado**	**Resultado obtenido**	**Logrado**
Funcional	Registro de dispositivo nuevo	Recepción de clave tras primer arranque, tanto placas como servidor	Clave recibida y confirmado en consola	✓ Sí
Funcional	Envío de datos periódico	Transmisión cifrada y recepción por el servidor	Datos enviados cada 30 s sin pérdidas	✓ Sí
Funcional	Confirmación de servidor encendido	Mensaje en monitor serial con la verificación	Mensaje de servidor activo recibido	✓ Sí
Funcional	Confirmación de recepción (ACK)	ACK cifrado desde servidor con contador válido	ACK recibido, contador incrementado	✓ Sí
Seguridad	Cifrado punto a punto	Confidencialidad garantizada vía AES-GCM	Ninguna lectura sin clave fue legible	✓ Sí
Seguridad	Integridad de mensajes	Validación con etiqueta y tag criptográfico	Mensajes alterados son rechazados	✓ Sí
Rendimiento	Latencia de ida y vuelta (round-trip)	Inferior a 100 ms	Media de 24.5 ms	✓ Sí
Rendimiento	Uso de RAM en ESP32	Debe mantenerse al menos 10 KB libres	~20 KB libres	✓ Sí

Cuadro 5.2: Tabla resumen de pruebas y resultados

5.6. Evaluación general del sistema

El sistema propuesto para asegurar y optimizar la comunicación de datos en entornos IoT, utilizando el protocolo D2OTP, ha sido sometido a un análisis exhaustivo. Esto incluye pruebas funcionales, simulaciones de amenazas y análisis de rendimiento . Los resultados obtenidos demuestran que el proyecto ha cumplido con los objetivos establecidos y ha alcanzado un nivel de madurez tanto funcional como técnica muy sólido.

Robustez y fiabilidad

El sistema ha demostrado ser bastante estable incluso en condiciones adversas. Gracias a su arquitectura descentralizada y a la adecuada separación entre el gestor de claves, el servidor de recepción y los dispositivos IoT, cada uno de estos componentes puede funcionar de manera autónoma y resistir fallos parciales.

Además, se ha confirmado que el sistema es capaz de recuperar su estado sin necesidad de intervención manual, incluso ante cortes forzados, lo cual refuerza su robustez para entornos reales.

Seguridad

El sistema utiliza un modelo de cifrado muy seguro que se basa en AES-GCM y la filosofía del One-Time Pad con contador, creando claves únicas para cada mensaje. Este método ha demostrado ser muy resistente a ataques pasivos (como la interceptación) y activos (como la repetición, modificación o suplantación), tal como se comprobó en la sección 5.3. La confidencialidad se asegura mediante cifrado autenticado, mientras que la integridad se verifica con etiquetas criptográficas.

La gestión de claves se realiza a través de un gestor que utiliza un archivo JSON centralizado y controles de acceso por ID, lo que ofrece una solución que es escalable, fácil de mantener y segura.

Eficiencia y Rendimiento

El sistema ha sido creado con un enfoque eficiente que abarca desde el firmware hasta la interfaz de usuario. En el ESP32-C3, el uso de recursos se mantiene bajo control, como se ha podido demostrar en la sección 5.4. Esto permite que funcione de manera continua durante largas sesiones sin necesidad de reinicios ni pérdida de rendimiento.

En lo que respecta a la comunicación, la latencia promedio para el ciclo completo (lectura, cifrado, envío, descifrado y ACK) ha sido inferior a 30 ms en condiciones de red estables, lo que está muy por debajo de los límites aceptables para aplicaciones de monitoreo ambiental.

Escalabilidad y Modularidad

Uno de los grandes puntos a favor del diseño es su capacidad de escalar horizontalmente. Se ha comprobado que el sistema puede expandirse sin problemas a varios dispositivos, ya que el modelo de registro y autenticación funciona de manera independiente para cada uno. Además, al haber separado el gestor de claves del servidor y del firmware, se abre la puerta a reutilizar o intercambiar componentes sin necesidad de alterar el resto de la infraestructura.

La estructura modular también hace que el mantenimiento sea más sencillo, ya que cada componente cuenta con su propio código, repositorio de datos y lógica de ejecución independiente.

Interfaz y Experiencia de usuario

La interfaz web desarrollada permite ver en tiempo real las mediciones que provienen de los dispositivos. Gracias a tecnologías modernas como FastAPI en el backend y JavaScript dinámico en el frontend, se ha logrado una visualización ágil y clara.

A pesar de algunas pequeñas latencias detectadas en momentos específicos (que hemos solucionado optimizando las peticiones), la experiencia del usuario es bastante positiva y se ajusta a los estándares que se esperan de un sistema de monitorización IoT.

Conclusión

En su conjunto, el sistema implementado puede considerarse una solución integral, segura y funcional para entornos IoT de pequeño y mediano tamaño, destacando especialmente por:

- Su diseño modular y escalable.
- La robustez de su cifrado basado en claves de un solo uso.
- La fiabilidad de la comunicación includo ante fallos.
- La claridad en la visualización de resultados.

Todo ello convierte a este Trabajo de Fin de Grado en una propuesta sólida no solo a nivel académico, sino también con proyección hacia aplicaciones prácticas reales.

Capítulo 6

Conclusiones y Líneas futuras

6.1. Grado de consecución de objetivos

El presente Trabajo de Fin de Grado ha logrado cumplir de manera sobresaliente con los objetivos que se establecieron al inicio del proyecto. Desde la fase de conceptualización hasta la información práctica, cada uno de los elementos del sistema ha sido desarrollado, probado e integrado con éxito.

En particular, se ha logrado:

- **Diseñar e implementar un sistema funcional de comunicación segura para dispositivos IoT**, utilizando el protocolo D2OTP como núcleo de la seguridad punto a punto.
- **Desarrollar un firmware autónomo para el ESP32**, capaz de realizar la lectura de sensores, gestionar claves, cifrar datos y transmitirlos de forma periódica al servidor.
- **Construir una librería criptográfica propia (D2OTP_completa.cpp)**, basada en técnicas de cifrado con autenticación (AES-GCM), y adaptada a las limitaciones del entorno embebido.
- **Implementar un gestor de red capaz de distribuir claves de forma segura**, atender solicitudes simultáneas de dispositivos y servidores, y almacenar credenciales de forma persistente.
- **Desarrollar un servidor central robusto**, con capacidad de recepción de datos UDP cifrados, de hacer un descifrado autenticado, y de almacenar estructuradamente las medidas en tiempo real.
- **Diseñar una aplicación web de visualización**, basada en FastAPI y tecnologías web modernas, con interfaz sencilla y visualmente clara para representar las lecturas recogidas.

