

Tesinas de Grado

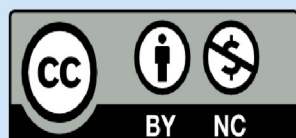
Chazarreta, Facundo Ariel

Implementación de los aspectos de seguridad en sistemas IoT con aplicación en el monitoreo de cultivo y calidad de agua

2023

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática



Esta obra está bajo una Licencia Creative Commons.

Atribución – no comercial 4.0

<https://creativecommons.org/licenses/by-nc/4.0/>

Documento descargado de RID - UNAJ Repositorio Institucional Digital de la Universidad Nacional Arturo Jauretche

Cita recomendada:

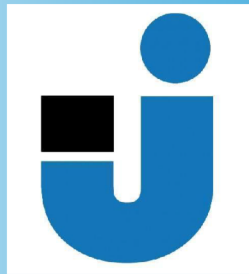
Chazarreta, F. A. (2023). *Implementación de los aspectos de seguridad en sistemas IoT con aplicación en el monitoreo de cultivo y calidad de agua* [Tesis de grado, Universidad Nacional Arturo Jauretche]. Disponible en RID - UNAJ Repositorio Institucional Digital UNAJ

<https://biblioteca.unaj.edu.ar/rid-unaj-repositorio-institucional-digital-unaj>

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carreta de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

*Implementación de los aspectos de seguridad
en sistemas IoT con aplicación en el monitoreo
de cultivo y calidad de agua*

Chazarreta Facundo Ariel

Florencio Varela, Agosto, 2023

ESTUDIANTE

Apellido y Nombres: *Chazarreta Facundo Ariel*

Correo electrónico: *Chazita9@gmail.com*

**ORGANIZACIÓN DONDE SE REALIZA LA PRÁCTICA PROFESIONAL
SUPERVISADA**

Nombre de la institución: *Universidad Nacional Arturo Jauretche*

Dirección: *Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina*

Teléfono: *+54 11 4275-6100*

Sector: *Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones
de interés social, Instituto de Ingeniería y Agronomía*

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: *Prof. Mg. OSIO, Jorge*

Correo electrónico: josio@unaj.edu.ar

DOCENTE SUPERVISOR

Apellido y Nombres: *Ing. Salvatore Juan Eduardo*

Correo electrónico: jsalvatore@unaj.edu.ar

COORDINADOR DE LA CARRERA DE INGENIERÍA INFORMÁTICA

Apellido y Nombres: *Dr. Ing. Morales, Martin*

Correo electrónico: martin.morales@unaj.edu.ar

Resumen

Actualmente, los sistemas de Internet de las Cosas (IoT) han experimentado un avance significativo, expandiéndose en diversas áreas como la domótica, la industria y la agricultura. Una de las facetas más prominentes es la capacidad de estos sistemas para mejorar la eficiencia en procesos complejos y ofrecer soluciones innovadoras en la vida diaria. Sin embargo, con este crecimiento, también se ha observado una mayor vulnerabilidad ante ciberataques, lo que hace imperativo el fortalecimiento de las medidas de seguridad en la arquitectura IoT. En este contexto, se están explorando y desarrollando esquemas avanzados de seguridad para garantizar la integridad, confidencialidad y disponibilidad de la información en los sistemas de IoT. Estos esfuerzos incluyen la aplicación de protocolos seguros, tecnologías de cifrado, autenticación y otros mecanismos de protección que son vitales para la operación segura y confiable de los dispositivos y redes de IoT.

Palabras Clave: IoT(Internet de las cosas), Agricultura, Seguridad, Autenticación, Dispositivos, Protocolos seguros, Tecnologías de cifrado.

Abstract

Currently, the Internet of Things (IoT) systems have experienced significant advancement, expanding into various areas such as home automation, industry, and agriculture. One of the most prominent facets is these systems' ability to improve efficiency in complex processes and provide innovative solutions in daily life. However, with this growth, there has also been an increased vulnerability to cyberattacks, making the strengthening of security measures in IoT architecture imperative. In this context, advanced security schemes are being explored and developed to ensure the integrity, confidentiality, and availability of information in IoT systems. These efforts include the implementation of secure protocols, encryption technologies, authentication, and other protection mechanisms that are vital for the safe and reliable operation of IoT devices and networks.

Keywords: IoT (Internet of Things), Agriculture, Security, Authentication, Devices, Secure Protocols, Encryption Technologies.

Dedicatorias y Agradecimientos

Índice General

Resumen	2
Abstract	4
Dedicatorias y Agradecimientos	5
Índice General	6
Índice de figuras	8
Marco teórico	11
2.1 Introducción	11
2.2 Objetivos	11
Caso de estudio	12
3.1 Internet de las cosas	12
3.2 Criptografía	14
Conceptos, Tecnologías y Herramientas	16
4.1 Motor de base de datos (MySQL)	17
4.2 Certificación de seguridad Web TLS y SSL	18
4.2.1 Protocolo “HandShake” TLS	20
4.3 Broker (EMQX)	21
4.3.1 Dashboard (Emqx)	23
4.3.2 Autenticación con MySQL(Emqx)	24
4.3.3 MQTTX (Emqx)	28
4.4 Node Red	29
4.4.1 Configuración de Node Red	31
4.4.1.1 Creación de un usuario	31
4.4.1.2 Distribucion de Nodos para el procesamiento de datos	32
4.4.1.3 Utilización de un proxy inverso	33
4.5 Balanceador de Carga y Proxy Inverso (NginX)	34
4.5.1 ¿Por qué Nginx?	34
4.5.2 Proxy Inverso	36
4.5.3 Configuración del Firewall de Nginx para SSH	37
4.6 Arduino IDE	38
4.6.1 Compilación	39
4.6.2 Bibliotecas Arduino	40
4.7 ESP32 (Dispositivo programable con conectividad wifi)	41
4.7.1 ¿Porque el ESP32?	42
4.7.2 ESP32 Seguridad	44

4.8 Grafana (Dashboard)	45
4.8.1 ¿Porque Grafana?	47
4.8.2 Funcionamiento de Grafana	48
4.8.3 Grafana y MySQL	49
4.9 Librería mbedtls	50
4.10 Algoritmo AES	51
4.10.1 AES Encriptado	51
4.10.2 AES Decifrado	61
4.10.3 AES Ejemplo práctico	64
4.11 Algoritmo de cifrado XOR	66
4.12 Python	67
4.12.1 ¿Porque python?	68
4.12.2 Scripts Python	69
Implementación del algoritmo de cifrado	70
5.1 Puesta en funcionamiento del esquema	70
5.1.1 MySQL inicio	70
5.1.2 Emqx inicio	73
5.1.3 Ningx inicio	76
5.1.4 NodeRed inicio	76
5.1.5 Grafana inicio	77
5.2 Algoritmo de cifrado XOR e implementación de librería mbedtls	78
5.2.1 Algoritmo de cifrado XOR (Cifrado de datos)	78
5.2.2 Algoritmo de cifrado XOR (Descifrado de datos)	80
5.2.3 Implementación de librería mbedtls Cifrado	83
5.2.4 Implementación de librería mbedtls Descifrado	85
Resultados	87
6.1 Resultados del algoritmo XOR	87
6.2 Resultados del algoritmo AES	90
6.2.1 Análisis de los resultados del algoritmo AES	94
6.2.2 Análisis de soluciones y mejoras para AES	95
Conclusiones	97
Referencias bibliográficas	98

Índice de figuras

Figura 4.1 - Handshake Protocol -	
https://www.researchgate.net/figure/TLS-handshake-protocol_fig1_2980656051(2023)	21
Figura 4.2 - Funcionamiento del broker. En la izquierda los dispositivos que emiten los datos, en el centro el broker encargado de redirigir los mensajes(datos), en la izquierda los dispositivos finales que esperan leer los datos- Recuperado de https://www.emqx.io/docs/en/v3.0/(2023)	22
Figura 4.3 - Dashboard de EMQX - Fuente: Propia.....	23
Figura 4.4 - App MQTTX - Fuente: Propia.....	28
Figura 4.5 - MQTTX Script - Fuente: Propia.....	29
Figura 4.6 - Nodos Node Red - Fuente: Propia.....	33
Figura 4.7 - Esquema de funcionamiento de proxy inverso -	
https://www.herlitz.io/2022/03/27/running-optimizely-cms-behind-nginx-reverse-proxy(2023)	37
Figura 4.8 - Data source Grafana -Fuente: Propia.....	50
Figura 4.9 - Encriptado AES (Mensaje a hexadecimal) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	51
Figura 4.10 - Encriptado AES (Operación XORl) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	52
Figura 4.11 - Encriptado AES (Operación SubBytes) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	52
Figura 4.12 - Encriptado AES (Tabla S-BOX) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	53
Figura 4.13 - Encriptado AES (Operación ShiftRows) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	54
Figura 4.14 - Encriptado AES (Operación ShiftRows movimiento de línea) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	54
Figura 4.15 - Encriptado AES (Operación MixColumns) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	55
Figura 4.16 - Encriptado AES (Tabla State y M, Método Rijndael Galois Field) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	55
Figura 4.17 - Encriptado AES (Tabla S-BOX) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	56

Figura 4.18 - Encriptado AES (Tabla S-BOX) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	57
Figura 4.19 - Encriptado AES (Operacion XOR con Key1) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	57
Figura 4.20 - Encriptado AES (Operación Vuelta 2) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	58
Figura 4.21 - Encriptado AES (Operación Vuelta 3) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	58
Figura 4.22 - Encriptado AES (Operación Vuelta 4) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	58
Figura 4.23 - Encriptado AES (Operación Vuelta 9) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	59
Figura 4.24 - Encriptado AES (Resultado final) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	59
Figura 4.25 - Encriptado AES (Generación de claves) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	60
Figura 4.26 - Encriptado AES (Generación de claves) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	61
Figura 4.27 - Decifrado AES (XOR ($\oplus$) entre cada célula del state y de la clave 10 (key)) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	71
https://www.researchgate.net/figure/TLS-handshake-protocol_fig1_2980656051(2023)	62
Figura 4.28 - Decifrado AES (Tabla S-BOX) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	63
Figura 4.29 - Decifrado AES (Vuelta 9 proceso inverso) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	63
Figura 4.30 - Decifrado AES (Matriz de proceso inverso MixRows) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	64
Figura 4.31 - Decifrado AES (Operación final de decodificación del mensaje) -	
https://www.teoria.com/jra/aes/encrypt.html(2023)	64
Figura 5.1 - MySql (Running) - Fuente: Propia.....	70
Figura 5.2 - MySql (Conexion) - Fuente: Propia.....	71
Figura 5.3 - MySql (Base de datos) - Fuente: Propia.....	72
Figura 5.4 - MySql (Usuarios) - Fuente: Propia.....	73
Figura 5.5 - Emqx(Status) - Fuente: Propia.....	73

<i>Figura 5.6 - Emqx(Dashboard) - Fuente: Propia.....</i>	<i>74</i>
<i>Figura 5.7 - Puerto (8083) - Fuente: Propia.....</i>	<i>74</i>
<i>Figura 5.8 - Puerto (8084) - Fuente: Propia.....</i>	<i>74</i>
<i>Figura 5.9 - Conexión a WebSocket SSL - Fuente: Propia.....</i>	<i>75</i>
<i>Figura 5.10 - Conexión a WebSocket SSL (Exitosa) - Fuente: Propia.....</i>	<i>75</i>
<i>Figura 5.11 - Nginx inicio - Fuente: Propia.....</i>	<i>76</i>
<i>Figura 5.12 - Puerto (1880) - Fuente: Propia.....</i>	<i>76</i>
<i>Figura 5.13 - NodeRed inicio - Fuente: Propia.....</i>	<i>77</i>
<i>Figura 5.14 - Grafana (Running) - Fuente: Propia.....</i>	<i>77</i>
<i>Figura 5.15 - Script descifrado MQTTX - Fuente: Propia.....</i>	<i>82</i>
<i>Figura 5.16 - Script descifrado MQTTX (Puesto en funcionamiento) - Fuente: Propia.....</i>	<i>82</i>
<i>Figura 6.1 - Mensajes cifrado y originales (Algoritmo XOR) - Fuente: Propia.....</i>	<i>87</i>
<i>Figura 6.2 - Mensajes cifrado y originales (MQTTX) - Fuente: Propia.....</i>	<i>88</i>
<i>Figura 6.3 - Mensajes cifrado y originales (MQTTX) vs Monitor Serial - Fuente: Propia.....</i>	<i>89</i>
<i>Figura 6.4 - Mensajes cifrado y originales (Python) - Fuente: Propia.....</i>	<i>91</i>
<i>Figura 6.5 - Suscripción tópico (Emqx) - Fuente: Propia.....</i>	<i>92</i>
<i>Figura 6.6 - Datos recibidos (Emqx) - Fuente: Propia.....</i>	<i>92</i>
<i>Figura 6.7 - Datos recibidos (Grafana) - Fuente: Propia.....</i>	<i>93</i>
<i>Figura 6.8 - Datos recibidos (Grafana) - Fuente: Propia.....</i>	<i>94</i>

Marco teórico

2.1 Introducción

En la actualidad el incremento de los sistemas de IoT se hace cada vez más evidente, esto se puede observar a simple vista en todo tipo de áreas, como en dispositivos de uso cotidiano que posean una conexión a internet, por ejemplo los relojes inteligentes, heladeras, hornos, alarmas, etc, o la agricultura, para el riego y gestión inteligente de los recursos agrícolas, con el fin de mejorar la eficiencia en de una plantación.