Además, todos los módulos fueron integrados satisfactoriamente en una arquitectura completa, desplegada en una red local controlada, donde se realizaron diversas pruebas con resultados plenamente satisfactorios (ver Capítulo 5).

En resumen, el sistema ha demostrado ser técnicamente viable, operativo y seguro, cumpliendo con creces los objetivos establecidos y proporcionando una base sólida para futuras expansiones o adaptaciones.

6.2. Aportaciones del proyecto

Este proyecto ofrece varias contribuciones significativas dentro del ámbito de la seguridad en el Internet de las Cosas (IoT), destacando tanto a nivel técnico como académico:

- **Aplicación real del One-Time Pad con contador**: En lugar de recurrir a soluciones tradicionales como TLS o MQTT sobre TCP/IP, este trabajo presenta una alternativa interesante para redes UDP. Se enfoca en la combinación de claves efímeras, autenticación de mensajes y sincronización de estado, todo sin necesidad de una conexión permanente.
- **Desarrollo desde cero de una librería criptográfica propia**: Adaptada a dispositivos con recursos limitados y totalmente integrada en el flujo de comunicación. Destaca por su ligereza, seguridad y facilidad de integración.
- **Diseño de una arquitectura distribuida modular** con una separación clara de las responsabilidades de cada componente (placa, gestor, servidor e interfaz), que facilita su mantenimiento, escalabilidad y análisis de seguridad.
- **Amplia documentación**: El trabajo incluye un análisis completo de diseño, implementación y pruebas, así como instrucciones precisas para su despliegue, asegurando que pueda ser replicado o extendido por terceros.

Estas contribuciones no solo mejoran el contexto específico del proyecto, sino que también pueden abrir la puerta a nuevas líneas de investigación y desarrollo en ámbitos académicos o industriales, enfocándose en la protección de datos sensibles generados por sensores conectados.

6.3. Líneas de trabajo futuro

Aunque el sistema desarrollado ha demostrado ser funcional, seguro y robusto en su configuración actual, existen diversas líneas de mejora y ampliación que podrían llevarse a cabo en desarrollos futuros:

- **Escalabilidad y despliegue en entornos reales**: Actualmente, el sistema ha sido validado en una red local con un número limitado de dispositivos. Sería muy interesante llevar a cabo un despliegue controlado en un entorno real que incluya múltiples nodos, diferentes ubicaciones y una red Wi-Fi compartida. Esto nos permitiría evaluar la estabilidad y el rendimiento bajo condiciones más exigentes.
- **Migración a IPv6 o redes LPWAN**: Se podría extender el protocolo para ser compatible con otras tecnologías de red propias del IoT como LoRaWAN,

NB-IoT o incluso enlaces sobre IPv6, permitiendo su uso en aplicaciones de monitorización rural o industrial remota.

- **Integración con bases de datos no relacionales**: Aunque en este momento las lecturas se guardan en archivos estructurados, una evolución natural sería integrarlas con sistemas de almacenamiento como MongoDB, InfluxDB o Firebase. Esto facilitaría el análisis temporal, la trazabilidad y el acceso externo a través de API REST o sockets.
- **Autenticación y autorización extendidas**: En lugar de depender únicamente del gestor de claves, podríamos considerar la integración de un sistema adicional de control de acceso que utilice certificados digitales, OAuth2 o tokens JWT. Esto añadiría una capa extra de seguridad, especialmente en entornos más sensibles.
- **Hardening de la librería criptográfica**: Aunque el uso de AES-GCM es bastante adecuado, se podrían implementar algunas mejoras, como la adición de mecanismos de doble factor criptográfico en la derivación de claves OTP.
- **Actualización remota del firmware (OTA)**: Implementar una lógica de actualización segura desde el servidor central permitiría distribuir nuevas versiones del firmware sin necesidad de la intervención manual. Este es un aspecto fundamental para asegurar la mantenibilidad de las redes IoT a largo plazo.
- **Mejora de la interfaz de monitorización**: Aunque la web desarrollada es funcional, sería interesante considerar el uso de frameworks más avanzados como React o Vue. Esto podría ayudarnos a crear interfaces que sean más intuitivas y visualmente atractivas, incorporando elementos gráficos interactivos o alertas en tiempo real que los usuarios puedan configurar a su gusto.
- **Sistemas de alerta temprana**: Se podría implementar un sistema que detecte automáticamente valores anómalos en las mediciones de los sensores y envíe notificaciones a través de, por ejemplo, correo electrónico o SMS. Esto podría ser especialmente valioso en situaciones críticas como la agricultura, la salud ambiental o el control industrial.

Todas estas mejoras no solo ayudan a extender la vida útil del sistema, sino que también facilitan su uso en situaciones reales, tanto en el ámbito académico como en el empresarial. Esto proporciona una base sólida para futuras investigaciones, productos o soluciones.

6.4. Reflexión personal

A lo largo de este Trabajo de Fin de Grado, he tenido la increíble oportunidad de enfrentar uno de los retos más ambiciosos y enriquecedores de toda mi carrera. Partiendo de los conocimientos que adquirí en asignaturas como Redes de Computadores, Sistemas Operativos y Seguridad de las Tecnologías de la Infor-

mación, decidí aplicar estos conceptos a un proyecto real, tangible y útil, donde se cruzan múltiples disciplinas: electrónica, programación embebida, criptografía, desarrollo web y gestión de sistemas.

El camino no ha estado libre de dificultades. Algunos de los mayores desafíos han sido asegurar que los dispositivos se sincronizan correctamente, garantizar la integridad de los datos transmitidos por UDP, diseñar un protocolo propio adaptado al hardware disponible y depurar errores complejos que involucraban múltiples elementos al mismo tiempo (firmware, servidor, gestor, red). En más de una ocasión, encontrar la raíz de un fallo me llevó varias horas de análisis, sin poder usar herramientas de depuración tradicionales.

Sin embargo, cada obstáculo se convirtió en una oportunidad de aprendizaje. Gracias a estas experiencias, he fortalecido habilidades como la escritura de código rubusto en C++ para entornos con recursos limitados, la integración de sistemas heterogéneos y la planificación metódica de un proyecto complejo. Además, este trabajo me ha permitido experimentar de primera mano el valor de la modularidad, la documentación rigurosa y la validación continua a través de pruebas reales.

Me siento especialmente orgulloso de haber desarrollado desde cero un sistema funcional completo, abarcando desde el hardware hasta la interfaz de usuario, y de haberlo hecho con un enfoque en la seguridad y la eficiencia. Para mí, el hecho de que todas las piezas encajen y funcionen como un sistema distribuido coherente es el mayor logro de este proyecto.

En definitiva, este TFG no solo marca el final de una etapa académica, sino que también abre la puerta a una nueva fase profesional donde espero seguir creando sistemas que sean seguros, eficientes y que realmente hagan la diferencia. He descubierto un gran mundo del IoT y la ciberseguridad y tengo ganas de seguir aprendiendo sobre este campo en los próximos años.

Capítulo 7

Análisis del impacto

7.1. Impacto social y ético

El desarrollo de un sistema de comunicación seguro y eficiente para dispositivos IoT, como el que se presenta en este Trabajo de Fin de Grado, tiene importantes implicaciones sociales y éticas que no podemos pasar por alto, especialmente considerando nuestra creciente dependencia de tecnologías conectadas en áreas como la salud, la agricultura, la industria y los hogares inteligentes.

Desde una perspectiva social, el uso del protocolo D2OTP asegura la protección de la privacidad de los datos que transmiten los sensores embebidos, incluso en situaciones delicadas. En entornos como la domótica, donde se controlan temperaturas, niveles de humedad o incluso la presencia de personas en ciertas habitaciones, es crucial que estos datos no puedan ser interceptados o manipulados. Esto no solo protege la intimidad de los usuarios, sino que también previene posibles violaciones de sus derechos fundamentales. La importancia de esta protección se hace aún más evidente si consideramos la falta de atención que, históricamente, han recibido muchos dispositivos IoT en términos de seguridad por parte de varios fabricantes y desarrolladores.

Desde un punto de vista ético, este proyecto fomenta una cultura de responsabilidad tecnológica. En lugar de enfocarse únicamente en la eficiencia o el rendimiento del sistema, se ha priorizado la integridad de las comunicaciones y la autenticación de los participantes. Se ha elegido un enfoque sólido de cifrado punto a punto con claves efímeras, lo que reduce el riesgo de suplantaciones o accesos no autorizados. Este tipo de enfoque es valioso desde el punto de vista ético, ya que coloca al usuario final en el centro del diseño, protegiendo sus intereses incluso frente a atacantes con habilidades técnicas avanzadas.

Además, el código fuente del sistema está diseñado de manera clara y accesible, lo que facilita tanto la auditoría externa como la posibilidad de que la comunidad académica o profesional lo mejore. Este enfoque en la transparencia y la calidad del software se alinea con los principios de ética tecnológica promovidos por organizaciones como la IEEE y la ACM.

Por último, es importante mencionar que el sistema que hemos desarrollado podría ser la base para soluciones escalables de monitoreo en áreas desfavorecidas, incluso con una infraestructura mínima. Al funcionar en redes locales sin necesidad de una conexión constante a internet, puede facilitar la instalación de sensores ambientales o de salud en lugares donde la conectividad es escasa, ayudando así a cerrar la brecha digital de manera responsable.

7.2. Impacto medioambiental

En términos medioambientales, el impacto directo de este trabajo es bastante bajo, ya que el consumo energético de los dispositivos utilizados (como el ESP32 y los sensores DHT22) y del servidor central es mínimo y se ha optimizado con medidas de eficiencia, como el envío periódico con espera activa. Sin embargo, se han tomado decisiones conscientes durante el desarrollo para reducir la huella ecológica del sistema.

Por un lado, optar por el protocolo UDP en redes locales disminuye notablemente la cantidad de datos redundantes que se transmiten, así como la necesidad de retransmisiones pesadas que son típicas en protocolos como TCP. Esto no solo ahorra energía en el dispositivo, sino también en la infraestructura de red. Además, el diseño ligero del firmware evita operaciones innecesarias en el microcontrolador, manteniendo el procesador en modo de reposo cuando no se requieren envíos.

Asimismo, los materiales utilizados (placas ESP32, sensores digitales, protoboards reutilizables) han sido elegidos teniendo en cuenta su bajo coste y durabilidad. Esto promueve la reutilización de componentes y ayuda a reducir la generación de residuos electrónicos. De hecho, el mismo dispositivo puede reconfigurarse para diferentes escenarios de prueba, evitando así la obsolescencia prematura.

En cuanto al software, tanto el gestor como el servidor están diseñados para funcionar en sistemas Linux de bajo consumo, como un servidor doméstico o incluso una Raspberry Pi, lo que permite aprovechar equipos ya existentes sin necesidad de comprar nuevo hardware. Igualmente, la interfaz web se ha diseñado sin dependencias pesadas y está optimizada para cargar rápidamente en navegadores estándar, lo que reduce el coste energético en el cliente.

En resumen, este trabajo demuestra que es posible desarrollar soluciones IoT seguras sin sacrificar la sostenibilidad ambiental, integrando consideraciones ecológicas desde las primeras etapas del diseño hasta su implementación final.

7.3. Impacto económico y técnico

Desde una perspectiva económica, el sistema que se presenta en este TFG resalta por su viabilidad y su bajo coste de implementación. Esto lo convierte en una opción especialmente atractiva para situaciones donde los recursos son escasos,

como en instituciones educativas, áreas rurales o pequeñas organizaciones que necesitan una infraestructura básica para una monitorización segura.