Ahora la pregunta, ¿Qué es un sistema o arquitectura IoT?, la arquitectura de IoT es un sistema de elementos constituyentes, estructura de red, tecnologías de nube que funcionan de conformidad con los protocolos y estándares de seguridad establecidos. Este sistema se compone de diferentes capas. La capa física, que incorpora sensores para detectar, recopilar información sobre el entorno y detecta algunos parámetros físicos o reconoce otros objetos inteligentes en el entorno. La capa de red, que es responsable de la correlación con los dispositivos vinculados en la red y los servidores. Finalmente la capa de aplicación, es aquella que brinda servicios específicos de la aplicación al usuario.

Como se mencionó anteriormente el incremento de los sistemas IoT, en distintos tipos de áreas trae consigo el aumento de demanda para el mercado, haciendo de este mismo un punto de interés para diversos ciberataques, por eso la propuesta de la PPS consiste en investigar, desarrollar e implementar un esquema de seguridad en sistemas IoT, con aplicación en el monitoreo de cultivo y calidad de agua, con el fin de optimizar la utilización de los tanto los recursos hídricos como la plantación en sí.

2.2 Objetivos

El presente proyecto consiste en el desarrollo e implementación de los aspectos de seguridad en sistemas IoT con aplicación en el monitoreo de cultivo y calidad de agua. El mismo tiene como objetivo principal Implementar una solución que permita añadir seguridad a todo un esquema de comunicación IoT, desde los dispositivos electrónicos,

hasta los usuarios finales, mejorando la eficiencia en el tráfico de información cifrada en un ambiente seguro. Las metas previas para lograr el objetivo final, se detallan a continuación:

- Lograr el tráfico cifrado de la información a través de los servidores administrados
- Implementación de un algoritmo el cual permita mejorar la eficiencia en la transferencia de datos cifrados
- Demostrar la funcionalidad de un entorno IoT seguro, a través de dashboard, los cuales puedan ser accedidos por un usuario desde cualquier dispositivo con conexión a la red.

Caso de estudio

3.1 Internet de las cosas

El Internet de las cosas (Internet Of Things) es un concepto que se refiere a la interconexión de objetos cotidianos a través de internet, permitiéndoles recopilar y compartir datos de manera inteligente. Estos objetos, también conocidos como dispositivos o sensores, van desde electrodomésticos y vehículos hasta dispositivos médicos y sistemas industriales. La idea detrás del IoT es que estos objetos pueden comunicarse entre sí y con los humanos, lo que lleva a una mayor eficiencia, comodidad y automatización en varios aspectos de nuestra vida cotidiana.

La arquitectura del Internet de las cosas (IoT) se compone de diferentes elementos que colaboran para habilitar la conectividad y funcionalidad de los dispositivos IoT. Estos elementos incluyen los dispositivos IoT, que son los componentes físicos encargados de recopilar datos y realizar acciones; la capa de conectividad, que establece la conexión entre los dispositivos y las redes mediante tecnologías como Wi-Fi, Bluetooth o redes celulares; los gateways, que actúan como intermediarios y facilitan la comunicación entre los dispositivos IoT y la infraestructura de red; la plataforma IoT, que proporciona servicios y funcionalidades para administrar y controlar los dispositivos IoT, así como para recopilar y

analizar datos; la capa de nube, que almacena y procesa los datos generados por los dispositivos IoT, ofreciendo escalabilidad y recursos de cómputo; y finalmente, las aplicaciones, que utilizan los datos generados por los dispositivos IoT para ofrecer servicios y funciones específicas, como monitoreo, análisis de datos, automatización, entre otros. Estos componentes trabajan de manera conjunta para permitir la interconexión inteligente de los dispositivos, el procesamiento de datos y la entrega de servicios innovadores basados en IoT.

A lo largo de la historia, la evolución del IoT ha sido impulsada por avances tecnológicos en varias áreas. A continuación, se presentan los hitos importantes en su desarrollo:

1. **Inicios:** Aunque el término "Internet de las cosas" se acuñó en 1999, los fundamentos del IoT se remontan a la década de 1970 con la creación de las primeras redes de sensores y sistemas de control industrial.
2. **Avance de la tecnología de sensores:** La miniaturización y reducción de costos de los sensores fueron clave para la expansión del IoT. Estos sensores pueden medir y recopilar datos sobre variables físicas como temperatura, humedad, luz, movimiento, entre otros.
3. **Conectividad:** El desarrollo de tecnologías de comunicación inalámbrica, como el Wi-Fi, Bluetooth y las redes celulares, permitió la conexión de dispositivos IoT a internet sin la necesidad de cables.
4. **Big Data y la nube:** La capacidad de almacenar grandes volúmenes de datos en la nube y analizarlos de manera eficiente fue fundamental para la gestión de datos generados por los dispositivos IoT. Esto permitió extraer información valiosa y tomar decisiones basadas en datos.
5. **Inteligencia Artificial y Machine Learning:** La integración de algoritmos de IA y aprendizaje automático en el IoT ha permitido a los dispositivos IoT tomar decisiones en tiempo real y adaptarse a diferentes situaciones. La IA mejora la capacidad de analizar y procesar grandes cantidades de datos generados por los dispositivos IoT.

6. **Expansión en diversos sectores:** El IoT se ha extendido a múltiples industrias, como la salud, el transporte, la agricultura, la domótica, la fabricación y la logística. Los dispositivos IoT están siendo utilizados para mejorar la eficiencia energética, monitorear la salud de los pacientes, optimizar la producción industrial y mucho más.

Actualmente, el IoT continúa evolucionando y su adopción se está acelerando. Se espera que la cantidad de dispositivos conectados a internet siga creciendo exponencialmente en los próximos años. Además, la integración de tecnologías emergentes, como la 5G, la computación en el borde (edge computing) y la realidad aumentada, ampliará aún más las posibilidades del IoT.

Sin embargo, a medida que el IoT se vuelve masivo, también surgen desafíos relacionados con la seguridad, la privacidad y la interoperabilidad, la protección de los datos personales, la garantía de la confidencialidad y autenticidad de la información, se han convertido en preocupaciones clave en el entorno del IoT. Sin embargo, el IoT está transformando la forma en que interactuamos con el mundo y se espera que su influencia siga creciendo en los próximos años.

3.2 Criptografía

La criptografía es la ciencia de proteger y asegurar la información mediante técnicas de codificación y decodificación. Consiste en transformar los datos en un formato ilegible llamado "texto cifrado" (Codificación), que solo puede ser revertido a su forma original o "texto claro" utilizando una clave o algoritmo específico (Decodificación). Esta disciplina ha existido durante siglos y ha desempeñado un papel crucial en la protección de la comunicación y la confidencialidad de la información.

En la antigüedad, civilizaciones como Egipto y Roma utilizaron técnicas rudimentarias de cifrado para proteger la confidencialidad de sus mensajes militares y diplomáticos. Una de las técnicas más utilizadas fue la sustitución de letras, donde cada letra del mensaje original se reemplaza por otra letra o símbolo. Por ejemplo, en el cifrado de César, se desplazaban

las letras del alfabeto un número fijo de posiciones. De esta manera, la letra "A" podría reemplazarse por la letra "D", la "B" por la "E", y así sucesivamente.

Estas técnicas rudimentarias de cifrado ofrecían cierta seguridad en ese momento, pero con el avance de la criptoanálisis (campo de estudio y práctica que se ocupa de descifrar o romper sistemas criptográficos con el objetivo de obtener acceso no autorizado a la información), se hicieron más vulnerables. Fue en la Segunda Guerra Mundial cuando la criptografía dio un salto significativo en términos de desarrollo y sofisticación.

Un ejemplo destacado fue la máquina Enigma, utilizada por los Alemanes para cifrar sus comunicaciones. La máquina Enigma era una máquina de rotores electromecánicos que permitía realizar sustituciones y transposiciones en el texto. Cada vez que se presionaba una letra en el teclado, los rotores giraban, cambiando la sustitución de letras en cada pulsación. Esto generaba un texto cifrado altamente complejo y difícil de descifrar.

En la actualidad, se utilizan varios métodos para proteger la información y garantizar la seguridad de las comunicaciones, como por ejemplo:

- **Criptografía de clave simétrica:** Este método utiliza una única clave para cifrar y descifrar los datos. Algunos algoritmos populares son el Data Encryption Standard (DES), el Advanced Encryption Standard (AES) y el Triple DES (3DES).
- **Criptografía de clave asimétrica:** También conocida como criptografía de clave pública, utiliza un par de claves relacionadas: una clave pública y una clave privada. La clave pública se comparte libremente, mientras que la clave privada se mantiene en secreto. Los algoritmos comunes incluyen *RSA*, *Diffie-Hellman* y *ElGamal*.
- **Hashing:** No es exactamente un método de cifrado, pero se utiliza ampliamente en criptografía. Los algoritmos de hash, como MD5, SHA-1, SHA-256 y SHA-3, convierten datos de cualquier tamaño en un valor hash fijo. Son útiles para verificar la integridad de los datos y generar firmas digitales.

En la PPS se utilizará uno de los métodos más eficientes actualmente desarrollados en cuanto tránsito encriptado respecta (autenticación y cifrado en tránsito SSL/TLS). Secure

Sockets Layer (SSL) y su sucesor Transport Layer Security (TLS) son protocolos de seguridad utilizados para garantizar la confidencialidad y autenticidad de las comunicaciones en Internet. SSL/TLS se utiliza comúnmente en el protocolo HTTPS (HTTP seguro), que proporciona una capa adicional de seguridad para las comunicaciones web.

Autenticación: SSL/TLS utiliza certificados digitales para autenticar la identidad de los servidores y, en algunos casos, también de los clientes. Los certificados son emitidos por Autoridades de Certificación (AC) confiables y contienen información sobre la entidad certificada y su clave pública. El proceso de autenticación en SSL/TLS involucra una serie de intercambios de mensajes entre el cliente y el servidor para verificar la validez del certificado y establecer una conexión segura.

Cifrado en tránsito: Una vez que se ha establecido la autenticación, SSL/TLS proporciona cifrado para proteger los datos durante su transmisión. Utiliza algoritmos de cifrado simétrico para cifrar los datos y garantizar que solo el destinatario legítimo pueda descifrarlos. Además, se utilizan claves de sesión únicas generadas durante el proceso de conexión para proporcionar confidencialidad.