La elección del hardware se ha hecho con un enfoque claro en la economía y la eficiencia. Utilizar microcontroladores ESP32-C3 con conectividad Wi-Fi integrada, que se pueden encontrar en el mercado por menos de 5€, reduce considerablemente la inversión inicial en comparación con otras plataformas de desarrollo más complejas. El sensor de temperatura y humedad DHT22 (AM2302), que también es económico y cuenta con una amplia documentación, ofrece datos fiables sin requerir una infraestructura energética o computacional elevada. Otros materiales utilizados, como cables Dupont, protoboard y una resistencia de $10\,\mathrm{k}\Omega$, son reutilizables y de bajo coste, lo que ayuda a la sostenibilidad del proyecto.

En lo que respecta al software, todas las tecnologías empleadas son de código abierto, lo que elimina por completo los costes de licenciamiento. El entorno de desarrollo Arduino IDE, el framework ESP-IDF, la librería `mbedTLS`, el backend desarrollado en Python con FastAPI, y la interfaz web basada en HTML, CSS y JavaScript, permiten una implementación completa sin barreras económicas. Esto es especialmente relevante para instituciones públicas o proyectos comunitarios que no cuentan con financiación significativa. Además, el sistema puede ser desplegado en equipos modestos, como una Raspberry Pi o una máquina virtual en Linux, sin necesidad de contratar servidores en la nube de terceros, lo que reduce notablemente los gastos operativos.

Desde un enfoque técnico, el sistema que proponemos es una solución integral, modular y segura para la transmisión de datos en entornos locales de IoT. Hemos implementado mecanismos de cifrado simétrico basados en One-Time Pad (OTP) con contador, lo que garantiza la confidencialidad incluso en redes que no son del todo seguras. Esta elección no solo protege contra ataques de repetición o intercepción, sino que también lo hace con un consumo computacional que se ajusta a los recursos limitados de los microcontroladores.

El diseño distribuido, que incluye dispositivos, un gestor de claves y un servidor de almacenamiento, refuerza la escalabilidad del sistema. Puedes añadir más sensores sin necesidad de cambiar la arquitectura, manteniendo la seguridad gracias a la gestión independiente de las claves. Además, la estructura JSON de los mensajes intercambiados facilita su análisis y depuración, algo fundamental en entornos reales donde la trazabilidad es clave.

En lo que respecta al mantenimiento técnico, el sistema ha sido diseñado para ser modular y fácilmente ampliable. Por ejemplo, la web de monitorización puede adaptarse rápidamente a nuevas visualizaciones o tipos de sensores simplemente ajustando los endpoints de la API y la lógica del frontend. El código del backend está bien estructurado y desacoplado, lo que permite su portabilidad a otros lenguajes o frameworks si se desea en el futuro.

Por último, es importante destacar que este TFG no solo ha demostrado ser técnicamente viable, sino que también ha establecido una base sólida para futuros desarrollos. La robustez del protocolo D2OTP, la clara separación de respon-

sabilidades entre componentes y el uso de tecnologías ampliamente adoptadas aseguran que esta solución pueda ser mantenida, reutilizada y ampliada con facilidad por otros desarrolladores o investigadores.

Bibliografía

[1] R. Hat. «¿Qué es el Internet de las cosas (IoT)?» (2023), dirección: https://www.redhat.com/es/topics/internet-of-things/what-is-iot (visitado 28-05-2025).

[2] R. Fernández. «El Internet de las cosas (IoT) - Datos estadísticos». (2025), dirección: https://es.statista.com/temas/6976/el-internet-de-las-cosas-iot/#topFacts (visitado 28-05-2025).

[3] Zscaler. «¿Qué es la seguridad de IoT?» (2025), dirección: https://www.zscaler.com/es/zpedia/what-iot-security (visitado 28-05-2025).

[4] Cloudflare. «¿Qué es la seguridad del IoT? | Seguridad de los dispositivos IoT». (2025), dirección: https://www.cloudflare.com/es-es/learning/security/glossary/iot-security (visitado 28-05-2025).

[5] E. Suárez. «Guía completa sobre la metodología de un TFG». (2025), dirección: https://www.expertouniversitario.es/blog/metodologia-tfg (visitado 28-05-2025).

[6] Mioti. «La importancia de la ciberseguridad en la era IoT». (2023), dirección: https://mioti.es/es/la-importancia-de-la-ciberseguridad-en-la-era-iot/ (visitado 29-05-2025).

[7] Fortinet. «What Is IoT Security? Challenges and Requirements». (2025), dirección: https://www.fortinet.com/resources/cyberglossary/iot-security (visitado 29-05-2025).

[8] Illumio. «Dominar la segmentación de redes: la clave para mejorar la ciberseguridad». (2024), dirección: https://www.illumio.com/es-mx/cybersecurity-101/network-segmentation (visitado 29-05-2025).

[9] IoTerop. «OSCORE, the End-to-End Security Protocol for IoT». (2024), dirección: https://medium.com/%40IoTerop/oscore-the-end-to-end-security-protocol-for-iot-4498cf3aa50a (visitado 29-05-2025).

[10] Wikipedia. «IPsec». (2024), dirección: https://en.wikipedia.org/wiki/IPsec (visitado 29-05-2025).

[11] Paessler. «¿Qué es MQTT?» (2024), dirección: https://www.paessler.com/es/it-explained/mqtt#:~:text=MQTT%20son%20las%20siglas%20de,cuanto%20al%20ancho%20de%20banda. (visitado 29-05-2025).

[12] Wallarm. «¿Qué es el protocolo CoAP? Significado y arquitectura». (2025), dirección: https://lab.wallarm.com/what/que-es-el-protocolo-coap-significado-y-arquitectura/?lang=es (visitado 29-05-2025).

[13] Medium. «OSCORE, the End-to-End Security Protocol for IoT». (2024), dirección: https://medium.com/%40IoTerop/oscore-the-end-to-end-security-protocol-for-iot-4498cf3aa50a (visitado 29-05-2025).

[14] D2OTP. «El protocolo D2OTP supone un avance importante en la Ciberseguridad de las Comunicaciones». (2023), dirección: https://www.d2otp.com (visitado 29-05-2025).

[15] CDN. «El protocolo D2OTP supone un avance importante en la Ciberseguridad de las Comunicaciones». (2024), dirección: https://cdn.website-editor.net/s/35ad348736434f699d12b811938c63bd/files/uploaded/El_Protocolo_D2OTP_para_Comunicaciones_Seguras_V2-540ee708.pdf?Expires=1750410458&Signature=Haif3lYno35n~P8hgBozX9kxCefGTuQ5v1xwnnJ1XP1SIx8AegJunByONAXZsBRtKVGLZmXnRZWMUaVUPyZsjm45C6kd85yPfv3EAj1PRZlIVefT5RhUhNOWcJvfK6v~XZeVvX5xQgW8AwNPuAjC9i85V9x6Jj-mKGtBM2GkBuE5C2PJcuchRronM0AuN42So32tQaHiDV-x6xgYRwDNVtAAj-wgnU4V2H8kJJuOat~VCWGhQq39Nx62d0IJsYn-yvJGcyPpUPRJ3lnOoER32dYclclg-Y6QU-u1A9i-vC9CD1-HI4Ukbqmhg7HZUL~VkZhEr7R8wi3TQN2RQEQ7gA__&Key-Pair-Id=K2NXBXLF010TJW (visitado 29-05-2025).

[16] LinkedIn. «¿Cuál es el papel de la criptografía en la seguridad de los dispositivos IoT?» (2024), dirección: https://www.linkedin.com/advice/1/what-role-cryptography-securing-iot-devices-7xldc?lang=es (visitado 29-05-2025).

[17] U. T. Nacional. «Principales técnicas criptográficas aplicadas a la seguridad de la información en IoT: una revisión sistemática». (2023), dirección: https://portal.amelica.org/ameli/journal/266/2663842005/html (visitado 29-05-2025).

[18] E. Systems. «ESP32 Series Datasheet Version 4.9». (2025), dirección: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (visitado 29-05-2025).

[19] L. Aosong Electronics Co. «Digital-output relative humidity temperature sensor/module DHT22 (DHT22 also named as AM2302)». (2025), dirección: https://cdn.sparkfun.com/assets/f/7/d/9/c/DHT22.pdf (visitado 29-05-2025).

[20] Arduino. «Arduino Documentation». (2025), dirección: https://docs.arduino.cc (visitado 29-05-2025).

[21] E. Systems. «ESP-IDF Programming Guide». (2025), dirección: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html (visitado 29-05-2025).

[22] Prometec. «Guía a la programación de Arduino III – Librerías». (2023), dirección: https://www.prometec.net/funciones-iii (visitado 29-05-2025).

[23] cloudflare. «¿Qué es el UDP?» (2023), dirección: https://www.cloudflare.com/es-es/learning/ddos/glossary/user-datagram-protocol-udp (visitado 29-05-2025).

[24] S. OverFlow. «UDP broadcast for IoT discovery using Raspberry PI and public wifi». (2022), dirección: https://stackoverflow.com/questions/69339570/udp-broadcast-for-iot-discovery-using-raspberry-pi-and-public-wifi (visitado 29-05-2025).

[25] J. Vega. «Protocol for secure communications D2OTP». (2025), dirección: https://www.b2match.com/e/feindef-2025/opportunities/UGFydGljaXBhdGlvbk9wcG9ydHVuaXR5OjE2MDc1Nw%3D%3D (visitado 29-05-2025).

[26] Kiteworks. «Todo lo que Necesitas Saber sobre el Cifrado AES-256». (2025), dirección: https://www.kiteworks.com/es/glosario-riesgo-cumplimiento/aes-256-encryption (visitado 29-05-2025).

[27] Keyfactor. «Gestione y automatice IoT identidades de dispositivos». (2025), dirección: https://www.keyfactor.com/es/products/command-iot (visitado 30-05-2025).

[28] SanDoRobotics. «Super Mini Tarjeta de Desarrollo WiFi + Bluetooth — ESP32-C3». (2025), dirección: https://sandorobotics.com.mx/producto/hs5347/#:~:text=La%20tarjeta%20ESP32%2DC3%20es,WiFi%204%20y%20Bluetooth%205.0. (visitado 30-05-2025).

[29] TailwindCSS. «Rapidly build modern websites without ever leaving your HTML». (2025), dirección: https://tailwindcss.com (visitado 30-05-2025).

[30] Chart.js. «Chart.js: Simple yet flexible JavaScript charting library for the modern web». (2025), dirección: https://www.chartjs.org/ (visitado 30-05-2025).

Anexos

Apéndice A

Código fuente de la librería D2OTP

```
#include "D2OTP_completa.h"
#include <string.h>

#ifdef ESP32
  #include "mbedtls/gcm.h"
  #include "mbedtls/sha256.h"
#else
  #include <openssl/evp.h>
  #include <openssl/sha.h>
#endif

  // Método para calcular el SHA256
  void D2OTP::calcularSHA256(const uint8_t *input, size_t
      input_len, uint8_t *output) {
#ifdef ESP32
    mbedtls_sha256(input, input_len, output, 0);
#else
    SHA256(input, input_len, output);
#endif
  }

  // Método para encriptar
  int D2OTP::encrypt(const uint8_t *input, size_t input_len,
      uint8_t *output, size_t *output_len, uint8_t *tag, size_t
      tag_len, const uint8_t *iv, const uint8_t *key) {
#ifdef ESP32
        mbedtls_gcm_context ctx;
        mbedtls_gcm_init(&ctx);
        int res = mbedtls_gcm_setkey(&ctx,
           MBEDTLS_CIPHER_ID_AES, key, 256);
        if (res != 0)
```

```
28              return res;
29          res = mbedtls_gcm_crypt_and_tag(&ctx,
               MBEDTLS_GCM_ENCRYPT, input_len, iv, 12, nullptr, 0,
               input, output, tag_len, tag);
30          mbedtls_gcm_free(&ctx);
31          *output_len = input_len;
32          return res;
33
34  #else
35          EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
36          int len = 0;
37          *output_len = 0;
38          EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL,
               NULL);
39          EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, 12,
               NULL);
40          EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv);
41          EVP_EncryptUpdate(ctx, output, &len, input, input_len);
42          *output_len += len;
43          EVP_EncryptFinal_ex(ctx, output + len, &len);
44          *output_len += len;
45          EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, tag_len,
               tag);
46          EVP_CIPHER_CTX_free(ctx);
47          return 0;
48  #endif
49    }
50
51    // Método para desencriptar
52    int D2OTP::decrypt(const uint8_t *input, size_t input_len,
         uint8_t *output, size_t *output_len, const uint8_t *tag,
         size_t tag_len, const uint8_t *iv, const uint8_t *key) {
53  #ifdef ESP32
54          mbedtls_gcm_context ctx;
55          mbedtls_gcm_init(&ctx);
56          int res = mbedtls_gcm_setkey(&ctx,
               MBEDTLS_CIPHER_ID_AES, key, 256);
57          if (res != 0)
58              return res;
59          res = mbedtls_gcm_auth_decrypt(&ctx, input_len, iv, 12,
               nullptr, 0, tag, tag_len, input, output);
60          mbedtls_gcm_free(&ctx);
61          *output_len = input_len;
62          return res;
63
64  #else
65          EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
```

```
66          int len;
67          *output_len = 0;
68          EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL,
               NULL);
69          EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, 12,
               NULL);
70          EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv);
71          EVP_DecryptUpdate(ctx, output, &len, input, input_len);
72          *output_len += len;
73          EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_TAG, tag_len,
               (void *)tag);
74          int ret = EVP_DecryptFinal_ex(ctx, output + len, &len);
75          *output_len += len;
76          EVP_CIPHER_CTX_free(ctx);
77          return (ret > 0) ? 0 : -1;
78  #endif
79    }
```