En general, SSL/TLS utiliza una combinación de algoritmos criptográficos, como algoritmos de clave simétrica (por ejemplo, AES) para el cifrado de datos y algoritmos de clave asimétrica (por ejemplo, RSA) para el intercambio de claves y autenticación.

Conceptos, Tecnologías y Herramientas

A continuación, se listan los conceptos, tecnologías y herramientas que se utilizaron en esta PPS para investigar, desarrollar e implementar:

- Motor de base de datos (BD o MySQL)
- Broker (Emqx)
- MQTTX
- Arduino IDE (Plataforma de código abierto)

- ESP32 (Dispositivo programable con conectividad wifi)
- Certificación de seguridad Web TLS y SSL
- Node Red
- Balanceador de Carga y Proxy Inverso (NginX)
- Grafana (Dashboard)
- Librería mbedtls.h
- Algoritmo de cifrado AES
- Algoritmo de cifrado XOR
- Python

4.1 Motor de base de datos (MySQL)

MySQL es un sistema de gestión de bases de datos relacional de código abierto que es utilizado por muchas empresas y organizaciones en todo el mundo. Este software es una herramienta esencial para almacenar y gestionar grandes cantidades de datos de manera eficiente y segura.

Se caracteriza por ser una base de datos de alto rendimiento, escalable y fácil de usar. Es capaz de manejar grandes volúmenes de datos, además de contar con una amplia variedad de herramientas de administración y monitoreo que facilitan la gestión de la base de datos.

MySQL utiliza el lenguaje SQL (Structured Query Language) para gestionar y manipular los datos almacenados en la base de datos. SQL es un lenguaje de programación que permite interactuar con la base de datos a través de consultas y comandos específicos. Esto facilita la tarea de realizar consultas complejas y obtener información precisa de manera rápida y sencilla.

Otra de las ventajas, es que también es compatible con una amplia variedad de sistemas operativos y lenguajes de programación, lo que lo hace una opción flexible y adaptable a cualquier entorno. Además, ofrece una alta disponibilidad y tolerancia a fallos mediante técnicas de replicación y clustering, lo que asegura que los datos estén siempre disponibles y no se pierdan en caso de fallos.

Respecto a la seguridad de MySQL, esta proporciona un sistema de autenticación, a través un sistema de control de acceso flexible y granular que permite a los administradores de bases de datos definir quién tiene acceso a qué datos y operaciones en la base de datos. La autorización se puede controlar a nivel de base de datos, tabla y columna.

Este motor de base de datos también proporciona funciones de cifrado para garantizar la privacidad y seguridad de los datos almacenados en la base de datos. Admite la encriptación de datos en reposo y en tránsito. Para la encriptación de datos en reposo, MySQL utiliza el cifrado de disco transparente, lo que significa que los datos se cifran automáticamente antes de almacenarse en el disco. Para la encriptación de datos en tránsito, se admite el cifrado de la conexión SSL/TLS.

En resumen, MySQL es un sistema de gestión de bases de datos poderoso, flexible, y muy seguro que se adapta a las necesidades de cualquier organización. Por eso para esta PPS se utilizará y se hará énfasis en las funcionalidades de seguridad de datos que posee MySQL, iniciando así nuestro esquema seguro sobre IoT.

4.2 Certificación de seguridad Web TLS y SSL

TLS (Transport Layer Security) y su predecesor, **SSL (Secure Sockets Layer)**, son protocolos criptográficos que proporcionan comunicaciones seguras a través de una red, comúnmente Internet.

La función principal de SSL y TLS es proporcionar una conexión segura entre el cliente y el servidor para proteger los datos sensibles en tránsito. Lo logran a través de técnicas de cifrado, donde los datos enviados entre el cliente y el servidor se codifican de tal manera que solo pueden ser descifrados por el destinatario previsto. Además, proporcionan autenticación y garantizan la integridad de los datos.

SSL fue desarrollado por Netscape en los años 90. La versión 3.0 de SSL fue la última versión del protocolo antes de que se reemplazara por TLS. TLS 1.0, publicado en 1999, se basó en SSL 3.0, con varias mejoras y cambios para aumentar la seguridad. Desde entonces, ha habido varias versiones de TLS, con la última versión (en el momento de escribir esto en septiembre de 2021) siendo TLS 1.3.

La evolución de SSL a TLS ha aportado numerosas mejoras en seguridad y eficiencia, entre las que se incluyen:

Negociación de cifrado más segura: Las versiones modernas de TLS utilizan una negociación de cifrado más segura para prevenir ciertos tipos de ataques.

Cifrado más fuerte: A lo largo de los años, tanto SSL como TLS han dejado de utilizar algoritmos de cifrado que se consideran inseguros y han adoptado algoritmos más fuertes.

Mejoras en la eficiencia: Las versiones modernas de TLS, como TLS 1.3, han simplificado el proceso de "handshake" para mejorar la eficiencia y reducir la latencia.

Las ventajas de usar SSL/TLS en la actualidad incluyen:

Seguridad de los datos: Los protocolos SSL y TLS protegen los datos del usuario de ser leídos o modificados mientras están en tránsito.

Autenticación: Estos protocolos también verifican la identidad de los servidores (y en algunos casos, los clientes), lo que ayuda a prevenir ataques de phishing y man-in-the-middle.

Confidencialidad del cliente: Al encriptar los datos en tránsito, SSL/TLS ayudan a proteger la privacidad de los clientes.

Conformidad: Para ciertas industrias y regiones, el uso de encriptación en tránsito es un requisito para cumplir con las normas y regulaciones de protección de datos.

Confianza del cliente: Muchos navegadores web indican visualmente cuando una conexión es segura mediante el uso de un candado en la barra de direcciones, lo que puede aumentar la confianza del cliente en el sitio web.

4.2.1 Protocolo “HandShake” TLS

El protocolo de 'handshake' (apretón de manos) TLS es la parte del protocolo que se encarga de establecer la conexión segura entre el cliente y el servidor. Su proceso es el siguiente:

1. **Hello del cliente:** El cliente envía un mensaje 'ClientHello' al servidor, que contiene la versión de TLS que puede utilizar, qué suites de cifrado y métodos de compresión admite, y un número aleatorio generado por el cliente.
2. **Hello del servidor:** El servidor responde con un mensaje 'ServerHello', que especifica la versión de TLS, la suite de cifrado y el método de compresión que utilizarán. También incluye un número aleatorio generado por el servidor.
3. **Certificado del servidor:** A continuación, el servidor envía su certificado al cliente para autenticarse. Este certificado contiene la clave pública del servidor y es emitido por una Autoridad de Certificación (CA) que el cliente confía.
4. **Hello Done del servidor:** El servidor envía un mensaje 'ServerHelloDone' para indicar al cliente que ha terminado su fase de la negociación del handshake.
5. **Clave Pre-Master del cliente:** El cliente genera una 'pre-master secret' (clave pre-maestra), la cifra con la clave pública del servidor (obtenida del certificado del servidor) y la envía al servidor.
6. **Hello del cliente finalizado:** El cliente envía un mensaje 'ClientFinished'. Este mensaje está cifrado con una clave de sesión que se deriva de la 'pre-master secret'.
7. **Descriptado de la clave Pre-Master del servidor y Hello finalizado del servidor:** El servidor descifra la 'pre-master secret' con su clave privada y deriva la misma clave de sesión que el cliente. Luego, envía su propio mensaje 'ServerFinished', cifrado con la misma clave de sesión.

A partir de este punto, tanto el cliente como el servidor pueden cifrar toda la comunicación con la clave de sesión. Cabe señalar que en este proceso no se transmite ninguna información sensible sin cifrar. Aunque un atacante pueda interceptar la comunicación, no podrá descifrar la información sin la clave privada del servidor, que nunca se transmite.

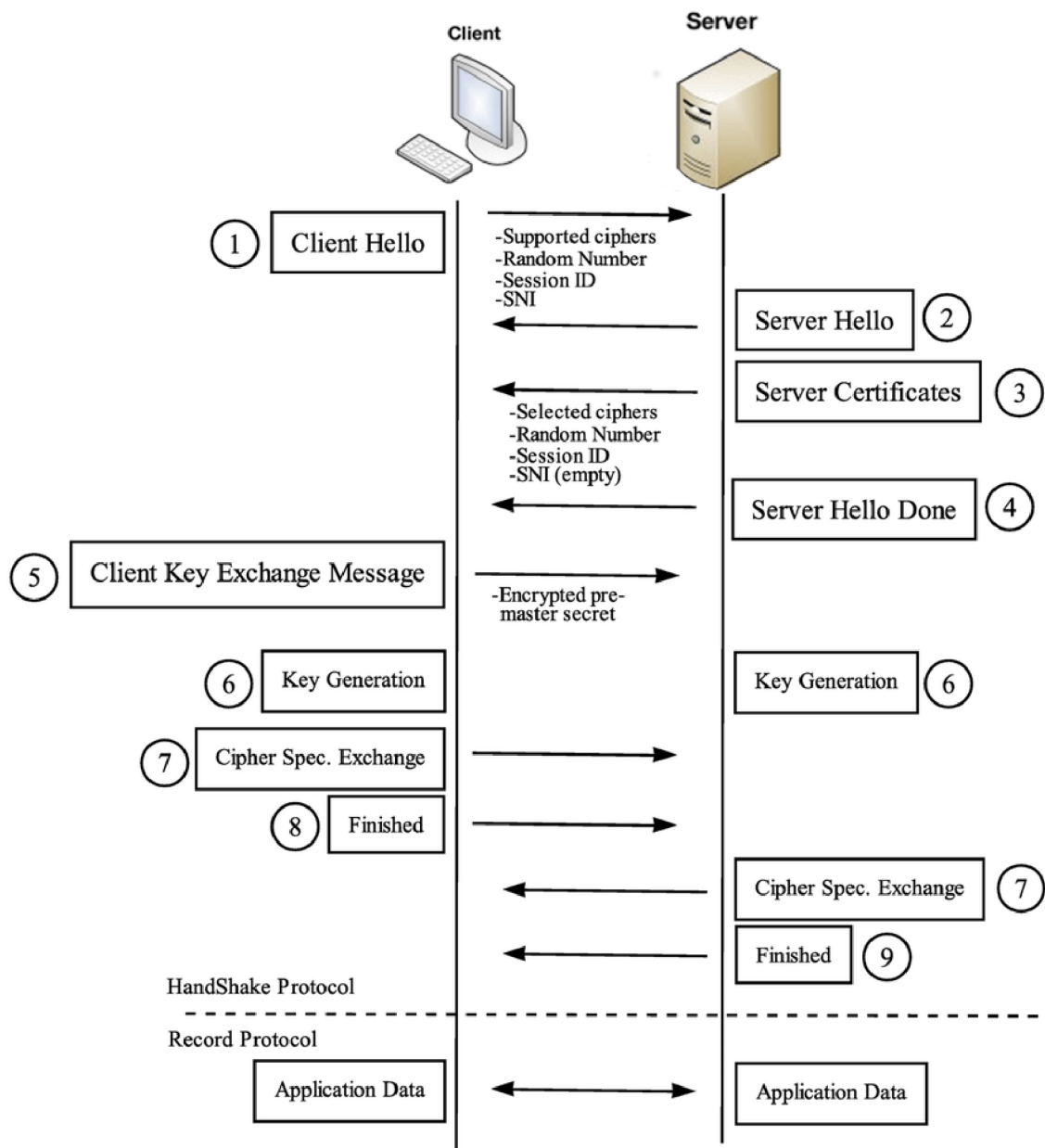


Figura 4.1 - Handshake Protocol

Fuente: https://www.researchgate.net/figure/TLS-handshake-protocol_fig1_298065605(2023)

4.3 Broker (EMQX)

Se comenzará explicando qué es un broker. Un broker es un servidor de mensajería que utiliza el protocolo MQTT (Message Queuing Telemetry Transport), que es un protocolo de mensajería de código abierto diseñado para su uso en situaciones de baja potencia y baja

conectividad de red, especiales para dispositivos como el ESP32. El protocolo cumple la función de coordinar la comunicación entre dispositivos de IoT.