Apéndice B

Código representativo del servidor UDP

B.1. Inicialización del servidor

```
if (!cargarConfigServidor()) {
    solicitarClaveServidor();
}

int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
sockaddr_in servidor{}, cliente{};
servidor.sin_family = AF_INET;
servidor.sin_port = htons(PUERTO_SERVIDOR);
servidor.sin_addr.s_addr = INADDR_ANY;
bind(sockfd, (sockaddr *)&servidor, sizeof(servidor));

cout << "Servidor activo en puerto " << PUERTO_SERVIDOR <<
    endl;

char buffer[1024];
socklen_t len = sizeof(cliente);
```

B.2. Análisis de mensajes

```
// Método para procesar el mensaje que le llega al servidor
void procesarMensaje(const json &j) {
      string id = j["id"];
      int contador = j["contador"];
      string payload = j["payload"];
      string tag = j["tag"];

      if (payload.length() % 2 != 0 || tag.length() != 32) {
            cerr << "Formato de payload o tag inválido\n";
```

```
10            return;
11        }
12
13        uint8_t cifrado[64], tagb[16];
14        hexToBytes(payload, cifrado);
15        hexToBytes(tag, tagb);
16
17        if (!clavePlacaRegistrada(id)) {
18            solicitarClavePlaca(id);
19        }
20
21        string claveHex = cacheClaves[id]["clave"];
22
23        if (claveHex.length() != 64) {
24            cerr << "Clave con longitud incorrecta en
                  cacheClaves para ID: "
25                 << id << endl;
26            return;
27        }
28
29        string nombre = cacheClaves[id]["nombre"];
30
31        uint8_t claveSesion[32];
32        for (int i = 0; i < 32; ++i)
33            claveSesion[i] =
34                (uint8_t)strtol(claveHex.substr(i * 2,
                    2).c_str(), nullptr, 16);
35
36        uint8_t iv[12] = {0};
37        memcpy(iv, &contador, min(sizeof(contador), sizeof(iv)));
38
39        uint8_t salida[128];
40        size_t out_len = 0;
41        int ok = D2OTP::decrypt(cifrado, payload.length() / 2,
              salida, &out_len,
42                                tagb, 16, iv, claveSesion);
43
44        if (ok == 0) {
45            string mensaje((char *)salida, out_len);
46            cout << "[" << nombre << "] " << mensaje << endl;
47            guardarDato(id, mensaje);
48        } else {
49            cerr << "Error al descifrar mensaje de " << id <<
                  endl;
50        }
51 }
```

B.3. Generación y envío del ACK cifrado

```
1    const char *respuesta = "OK";
2    sendto(sockfd, respuesta, strlen(respuesta), 0, (struct
        sockaddr *)&cliente, sizeof(cliente));
```

Apéndice C

Código representativo del firmware

C.1. Configuración inicial

```
String id = getChipId();
Serial.print("ID generado: ");
Serial.println(id);

EEPROM.get(POS_CLAVE, claveOTP);
EEPROM.get(POS_CONTADOR, contador);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Conectando WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}

gestorIP.fromString(GESTOR_IP);
servidorIP.fromString(SERVIDOR_IP);
udp.begin(12345);
```

C.2. Solicitud de clave

```
StaticJsonDocument<192> doc;
doc["tipo"] = "solicitud_clave_placa";
doc["id_origen"] = id;
doc["nonce"] = bytesToHex(nonce, 16);

String mensajeJson;
serializeJson(doc, mensajeJson);

```

```
udp.beginPacket(gestorIP, GESTOR_PUERTO);
udp.write((const uint8_t*)mensajeJson.c_str(),
    mensajeJson.length());
udp.endPacket();
```

C.3. Lectura del sensor DHT22

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

if (isnan(temp) || isnan(hum)) {
  Serial.println("Error al leer el sensor DHT22.");
  return;
}
```

C.4. Envío cifrado de datos y manejo del ACK

```
if (ok != 0) {
  Serial.println("Error al cifrar medida");
  return;
}

StaticJsonDocument<192> doc;
doc["id"] = id;
doc["contador"] = contador;
doc["payload"] = bytesToHex(cifrado, out_len);
doc["tag"] = bytesToHex(tag, 16);

String mensajeJson;
serializeJson(doc, mensajeJson);

udp.beginPacket(servidorIP, SERVIDOR_PUERTO);
udp.write((const uint8_t*)mensajeJson.c_str(),
    mensajeJson.length());
udp.endPacket();

Serial.println("Medida enviada: " + datos);

unsigned long tiempoInicio = millis();
bool ackRecibido = false;
char bufferAck[10];

while (millis() - tiempoInicio < 2000) {
  int packetSize = udp.parsePacket();
  if (packetSize > 0) {
    udp.read(bufferAck, 10);
```

```
        bufferAck[packetSize] = '\0';
        if (strcmp(bufferAck, "OK") == 0) {
          Serial.println("ACK recibido del servidor");
          ackRecibido = true;
          break;
        }
      }
    }
```

C.5. Persistencia de clave y contador en EEPROM

```
  EEPROM.put(POS_CLAVE, claveOTP);
  EEPROM.put(POS_CONTADOR, contador);
  EEPROM.commit();
```

Apéndice D

Código representativo del gestor de red

D.1. Configuración inicial

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) {
    cerr << "Error creando socket\n";
    return;
}

sockaddr_in servidor{}, cliente{};
servidor.sin_family = AF_INET;
servidor.sin_port = htons(PUERTO_UDP);
servidor.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (sockaddr *)&servidor, sizeof(servidor)) < 0) {
    cerr << "Error haciendo bind\n";
    close(sockfd);
    return;
}

cout << "Gestor escuchando en puerto " << PUERTO_UDP << "\n";
```

D.2. Recuperación de claves desde fichero JSON

```
if (!dispositivos.contains(id_placa)) {
    cerr << "ID de placa no registrado: " << id_placa << endl;
    return;
}

string clavePlacaHex = dispositivos[id_placa]["clave"];
uint8_t clavePlaca[32];
```

```
for (int i = 0; i < 32; ++i)
    clavePlaca[i] = (uint8_t)strtol(clavePlacaHex.substr(i * 2,
        2).c_str(), nullptr, 16);
```

D.3. Respuesta al servidor o a la placa

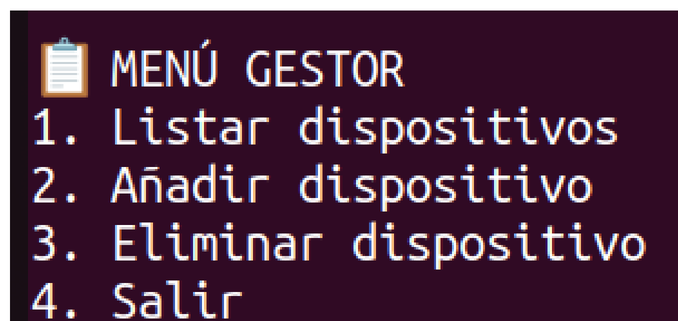
```
D2OTP::encrypt(clavePlaca, 32, claveCifrada, &len_out, tag, 16,
    iv, claveServidor);

json respuesta;
respuesta["clave_cifrada"] = binToHex(claveCifrada, len_out);
respuesta["tag"] = binToHex(tag, 16);
respuesta["nombre"] = dispositivos[id_placa]["nombre"];

string respuestaStr = respuesta.dump();

int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
sendto(sockfd, respuestaStr.c_str(), respuestaStr.length(), 0,
    (sockaddr*)&cliente, clienteLen);
close(sockfd);
```

D.4. Captura del menú del gestor



Apéndice E

Captura de interfaz

- Captura interfaz de la pantalla principal:

- Captura interfaz de la pantalla detalles:

Hora	Temperatura (°C)	Humedad (%)
21:12:08	29.3	29.3
21:11:38	29.3	29.3
21:11:09	29.3	29.4
21:10:39	29.3	29.2
21:10:09	29.3	29.1
21:09:39	29.2	29.2
21:09:09	29.3	29.2
21:08:39	29.2	29.3
21:06:33	29.1	29.6
21:06:03	29.1	29.7

Apéndice F

Archivo `dispositivos.json`


```
▼ 000024FFAC33E864:
    clave:      "2d6307d74b34beec63fc1477cff310c6ba523a9313602b5c51f6e8eb4a96710b"
    nombre:     "ESP32_Placa_1"
▼ 0000A8C6AC33E864:
    clave:      "91d33be866156219191016ebaee26db8948ecc18407d74d507b1590b1c30f11c"
    nombre:     "ESP32_Placa_2"
▼ SERVIDOR_TRABAJO:
    clave:      "cab6cdd76fbb543dbb72bdd63e38d8c0a3f7694c37a5da2487a6fa85d01cec31"
    nombre:     "Servidor"
```

Apéndice G

Archivo `config_inicial_servidor.json`

```
clave_compartida:    "cab6cdd76fbb543dbb72bdd63e38d8c0a3f7694c37a5da2487a6fa85d01cec31"
id_servidor:         "SERVIDOR_TRABAJO"
```