Ahora EMQX es un broker de mensajería de código abierto altamente escalable y confiable que está diseñado para la gestión de mensajes en tiempo real y en grandes cantidades.

El broker es compatible con una amplia variedad de dispositivos y plataformas. Además, de ofrecer una amplia gama de funciones de seguridad, como el cifrado **TLS** (Transport Layer Security) / **SSL** (Secure Sockets Layer), que es un protocolo de seguridad utilizado para establecer una conexión segura entre un servidor web y un navegador web. La **autenticación**, que se realiza a través de la validación de credenciales ingresadas por el usuario. Finalmente la **autorización está basada en certificados**, lo que garantiza que los datos estén seguros durante la transmisión y la convierte en una herramienta muy flexible y eficiente para el desarrollo de este trabajo.

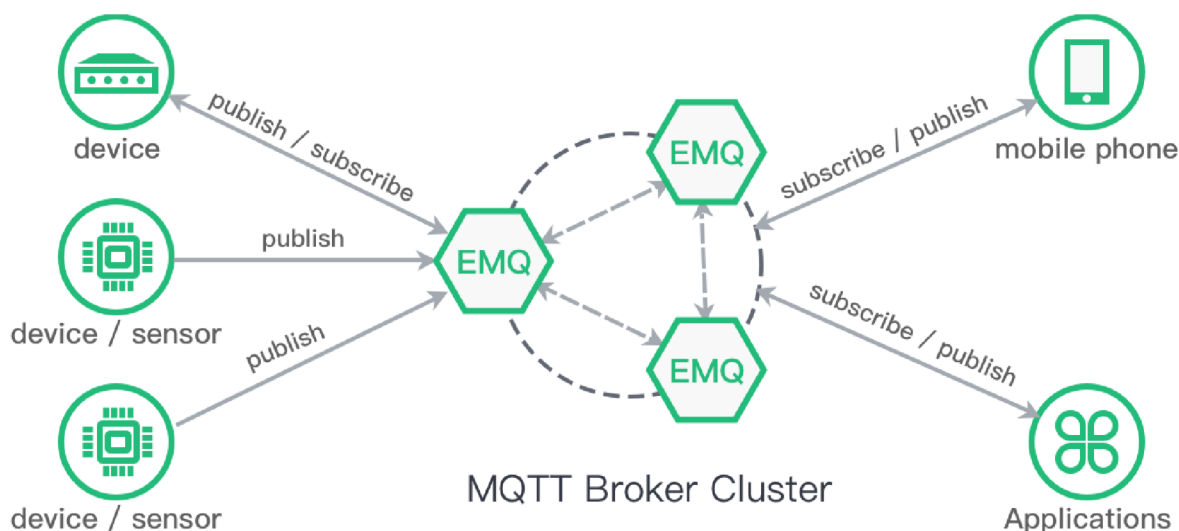


Figura 4.2 - Funcionamiento del broker. En la izquierda los dispositivos que emiten los datos, en el centro el broker encargado de redirigir los mensajes(datos), en la izquierda los dispositivos finales que esperan leer los datos.

Fuente: Recuperado de

<https://www.emqx.io/docs/en/v3.0/>

(2023)

4.3.1 Dashboard (Emqx)

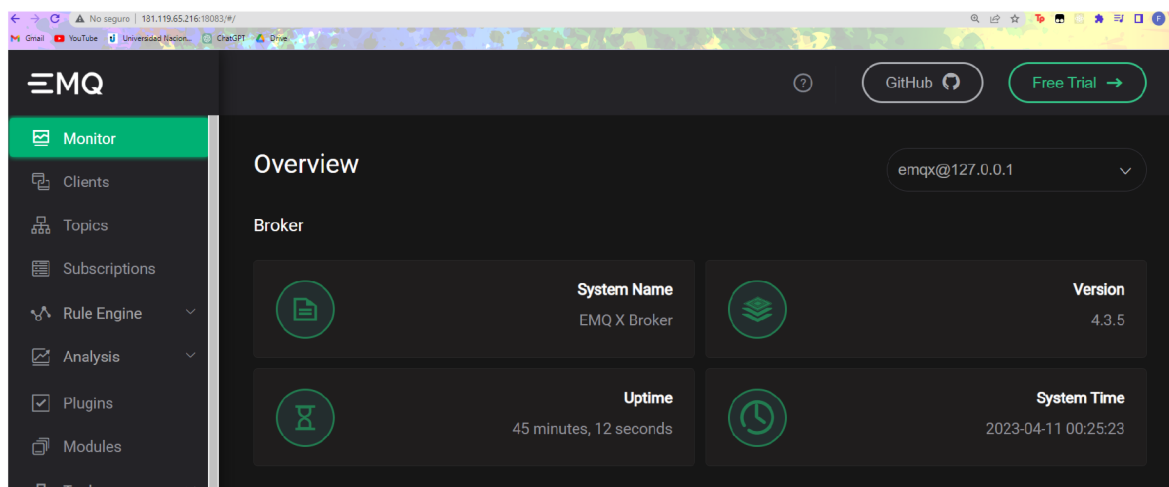


Figura 4.3 - Dashboard de EMQX

Fuente: Propia

El dashboard de EMQX es una interfaz gráfica de usuario que proporciona una vista completa y en tiempo real de su broker MQTT. El dashboard de EMQX es importante porque permite supervisar y gestionar fácilmente el estado de los mensajes que se están enviando a través de los dispositivos finales, incluyendo la cantidad de clientes conectados, la tasa de mensajes enviados/recibidos, la carga de CPU, la memoria y la información sobre los temas (topics) publicados y suscritos, lo que es esencial para garantizar una comunicación IoT eficiente y sin interrupciones. También permite realizar diversas tareas de gestión, como la creación y eliminación de clientes, suscripciones y temas.

La información que proporciona puede ser utilizada para identificar problemas y optimizar la configuración del broker MQTT para mejorar el rendimiento. Por ejemplo, si se observa una carga de CPU alta en el dashboard, se puede optar por agregar más recursos al servidor o ajustar la configuración de sus clientes MQTT para reducir la carga en el broker.

El dashboard de EMQX también es útil para fines de supervisión y análisis de datos. Se pueden configurar alertas para notificar cuando se superan ciertos umbrales, lo que les permite tomar medidas inmediatas en caso de problemas. Además, la información de análisis de datos recopilada puede utilizarse para identificar patrones y tendencias, lo que puede ayudar a optimizar el rendimiento y mejorar la eficiencia de la comunicación MQTT.

4.3.2 Autenticación con MySQL(Emqx)

La autenticación con MySQL en EMQ X permite a los usuarios autenticarse mediante credenciales almacenadas en una base de datos MySQL en lugar de en un archivo de configuración local.

Las razones por las cuales se procedió a utilizar este método de autenticaciones es debido a que el mismo ofrece varias ventajas, entre las que se incluyen:

- **Escalabilidad:** la autenticación con MySQL permite una fácil escalabilidad de EMQ X, ya que los usuarios pueden ser autenticados a través de una base de datos centralizada en lugar de un archivo de configuración local. Esto es especialmente útil para grandes despliegues con múltiples instancias de EMQ X.
- **Seguridad:** la autenticación con MySQL proporciona una forma más segura de autenticar a los usuarios, ya que las credenciales de usuario se almacenan de manera más segura en una base de datos encriptada en lugar de en un archivo de configuración local en texto plano.
- **Facilidad de uso:** la autenticación con MySQL simplifica la administración de usuarios y las credenciales, ya que los usuarios y sus credenciales pueden ser administrados de manera centralizada en la base de datos MySQL.
- **Flexibilidad:** la autenticación con MySQL permite a los usuarios autenticarse con diferentes credenciales de inicio de sesión, como direcciones de correo electrónico o nombres de usuario personalizados, lo que proporciona una mayor flexibilidad para la autenticación de usuarios.

Para configurar la autenticación con MySQL en EMQ X en el servidor, se procedió con los siguientes pasos:

1. **Configurar la conexión de MySQL:** se debe proporcionar la información de conexión de MySQL, como la dirección IP, el puerto, el nombre de usuario y la contraseña.

Para ello, en el archivo `emqx_auth_mysql.conf`, ubicado en `/emqx/etc/plugins/`, se localiza la siguiente línea y se verifica que esté configurada correctamente:

```
auth.mysql.acl_query = select allow, ipaddr, username, clientid, access,  
topic from mqtt_acl where ipaddr = '%a' or username = '%u' or username =  
'$all' or clientid = '%c'
```

2. **Configurar la tabla de usuarios:** se debe crear una tabla en la base de datos MySQL que contenga información de usuarios, como nombre de usuario y contraseña. La estructura de la tabla debe coincidir con el esquema de autenticación de EMQ X.

Creamos esta tabla en la base de datos MySql que fue configurada anteriormente:

```
CREATE TABLE `mqtt_acl` (  
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,  
  `allow` int(1) DEFAULT 1 COMMENT '0: deny, 1: allow',  
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',  
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',  
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',  
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',  
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Descripción de los campos de la tabla:

- **Allow:** *Denegar , Permitir.*
- **IpAddr:** Establecer dirección.
- **IP username:** Nombre de usuario para conectarse al cliente. Si el valor se establece en \$ all, la regla se aplica a todos los usuarios.
- **Clientid:** ID de cliente del cliente conectado.
- **Access:** Operaciones permitidas: *suscribir, publicar, ambas suscribir y publicar.*
- **Topic:** Temas a ser controlados, los cuales pueden usar comodines, y se pueden agregar marcadores de posición al tema para coincidir con la información del cliente. Por ejemplo, el tema será reemplazado con la ID del cliente actual cuando coincida con t/%c.
- **%u:** Nombre de usuario.

- **%c**: ID de cliente.

3. **Configurar el esquema de autenticación en EMQ X**: se debe configurar el archivo de EMQ X para que utilice el esquema de autenticación con MySQL en lugar de la autenticación local. Una vez configurado, EMQ X se conectará a la base de datos MySQL y autenticará a los usuarios utilizando las credenciales almacenadas en la tabla de usuarios.

Para esto se incorporan las reglas **ACL** (Control de acceso basado en listas), a la tabla, mediante consultas a la BD correspondiente. Esto permite controlar quién puede acceder a los recursos y servicios del servidor de mensajería MQTT.

A continuación se detallan ejemplos de algunas reglas personalizadas las cuales fueron configuradas para la base de datos MySQL utilizada para el trabajo.

Bloquear la suscripción a cualquier tópico

Para bloquear la suscripción y publicación de cualquier usuario a cualquier tópico, se añadió el siguiente registro a la tabla:

```
insert into `mqtt_acl` (`access`, `allow`, `clientid`, `id`, `ipaddr`,  
`topic`, `username`) values (3, 0, NULL, 1, NULL, '+/#', '$all');
```

Permitir la suscripción a un tópico

Para permitir que un usuario se suscriba a un determinado tópico, se añadió el siguiente registro:

```
insert into `mqtt_acl` (`access`, `allow`, `clientid`, `id`, `ipaddr`,  
`topic`, `username`) values (1, 1, NULL, 2, NULL,  
'v1/devices/cliente1/#', 'admin');
```

Permitir la publicación a un tópico

Para permitir que un usuario publique en un tópico, se ejecutó el siguiente comando:

```
insert into `mqtt_acl` (`access`, `allow`, `clientid`, `id`, `ipaddr`,  
`topic`, `username`) values (2, 1, NULL, 5, NULL,  
'v1/devices/cliente1/ejemplo', 'admin');
```

4. Añadir certificados SSL para WSS y SSL

Para añadir certificados SSL de Let's Encrypt, primero es necesario copiar estos certificados dentro del directorio `/etc/emqx/certs`. Por ejemplo, luego de haber ingresado como root a través del comando `sudo -i`, se podría ejecutar:

```
cat /etc/letsencrypt/live/iotsur.xyz/chain.pem >  
/etc/emqx/certs/chain.pem  
cat /etc/letsencrypt/live/iotsur.xyz/privkey.pem >  
/etc/emqx/certs/privkey.pem  
cat /etc/letsencrypt/live/iotsur.xyz/fullchain.pem >  
/etc/emqx/certs/fullchain.pem  
cat /etc/letsencrypt/live/iot-ia-group.com/chain.pem >  
/etc/emqx/certs/chain.pem  
cat /etc/letsencrypt/live/iot-ia-group.com/privkey.pem >  
/etc/emqx/certs/privkey.pem  
cat /etc/letsencrypt/live/iot-ia-group.com/fullchain.pem >  
/etc/emqx/certs/fullchain.pem
```

Posteriormente, se hace dueño a `emqx:emqx` de los archivos que se encuentran dentro de `/etc/emqx/certs/`. De otra manera, no funcionará. Entonces, corremos el siguiente comando dentro de la carpeta `certs`:

```
sudo chown emqx:emqx *
```

Finalmente, se edita el archivo de configuración de EMQ X, que se encuentra ubicado dentro de `/etc/emqx/`. Debería quedar configurado de la siguiente manera:

```
listener.wss.external.keyfile = /etc/emqx/certs/privkey.pem  
listener.wss.external.certfile = /etc/emqx/certs/fullchain.pem  
listener.wss.external.cacertfile = /etc/emqx/certs/chain.pem
```

4.3.3 MQTTX (Emqx)

Para el descifrado del algoritmo XOR, explicado en la sección (*Algoritmo de cifrado XOR*, Pág 73) , se utilizó como prueba la aplicación MQTTX, la cual es proporcionada por Emqx.

MQTTX soporta el protocolo MQTT v5.0 y MQTT v3.1.1, admite MQTT sobre WebSockets, es compatible para diversas plataformas como Windows, macOS y Linux.

Finalmente posee una característica de gran interés para este trabajo que es la capacidad de introducir un Script personalizable para simular pruebas de datos en lenguaje java.

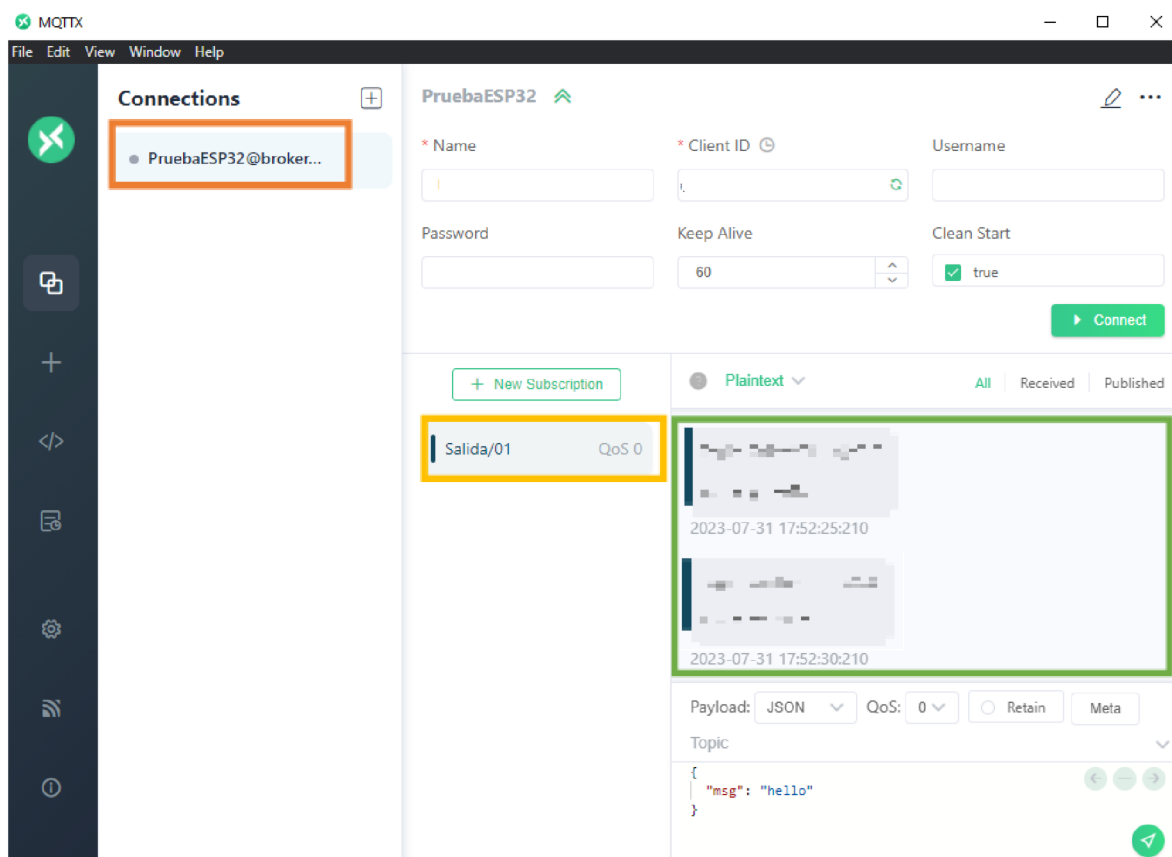


Figura 4.4 - App MQTTX

Fuente: Propia

La App proporciona un dashboard similar al de Emqx, donde encontramos una conexión Mqtt llamada “**PruebaESP32@broker**”, seguido de esto en la sección en que recubre el rectángulo verde se pueden ver los mensajes que recibe el broker y finalmente en la sección marcada en amarillo vemos el nombre del tópic al que estamos suscritos.

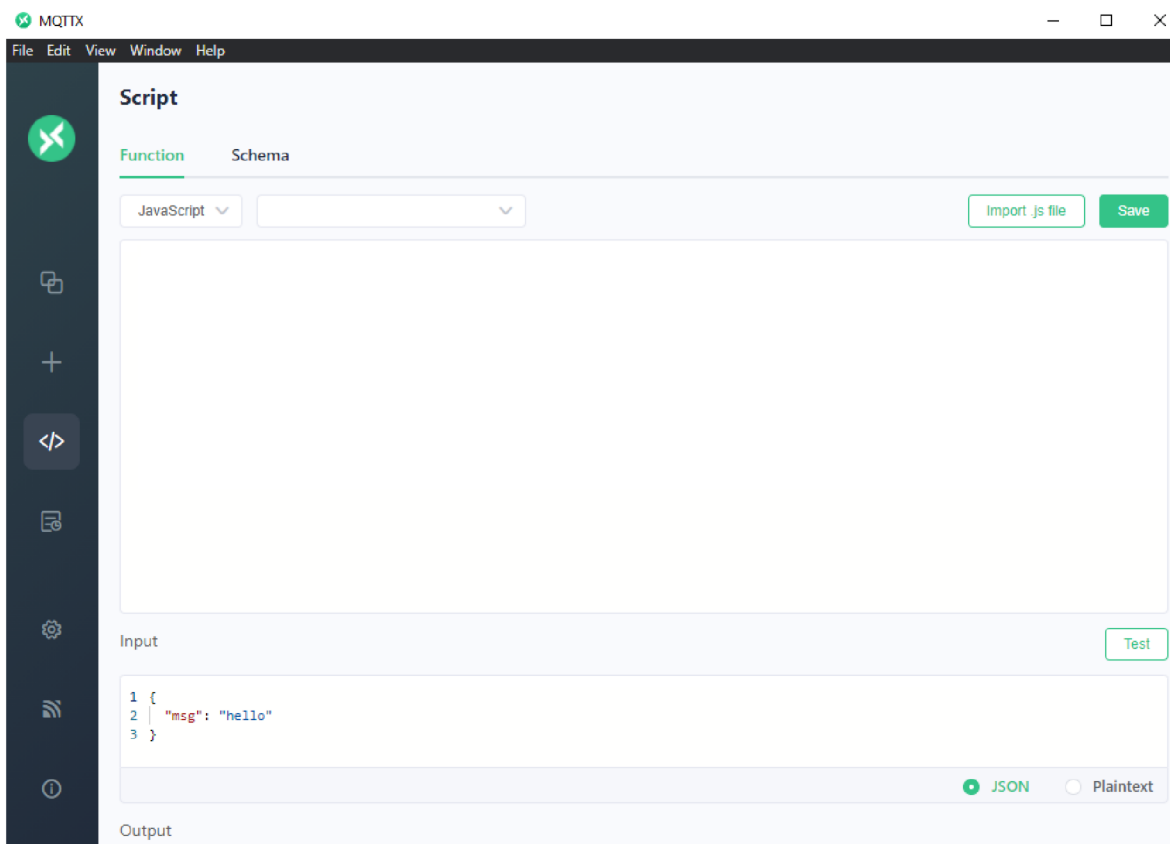


Figura 4.5 - MQTTX Script

Fuente: Propia

En la sección de “**Script**”, es donde la aplicación nos permite cargar el código en java, el cual luego podrá ejecutarse en tiempo real, en el caso de este trabajo para lograr descifrar los mensajes que lleguen encriptados al broker.

4.4 Node Red

Node-RED es una herramienta de programación visual que facilita la creación de aplicaciones y flujos de trabajo utilizando nodos interconectados. Se basa en Node.js, lo que le permite aprovechar la potencia y flexibilidad de JavaScript para la automatización y la integración de sistemas.

El objetivo principal de Node-RED es permitir a los desarrolladores y usuarios crear fácilmente flujos de trabajo y aplicaciones conectando diferentes servicios y dispositivos,

ya que proporciona una interfaz gráfica intuitiva, donde los nodos representan diferentes funciones o servicios, y los mismos se pueden conectar para crear flujos lógicos.

Algunas de sus características clave:

1. **Creación de flujos:** Node-RED permite crear flujos de trabajo visuales conectando nodos entre sí. Los nodos pueden representar servicios web, bases de datos, sensores, actuadores y muchas otras funciones. Al conectar los nodos, se crea un flujo lógico que define el comportamiento de la aplicación. Biblioteca de nodos: Node-RED cuenta con una amplia biblioteca de nodos predefinidos que cubren una variedad de servicios y dispositivos populares.
2. **Integración y automatización:** Con Node-RED, es posible integrar y automatizar sistemas y servicios de diferentes proveedores. Se puede utilizar los nodos disponibles para interactuar con servicios web, enviar y recibir datos, ejecutar comandos en el sistema operativo, acceder a bases de datos, etc. Esto permite una integración fluida entre los sistemas y la automatización de tareas.
3. **Desarrollo rápido de prototipos:** Node-RED es especialmente útil para el desarrollo rápido de prototipos y pruebas de concepto. Dado que no requiere una programación tradicional, puedes diseñar y probar flujos de trabajo rápidamente, lo que agiliza el proceso de desarrollo y permite una iteración más rápida.
4. **Escalabilidad y despliegue:** Aunque Node-RED es excelente para el desarrollo rápido, también puede escalarse para implementaciones más grandes. Puede ejecutarse en servidores locales, en la nube o incluso en dispositivos de Internet de las cosas (IoT). Además, se pueden crear flujos complejos utilizando técnicas de separación en módulos y abstracciones, lo que facilita el mantenimiento y la escalabilidad.

4.4.1 Seguridad en Node Red

En cuanto a la seguridad, Node-RED ofrece una serie de características y medidas para garantizar la protección de los datos y la integridad de las aplicaciones.

Autenticación: Node-RED permite configurar la autenticación basada en usuarios/contraseñas para acceder a la interfaz de administración y al editor de flujos. Esto evita accesos no autorizados y protege contra ataques de fuerza bruta.

Control de acceso basado en roles: Node-RED admite un sistema de control de acceso basado en roles, lo que significa que puedes asignar diferentes permisos a los usuarios según su función o nivel de acceso. Esto garantiza que solo los usuarios autorizados puedan realizar cambios o acceder a ciertas partes del sistema.

Encriptación de datos: Node-RED permite la configuración de conexiones seguras mediante el uso de protocolos de encriptación, como HTTPS (HTTP seguro) o MQTT con TLS/SSL. Esto asegura que la comunicación entre los nodos y los dispositivos conectados se realice de forma segura, evitando la interceptación o manipulación de datos sensibles.

Protección contra ataques: Node-RED tiene una comunidad activa, una amplia biblioteca de extensiones y paquetes que se someten a revisiones en conjunto con actualizaciones constantes. Esto ayuda a detectar y solucionar posibles vulnerabilidades de seguridad de manera oportuna, lo que reduce el riesgo de ataques externos.

Registro y auditoría: Node-RED permite llevar un registro detallado de las actividades realizadas en el sistema, lo que facilita la auditoría y el seguimiento de posibles incidentes de seguridad. Esto ayuda a identificar actividades sospechosas y tomar medidas correctivas de manera proactiva.

4.4.1 Configuración de Node Red

4.4.1.1 Creación de un usuario

La creación de los usuarios es de suma importancia, ya que se crean en Node-RED con el propósito de gestionar y controlar el acceso a la plataforma. Al crear usuarios en Node-RED, se pueden asignar diferentes roles y permisos para limitar las acciones que cada usuario puede realizar.

Para crear los usuario de este trabajo se realizaron los siguientes pasos:

1. **Ubicar el archivo de configuración:** Para obtener donde se encuentra ubicado el archivo de configuración, podemos ejecutar el comando: `node-red --settings`
2. **Creación del usuario:** Para crear el hash de la contraseña que utilizaremos para cualquier usuario, corremos el siguiente comando: `node-red admin hash-pw`
Dentro del archivo de configuración encontramos todos los usuarios que pueden acceder a la aplicación, detallando el nombre de usuario, contraseña y permisos de cada uno. Creamos los usuarios que necesitemos y luego pegamos la contraseña hasheada que creamos con el anterior comando. Por ejemplo:

```
(...)  
adminAuth: {  
  type: "credentials",  
  users: [  
    {  
      username: "admin",  
      password: "*hash*",  
      permissions: "*"   
    },  
    {  
      username: "guest",  
      password: "*hash*",  
      permissions: "read"  
    }  
  ]  
}  
(...)
```

4.4.1.2 Distribucion de Nodos para el procesamiento de datos

Los nodos ya mencionados anteriormente, son bloques fundamentales que representan las diferentes funcionalidades y componentes de una aplicación. Cada uno realiza una tarea específica, se puede utilizar para crear flujos de trabajo automatizados y lógica de programación sin necesidad de escribir código.

A continuación se presenta el esquema de flujo, utilizado en el trabajo realizado, donde se muestra, como cada dato que es enviado al tópic “/V1/#” es procesado por el Nodo “function”, el cual contiene un código en el lenguaje Java, que da formato al mismo, con el fin de almacenarlo en la BD. Luego se clasifica al tópic en “/V1/device1/temperatura”,

para que este mismo reciba los datos, en este caso particular la temperatura, de un dispositivo, y luego continúe con el proceso indicado anteriormente.

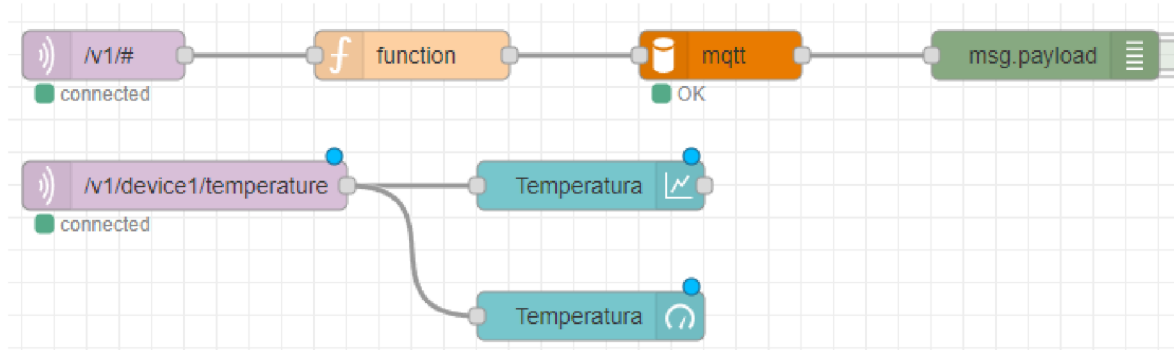


Figura 4.6 - Nodos Node Red

Fuente: Propia

4.4.1.3 Utilización de un proxy inverso

Para esta PPS se utilizó una configuración de Node Red en conjunto con Nginx, ya que la misma proporciona una mejora sustancial de seguridad, debido que al utilizar un proxy inverso como Nginx, se puede agregar una capa adicional de seguridad al entorno de Node-RED. Nginx actúa como un punto de entrada único y puede aplicar medidas de seguridad como filtrado de tráfico, bloqueo de direcciones IP no deseadas y prevención de ataques de denegación de servicio (DDoS). También puede proporcionar certificados SSL/TLS para habilitar conexiones seguras a Node-RED.

Otra de las ventajas más importante es el “Balanceo de carga”, ya que si se espera un alto volumen de tráfico o se desean distribuir las solicitudes entre varios servidores de Node-RED, Nginx permite implementar estrategias de balanceo de carga. Esto garantiza una distribución equitativa del tráfico entre los servidores y mejora el rendimiento y la capacidad de respuesta del sistema.

Finalmente la configuración de dominio personalizado, nos permite modelar un dominio para acceder a Node-RED. Esto facilita la gestión de la aplicación y proporciona una URL amigable para los usuarios.

4.5 Balanceador de Carga y Proxy Inverso (NginX)

Nginx es un servidor web de alto rendimiento y proxy inverso, así como un servidor de correo electrónico y proxy de TCP/UDP. Se ha vuelto muy popular en la comunidad de desarrollo web debido a su capacidad para manejar eficientemente una gran cantidad de conexiones concurrentes, es decir que tiene la capacidad de manejar la conexión de gran cantidad de usuarios a la vez, y además servir contenido estático de manera eficiente.

4.5.1 ¿Por qué Nginx?

En principio se mostrará una tabla comparativa con otros software de servidores existentes hoy en día, donde finalmente se explicará porqué se eligió a Nginx para realizar este trabajo.

Programa	Ventajas	Desventajas
Apache HTTP Server	Ampliamente utilizado y probado en el tiempo.	- Mayor consumo de recursos en comparación con otros.
	Gran compatibilidad con módulos y tecnologías.	
	Configuración flexible y personalizable.	
	Fuerte comunidad de soporte y documentación	
Microsoft IIS	Integración sólida con tecnologías de Microsoft.	Limitado a entornos Windows.
	Fácil administración y uso en entornos empresariales	
	Soporte para ASP.NET y otras tecnologías de Microsoft.	
	Integración con Active Directory y otras	

	herramientas.	
Caddy	Fácil configuración y uso.	No es tan maduro o estable como otros servidores.
	Integración automática de certificados SSL/TLS.	Funcionalidad limitada en comparación con otros.
	Recarga automática y generación de sitios estáticos.	
Nginx	Alto rendimiento y eficiencia en el uso de recursos.	La configuración puede ser compleja para principiantes.
	Manejo eficiente de conexiones concurrentes.	No es tan compatible con algunos módulos y tecnologías.
	Funciones avanzadas de proxy inverso y equilibrio de carga.	
	Soporte para almacenamiento en caché.	
	Configuraciones flexibles de seguridad y filtrado de tráfico	

Basándonos en la tabla comparativa, se eligió a Nginx sobre las demás opciones por las siguientes razones:

1. **Alto rendimiento y eficiencia:** Nginx es conocido por su excelente rendimiento y eficiencia en el uso de recursos. Puede manejar una gran cantidad de conexiones concurrentes y ofrecer una respuesta rápida incluso en situaciones de alta carga. Esto lo hace ideal para aplicaciones, como la implementada en este trabajo, que requieren una alta capacidad de respuesta y un rendimiento óptimo.
2. **Funciones avanzadas de proxy inverso y equilibrio de carga:** Nginx ofrece características avanzadas para actuar como proxy inverso y distribuir la carga entre

varios servidores. Esto permite una distribución equitativa de las solicitudes y evita la sobrecarga de un solo servidor, mejorando la disponibilidad y el rendimiento general del sitio.

3. **Amplia comunidad de soporte:** Nginx cuenta con una comunidad activa y un amplio respaldo en términos de documentación, tutoriales y recursos disponibles en línea. Esto facilitó el encontrar ayuda para dar soluciones a problemas y mantenerse actualizado con las mejores prácticas de configuración y seguridad.
4. **Configuraciones flexibles de seguridad y filtrado de tráfico:** Nginx ofrece opciones de configuración flexibles para aplicar listas negras de direcciones IP, establecer límites de velocidad y aplicar reglas de filtrado para bloquear tráfico malicioso o no deseado. Estas características ayudan a proteger los servidores y las aplicaciones web contra ataques y abusos, mejorando la seguridad global del sistema.

4.5.2 Proxy Inverso

Un proxy inverso es un servidor intermediario que actúa como punto de entrada para los clientes y redirige las solicitudes a los servidores de destino. A diferencia de un proxy normal, que se ubica entre los clientes y los servidores finales, el proxy inverso se coloca en el lado del servidor y oculta la infraestructura de backend, es decir la lógica del negocio, al cliente.

A continuación se presenta el esquema utilizando como guía en este trabajo, donde se implementó un firewall, el cual permite filtrar el tráfico de la red externa antes de llegar al servidor:

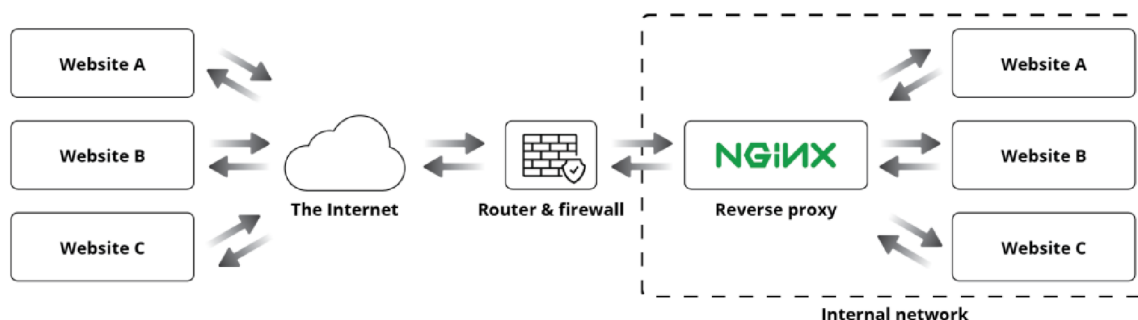


Figura 4.7 - Esquema de funcionamiento de proxy inverso.

Fuente:

<https://www.herlitz.io/2022/03/27/running-optimizely-cms-behind-nginx-reverse-proxy>(2023)

4.5.3 Configuración del Firewall de Nginx para SSH

Como primer paso, se ajustó el firewall para seguir permitiendo la conexión SSH.

Aclaración: si se activa el firewall sin configurar el SSH, se podría perder la conexión.

Listamos las aplicaciones que reconoce el UFW:

- `sudo ufw app list`

Habilitamos a la aplicación OpenSSH para no perder la conexión:

- `sudo ufw allow OpenSSH`

Habilitamos el Firewall:

- `sudo ufw enable`

Verificamos si la app se habilitó sin problemas:

- `sudo ufw status`
- `sudo ufw status verbose` # más detalles

Utilizando el mismo comando que en el primer punto de este apartado, `sudo ufw app list`, listamos las aplicaciones que reconoce UFW. Vemos que obtenemos una lista que cuenta con las siguientes opciones:

- **Nginx Full.** Abre el puerto 80 (HTTP, sin cifrado) y el puerto 443 (HTTPS, cifrado).
- **Nginx HTTP.** Abre sólo el puerto 80 (tráfico sin cifrar).
- **Nginx HTTPS.** Abre sólo el puerto 443 (tráfico cifrado).

Finalmente, habilitamos el perfil deseado a través del comando:

- `sudo ufw allow 'Nginx HTTP'`

4.6 Arduino IDE

Arduino IDE (Integrated Development Environment) es el entorno de desarrollo integrado oficial de Arduino. Es una herramienta de software que proporciona todo lo necesario para escribir, compilar y cargar programas en todo tipo de placas, en este trabajo específicamente se utilizó el ESP32 (Dispositivo programable con conectividad wifi).

La plataforma trae consigo ciertas ventajas que facilitan la compilación y escritura del código, como lo son por ejemplo:

- **Interfaz simple y fácil de usar:** Arduino IDE presenta una interfaz gráfica intuitiva y sencilla que permite a los usuarios escribir y editar su código de manera eficiente. Incluso para aquellos que no tienen experiencia previa en programación, la interfaz de Arduino IDE es accesible y amigable.
- **Lenguaje de programación basado en C/C++ simplificado:** Arduino IDE utiliza un lenguaje de programación basado en C/C++ que ha sido simplificado y adaptado para facilitar su aprendizaje y uso.
- **Biblioteca de código abierto:** Arduino IDE viene con una biblioteca de código abierto que incluye una amplia variedad de funciones y ejemplos predefinidos.
- **Facilidad de compilación y carga de programas:** Arduino IDE permite compilar y cargar fácilmente programas en las placas Arduino. Una vez que se ha escrito el código, se puede verificar su sintaxis y compilarlo con solo presionar un botón. Luego, el programa compilado se carga en la placa Arduino a través de una conexión USB, lo que permite su ejecución.

- **Amplia comunidad y soporte:** Arduino IDE cuenta con una comunidad activa de usuarios en todo el mundo. Esto significa que se puede acceder a una amplia gama de recursos, como tutoriales, documentación, foros de discusión y proyectos compartidos por otros usuarios.

4.6.1 Compilación

El compilador utilizado por Arduino IDE es el GCC (GNU Compiler Collection), que es un conjunto de herramientas de compilación de código abierto ampliamente utilizado en el desarrollo de software. El GCC es altamente configurable y compatible con una amplia variedad de plataformas y arquitecturas de microcontroladores, incluyendo aquellos utilizados en las placas Arduino.

El compilador de Arduino IDE funciona en varias etapas para traducir el código escrito en el lenguaje de programación de Arduino (basado en C/C++) en un lenguaje de bajo nivel que pueda ser comprendido y ejecutado por el microcontrolador de la placa Arduino.

A continuación, se explica el proceso general de compilación:

1. **Análisis léxico y sintáctico:** El compilador analiza el código fuente de Arduino, lo divide en unidades léxicas, como palabras clave, identificadores, operadores y constantes. Luego, realiza un análisis sintáctico para verificar si el código cumple con las reglas gramaticales del lenguaje.
2. **Generación de código intermedio:** El compilador genera un código intermedio, también conocido como código objeto, que es una representación interna del programa en un formato más bajo nivel que el código fuente original. Este código intermedio se almacena en archivos objeto (.o).
3. **Optimización de código:** El compilador aplica diversas técnicas de optimización al código intermedio para mejorar su eficiencia en términos de rendimiento y uso de memoria. Estas optimizaciones pueden incluir la eliminación de código redundante, la simplificación de expresiones matemáticas y la reorganización del código para mejorar el acceso a la memoria.

4. **Enlace (Linking):** Durante el proceso de enlace, el compilador vincula el código objeto generado junto con las bibliotecas y dependencias necesarias. El enlace resuelve las referencias a funciones y variables externas, asegurando que todas las partes del programa estén conectadas correctamente. Esto resulta en la creación de un archivo binario final en formato Intel Hex (.hex) que contiene el programa completo listo para ser cargado en la placa Arduino.
5. **Generación del archivo final:** El compilador genera el archivo binario (.hex) que contiene el programa completo y puede ser cargado en la memoria del microcontrolador de la placa Arduino. Este archivo binario es el resultado final del proceso de compilación y enlace.

4.6.2 Bibliotecas Arduino

La biblioteca de código abierto en Arduino IDE es una colección de funciones, rutinas y ejemplos de código predefinidos para facilitar la programación de proyectos. Estas bibliotecas están diseñadas para interactuar con diferentes componentes electrónicos y realizar tareas comunes de manera más sencilla.

Cuando se trabaja en un proyecto, a menudo se necesita realizar operaciones como leer datos de sensores, controlar actuadores o comunicarse con otros dispositivos. En lugar de tener que escribir todo el código desde cero para realizar estas operaciones, la biblioteca de código abierto proporciona funciones y ejemplos ya desarrollados que se pueden utilizar directamente o como base para la programación personalizada del proyecto.

La biblioteca de Arduino IDE viene con una variedad de funciones y ejemplos predefinidos que cubren una amplia gama de aplicaciones comunes. Por ejemplo, existen bibliotecas para controlar pantallas LCD, sensores de temperatura, módulos de comunicación inalámbrica, motores, luces LED, entre otros componentes electrónicos populares. Estas bibliotecas están escritas en código C/C++ y encapsulan las operaciones complejas necesarias para interactuar con cada componente específico.

Además de la biblioteca incluida en el IDE, la comunidad de Arduino ha desarrollado una gran cantidad de librerías adicionales que se pueden agregar al entorno. Estas librerías son

creadas y compartidas por usuarios y desarrolladores de Arduino de todo el mundo, las mismas abarcan desde funciones específicas para dispositivos particulares, hasta librerías más genéricas que amplían las capacidades del IDE.

La posibilidad de agregar librerías adicionales al IDE es una de las ventajas clave de la biblioteca de código abierto en Arduino. Esto permite aprovechar las contribuciones y experiencias de la comunidad para agregar funcionalidades específicas a sus proyectos sin tener que desarrollar todo el código desde cero. Además, la disponibilidad de librerías adicionales ayuda a acelerar el proceso de desarrollo y permite centrarse en la lógica específica del proyecto en lugar de tener que implementar operaciones básicas.

4.7 ESP32 (Dispositivo programable con conectividad wifi)

El ESP32 es un microcontrolador de bajo consumo de energía que combina Wi-Fi y Bluetooth en un solo chip.

En la actualidad se utiliza para desarrollar proyectos que requieren conectividad inalámbrica, como aplicaciones de Internet de las cosas (IoT), automatización del hogar, monitoreo y control remoto, además de muchos otros proyectos de electrónica y programación.

Este dispositivo presenta varias ventajas las cuales lo hacen ideal para la utilización en este trabajo, ya que incorpora capacidades Wi-Fi y Bluetooth integradas, lo que permite la conectividad inalámbrica fácil y versátil. Puede conectarse a redes Wi-Fi existentes y establecer comunicación inalámbrica con otros dispositivos a través de Bluetooth.

También posee un procesador dual-core Xtensa LX6, que funciona a una velocidad de hasta 240 MHz, esta potencia de procesamiento permite ejecutar aplicaciones complejas y manejar múltiples tareas simultáneamente.

El ESP32 tiene un muy bajo consumo de energía, a pesar de su potencia de procesamiento, es altamente eficiente en términos de consumo de energía, pudiendo operar en diferentes modos de bajo consumo, lo que lo hace adecuado para proyectos como el de este trabajo,

que requieren una duración prolongada de la batería ya que se tomarán mediciones constantes durante un tiempo prolongado.

La amplia gama de pines Entrada/Salida (E/S), permiten la conexión y control de diversos dispositivos y sensores (Temperatura, humedad, etc). Esto brinda flexibilidad en el diseño y grandes posibilidades para la expansión del proyecto.

Una gran capacidad de memoria es muy importante, ya que el almacenamiento donde va a estar alojado el programa tiene que ser suficiente para poder ejecutar el mismo de la manera más óptima. El ESP32 cuenta con 4 MB de memoria flash integrada, lo que proporciona un amplio espacio de almacenamiento para el programa y los datos. Además, dispone de 520 KB de SRAM para el almacenamiento en tiempo de ejecución.

Finalmente, a las demás características ya mencionadas se le suma el bajo costo que posee el dispositivo, lo cual lo hace una opción muy llamativa y asequible para prototipos y proyectos de bajo costo, frente a otros microcontroladores con capacidades similares.

4.7.1 ¿Porque el ESP32?

A continuación, se presenta una tabla comparativa detallando las ventajas y desventajas de los dispositivos vistos para la realización de este trabajo: ESP32, ESP8266, Arduino UNO y Raspberry Pi. Al final, se explica por qué se seleccionó ESP32 en ciertos escenarios.

Dispositivo	Ventajas	Desventajas
ESP32	Procesador de doble núcleo y alto rendimiento.	Precio ligeramente más alto que el ESP8266 y Arduino UNO.
	Conectividad Wi-Fi y Bluetooth integrada.	Mayor consumo de energía en comparación con el ESP8266.
	Amplia memoria y capacidad para ejecutar aplicaciones.	Menos soporte y documentación en comparación con Arduino.
	Gran cantidad de pines	No tiene tantas opciones de

	GPIO y periféricos.	shields y accesorios como Arduino.
	Ideal para proyectos de IoT y conectividad inalámbrica.	
ESP8266	Bajo costo y asequible para proyectos económicos	Procesador menos potente que el ESP32 y Raspberry Pi
	Conectividad Wi-Fi integrada y fácil de configurar	Menos memoria y capacidad de almacenamiento que el ESP32
	Amplia comunidad y abundante documentación disponible.	No tiene soporte Bluetooth integrado.
	Adecuado para proyectos simples y de baja complejidad	
Arduino UNO	Fácil de usar, ideal para principiantes en electrónica..	Limitado en términos de capacidad de procesamiento.
	Amplia gama de shields y accesorios disponibles.	No tiene conectividad inalámbrica incorporada.
	Menor consumo de energía en comparación con ESP8266.	Limitado en cantidad de pines GPIO y periféricos.
	Enfoque en control y monitoreo de dispositivos.	
Raspberry Pi	Potente capacidad de procesamiento, similar a una PC.	Mayor consumo de energía en comparación con otros.
	Conectividad Wi-Fi y Bluetooth integrada (en modelos recientes).	Precio más alto que los microcontroladores.
	Gran cantidad de memoria y capacidad de	No es ideal para aplicaciones de bajo

	almacenamiento.	consumo.
	Sistema operativo completo, lo que permite ejecutar aplicaciones complejas.	

El ESP32 es la mejor opción para el proyecto realizado en este trabajo, ya que es el dispositivo más equilibrado en los siguientes aspectos, los cuales son fundamentales a la hora de la implementación:

- **Conectividad versátil:** El ESP32 ofrece conectividad Wi-Fi y Bluetooth integrada, lo que permite una fácil comunicación inalámbrica con otros dispositivos y redes. Esto lo hace ideal para este proyecto de IoT, junto con la aplicación la cual requiere de una conexión inalámbrica.
- **Potencia de Procesamiento:** El ESP32 tiene un procesador de doble núcleo, un rendimiento significativamente mejor que el ESP8266 y Arduino UNO.
- **Amplia Memoria:** Con una mayor capacidad de memoria en comparación con el ESP8266 y Arduino UNO, el ESP32 permite almacenar una gran cantidad de datos y ejecutar aplicaciones más complejas.
- **Periféricos y Pines GPIO:** El ESP32 ofrece una gran cantidad de pines GPIO y una variedad de periféricos, lo que facilita la conexión de sensores, actuadores y otros dispositivos externos.
- **Comunidad y Soporte:** Aunque es menos popular que Arduino, el ESP32 cuenta con una comunidad en crecimiento y una abundante documentación disponible, lo que facilita el aprendizaje y la resolución de problemas.
- **Relación Precio-Rendimiento:** Aunque el ESP32 es ligeramente más costoso que el ESP8266 y Arduino UNO, su relación precio-rendimiento es excelente considerando sus capacidades y características.

4.7.2 ESP32 Seguridad

El ESP32 ofrece varias opciones de seguridad para garantizar la protección de los datos, y la comunicación. Estas son algunas de las características de seguridad del ESP32:

- Soporte para Wi-Fi protegido (WPA/WPA2) y cifrado de datos mediante AES y criptografía de clave pública.
- **Bluetooth Secure Connections:** el ESP32 admite Bluetooth de baja energía (BLE) con soporte para conexiones seguras mediante la encriptación AES-CCM.
- Almacenamiento seguro de claves: el ESP32 tiene una memoria segura para almacenar claves y datos sensibles de forma cifrada.
- **Firma digital y verificación:** el ESP32 puede firmar digitalmente los datos y verificar la autenticidad de las firmas.