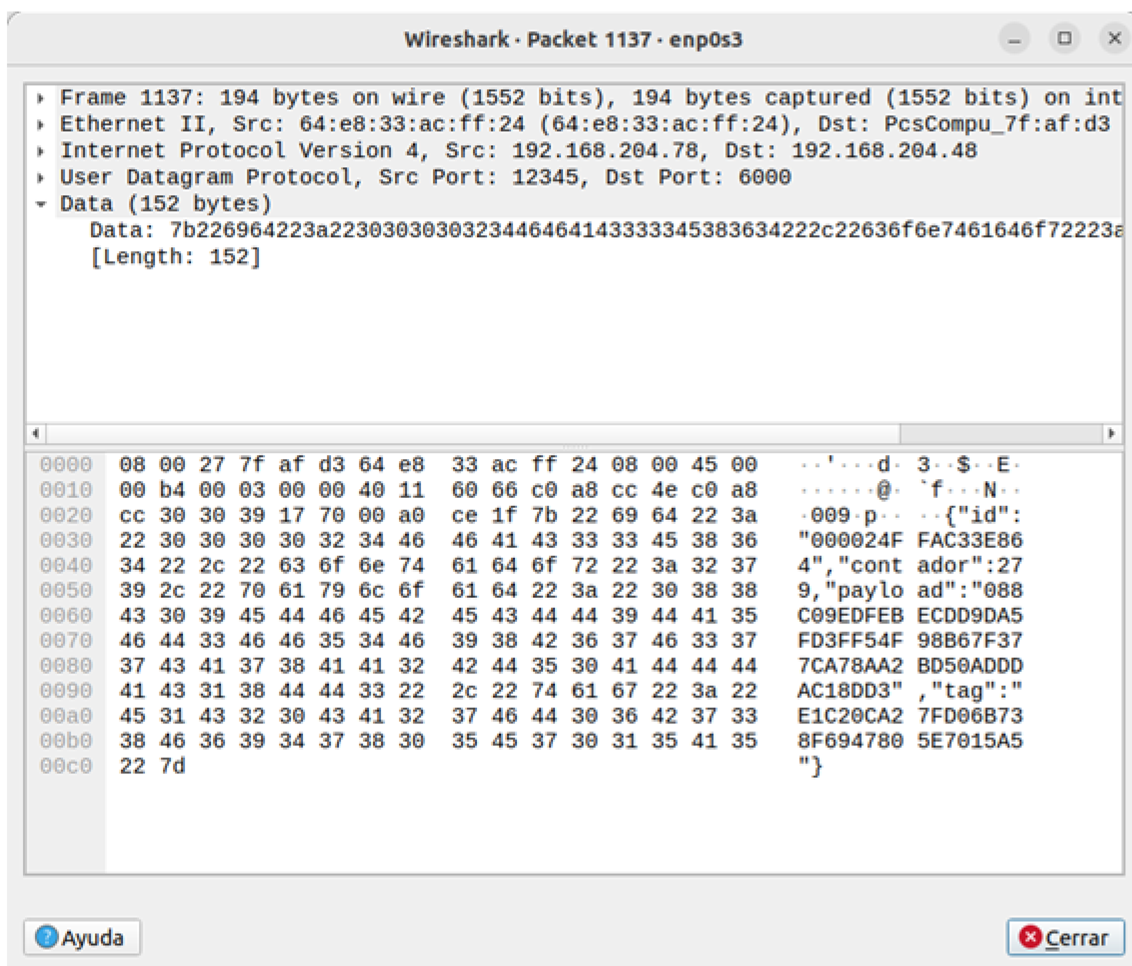
Apéndice H

Archivo `datos.json`


```
▼ dispositivos:
  ▼ 000024FFAC33E864:
    ▶ actual:       { hora: "21:14:55", humedad: "26.4", temperatura: "29.8" }
    ▼ historial:
      ▶ 0:          { hora: "21:10:40", humedad: "25.9", temperatura: "29.8" }
      ▶ 1:          { hora: "21:10:55", humedad: "26.1", temperatura: "29.8" }
      ▶ 2:          { hora: "21:11:25", humedad: "26.5", temperatura: "29.8" }
      ▶ 3:          { hora: "21:11:55", humedad: "26.9", temperatura: "29.8" }
      ▶ 4:          { hora: "21:12:25", humedad: "27.1", temperatura: "29.8" }
      ▶ 5:          { hora: "21:12:55", humedad: "27.0", temperatura: "29.8" }
      ▶ 6:          { hora: "21:13:25", humedad: "27.8", temperatura: "29.8" }
      ▶ 7:          { hora: "21:13:55", humedad: "26.2", temperatura: "29.9" }
      ▶ 8:          { hora: "21:14:25", humedad: "26.8", temperatura: "29.8" }
      ▶ 9:          { hora: "21:14:55", humedad: "26.4", temperatura: "29.8" }
    nombre:         "ESP32_Placa_1"
  ▼ 0000A8C6AC33E864:
    ▶ actual:       { hora: "21:15:08", humedad: "29.3", temperatura: "29.4" }
    ▼ historial:
      ▶ 0:          { hora: "21:10:39", humedad: "29.2", temperatura: "29.3" }
      ▶ 1:          { hora: "21:11:09", humedad: "29.4", temperatura: "29.3" }
      ▶ 2:          { hora: "21:11:38", humedad: "29.3", temperatura: "29.3" }
      ▶ 3:          { hora: "21:12:08", humedad: "29.3", temperatura: "29.3" }
      ▶ 4:          { hora: "21:12:38", humedad: "29.4", temperatura: "29.3" }
      ▶ 5:          { hora: "21:13:08", humedad: "29.3", temperatura: "29.4" }
      ▶ 6:          { hora: "21:13:38", humedad: "29.3", temperatura: "29.4" }
      ▶ 7:          { hora: "21:14:08", humedad: "29.2", temperatura: "29.4" }
      ▶ 8:          { hora: "21:14:38", humedad: "29.2", temperatura: "29.4" }
      ▶ 9:          { hora: "21:15:08", humedad: "29.3", temperatura: "29.4" }
    nombre:         "ESP32_Placa_2"
```

Apéndice I

Captura Wireshark



Apéndice J

Turniting

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

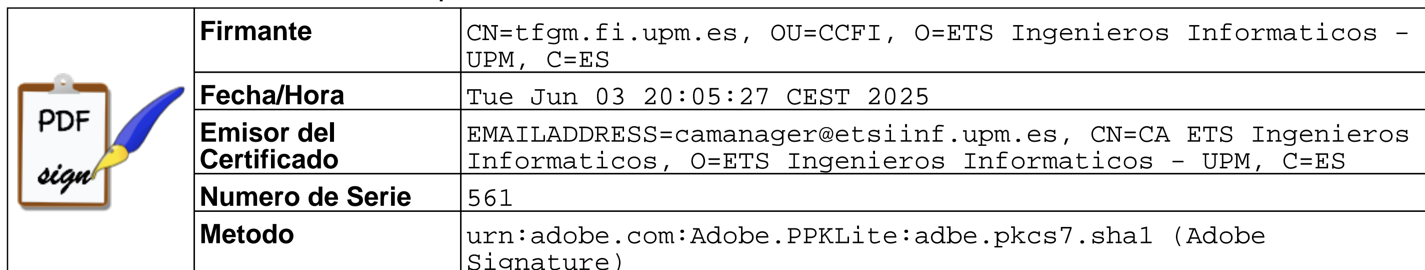
Submission author:	ADRIAN RODRIGUEZ SERRANO
Assignment title:	Turnitin Memoria Final
Submission title:	TFG_Adrián_Rodríguez.pdf
File name:	14574_ADRIAN_RODRIGUEZ_SERRANO_TFG_Adrián_Rodríguez...
File size:	981.87K
Page count:	97
Word count:	21,900
Character count:	117,447
Submission date:	03-Jun-2025 01:35PM (UTC+0200)
Submission ID:	2691270927

2% Standard

Similarity

Sources

Este documento esta firmado por



Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Tue Jun 03 20:05:27 CEST 2025
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)