- **Generación de números aleatorios seguros:** el ESP32 cuenta con un generador de números aleatorios criptográficamente seguro para garantizar la seguridad en aplicaciones como el cifrado.
- **Soporte para Secure Sockets Layer (SSL)/Transport Layer Security (TLS):** el ESP32 puede utilizar SSL/TLS para establecer conexiones seguras en aplicaciones de red.
- **Protección contra ataques de desbordamiento de búfer y otras vulnerabilidades comunes:** el ESP32 implementa mecanismos de protección para prevenir ataques a través de desbordamientos de búfer y otros tipos de vulnerabilidades de software.
- **OTA (Over-The-Air) seguro:** el ESP32 permite la actualización de firmware de forma segura a través de la red Wi-Fi o Bluetooth utilizando protocolos seguros y verificación de integridad.
- **Soporte para criptografía de clave pública:** el ESP32 puede generar y utilizar claves públicas y privadas para cifrar y firmar digitalmente datos.

Estas características de seguridad permiten proteger la comunicación, autenticar dispositivos, cifrar datos y garantizar la integridad del firmware en el ESP32, lo que lo hace adecuado para llevar la aplicación a niveles de seguridad más altos.

4.8 Grafana (Dashboard)

Grafana es una plataforma de código abierto ampliamente utilizada para la visualización y análisis de datos en tiempo real. Su objetivo principal es proporcionar una interfaz intuitiva

y amigable para crear paneles y gráficos interactivos que permitan monitorear, analizar y entender datos provenientes de diversas fuentes.

Entre sus usos más comunes se encuentran:

1. **El Monitoreo de Infraestructura:** Grafana es ampliamente utilizado para monitorear infraestructuras, como servidores, redes, bases de datos y dispositivos de IoT. Puede conectarse a herramientas de monitoreo como Prometheus, InfluxDB, Graphite, y mostrar gráficos en tiempo real de métricas, alertas y datos de rendimiento.
2. **Monitorización de Aplicaciones:** Grafana es útil para visualizar y analizar métricas de aplicaciones, como el rendimiento de una aplicación web o el uso de recursos de una aplicación en la nube. Se puede integrar con soluciones como Jaeger o Elasticsearch para rastrear el rendimiento de las aplicaciones y encontrar posibles problemas.
3. **Análisis de Datos Empresariales:** Grafana también se usa para visualizar datos empresariales, como el rendimiento de ventas, métricas de marketing, análisis de datos financieros y otros indicadores clave de rendimiento (KPIs). Esto permite a las empresas tomar decisiones basadas en datos de manera más efectiva.
4. **Visualización de Datos IoT:** En el ámbito del Internet de las cosas (IoT), Grafana se utiliza para representar gráficamente datos recopilados de sensores y dispositivos conectados, lo que facilita la supervisión y análisis del comportamiento de estos dispositivos.
5. **Monitorización de Energía y Servicios Públicos:** Grafana se utiliza en proyectos de eficiencia energética y monitorización de servicios públicos para visualizar y analizar datos relacionados con el consumo de energía y otros recursos.
6. **Creación de Paneles de Control Personalizados:** Grafana permite a los usuarios crear tableros personalizados que se adapten a sus necesidades específicas, lo que facilita el seguimiento y análisis de datos en tiempo real.

4.8.1 ¿Porque Grafana?

A continuación se presenta una comparativa con otros dashboards actuales (Kiba, Power Bi y Tableau), los cuales poseen características similares a grafana:

Grafana vs Kibana:

- *Soporte de fuente de datos:* Grafana tiene un soporte más amplio para bases de datos de series de tiempo como Graphite, InfluxDB, OpenTSDB, y más.
- *Versatilidad:* Grafana puede manejar métricas, logs y rastreos, mientras que Kibana es fuerte en el análisis de registros.
- *Comunidad y plugins:* Grafana tiene una comunidad más grande con más plugins disponibles para extender su funcionalidad.

Grafana vs Power BI:

- *Modelado de datos:* Power BI tiene capacidades más robustas para el modelado de datos y la creación de relaciones entre diferentes conjuntos de datos.
- *Precio:* Grafana es de código abierto y gratuito, mientras que Power BI tiene un alto costo asociado.
- *Despliegue:* Grafana puede desplegarse en cualquier lugar, mientras que Power BI se utiliza principalmente en la nube de Microsoft.

Grafana vs Tableau:

- *Interfaz de usuario:* Tableau tiene una interfaz más intuitiva y fácil de usar para usuarios no técnicos.
- *Precio:* Al igual que Power BI, Tableau tiene un costo asociado mientras que Grafana es de código abierto y gratuito.
- *Soporte de base de datos:* Grafana soporta una variedad de bases de datos de series de tiempo, mientras que Tableau es mejor para trabajar con bases de datos relacionales.

En conclusión, Grafana destaca por su versatilidad, su amplio soporte de bases de datos, su gran comunidad y su accesibilidad de código abierto. Aunque otras herramientas pueden tener puntos fuertes en áreas específicas, Grafana ofrece una solución de visualización de datos potente y altamente personalizable las cuales se ajustan perfecto al proyecto realizado.

4.8.2 Funcionamiento de Grafana

El primer paso para utilizar Grafana es conectarlo a una o varias fuentes de datos. Puede conectarse a una amplia variedad de orígenes, como bases de datos SQL (por ejemplo, MySQL, PostgreSQL), bases de datos NoSQL (por ejemplo, InfluxDB, Elasticsearch), servicios en la nube (por ejemplo, AWS CloudWatch, Google Analytics), entre otros.

Una vez que Grafana está conectado a las fuentes de datos, se puede comenzar a crear paneles. Los paneles son las visualizaciones principales en Grafana y pueden contener gráficos, tablas, medidores, mapas, entre otros tipos de visualizaciones.

Para generar las visualizaciones en un panel, es necesario crear consultas (queries) que recuperen los datos relevantes de la fuente de datos. Grafana ofrece una interfaz gráfica para construir consultas sin necesidad de escribir código. Estas consultas pueden incluir filtros, agrupaciones, funciones de agregación, etc.

Una vez que los datos se obtienen, se puede personalizar la apariencia de las visualizaciones, eligiendo colores, estilos, escalas, etc. Esto ayuda a que la información se muestre de manera clara y comprensible.

Grafana permite organizar paneles relacionados en tableros. Los tableros son una colección de paneles diseñados para proporcionar una vista holística de los datos relacionados. Esto es útil para supervisar múltiples aspectos de un sistema o proceso en una sola interfaz.

Grafana permite interactuar con los datos visualizados. Se utilizan filtros para enfocar en subconjuntos específicos de información o ajustar las escalas para obtener una vista más detallada o general de los datos.

Una de las características poderosas de Grafana es su capacidad para configurar alertas basadas en umbrales predefinidos o condiciones específicas. Cuando los datos superan o caen por debajo de estos umbrales, Grafana puede enviar notificaciones a través de canales como correo electrónico, Slack, etc.

Grafana facilita la compartición de tableros con otros usuarios o equipos. Pudiendo así compartir un enlace al tablero o embeberlo en otras aplicaciones o sitios web.

4.8.3 Grafana y MySQL

1. Creación del usuario

Para conectar MySQL y Grafana, primero necesitamos crear un usuario. Entonces, debemos entrar a MySQL y generar un nuevo usuario, con los siguientes comandos:

- `mysql> CREATE USER 'grafana'@'localhost' IDENTIFIED WITH mysql_native_password BY '*****';`

2. Creación de la base de datos y permisos

Luego, debemos crear una nueva database y otorgarle todos los permisos al usuario que creamos.

- `mysql> CREATE DATABASE grafana;`
- `mysql> GRANT ALL PRIVILEGES ON grafana.* TO 'grafana'@'localhost';`
- `mysql> FLUSH PRIVILEGES;`

3. Adición al data source

Por último, nos dirigimos al dashboard de Grafana y en el apartado de Configuration > Data sources añadimos una nueva base de datos MySQL.

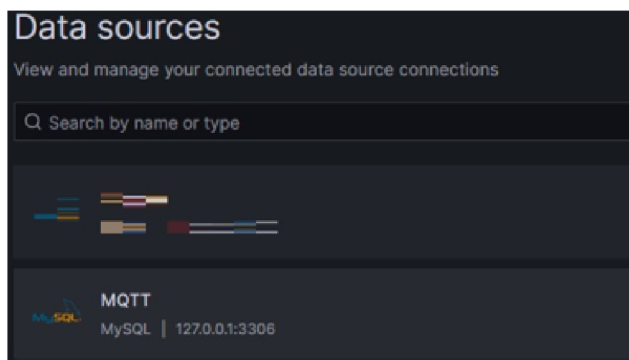


Figura 4.8 - Data source Grafana

Fuente: Propia

4.9 Librería mbedtls

mbed TLS es una biblioteca de código abierto que implementa el protocolo Transport Layer Security (TLS) y el protocolo Secure Sockets Layer (SSL) explicados anteriormente, así como un conjunto completo de criptografía relacionada. Fue desarrollada por ARM y es especialmente adecuada para dispositivos integrados debido a su diseño liviano y modular.

La biblioteca mbed TLS proporciona los algoritmos criptográficos fundamentales para la seguridad en la comunicación de datos, incluyendo el cifrado simétrico y asimétrico, funciones hash, generación de números aleatorios, y más.

Además de su utilidad en el cifrado y la protección de las comunicaciones de red, mbed TLS también es útil para autenticar a las partes en una comunicación, asegurando que los datos no hayan sido manipulados en tránsito (integridad de datos), y para garantizar que los datos sólo puedan ser accedidos por los destinatarios previstos (privacidad de los datos).

A pesar de ser una biblioteca liviana, mbed TLS es capaz de soportar una amplia gama de estándares criptográficos y protocolos de seguridad, incluyendo SSL v3, TLS v1, TLS v1.1, y TLS v1.2. También soporta una variedad de algoritmos de cifrado, incluyendo AES, DES, 3DES, RC4, Camellia, y más.

La biblioteca mbed TLS es altamente configurable y se puede compilar para incluir solo las características específicas que necesita una aplicación particular. La misma tiene

implementaciones dedicadas en muchos de los IDEs y lenguajes de programación, como por ejemplo, Arduino IDE y Python, lo que la hace ideal para este trabajo.

4.10 Algoritmo AES

AES, o Advanced Encryption Standard, es un algoritmo de cifrado simétrico. Esto significa que utiliza la misma clave para el cifrado y el descifrado. AES fue desarrollado por dos criptógrafos belgas, Vincent Rijmen y Joan Daemen, y fue elegido como estándar por el Instituto Nacional de Estándares y Tecnología (NIST) de EE.UU. en 2001.

AES es un algoritmo de cifrado por bloques, lo que significa que divide los datos de entrada en bloques de un tamaño fijo (128 bits en el caso de AES), y luego realiza una serie de operaciones en cada bloque.

4.10.1 AES Encriptado

La información a cifrar, en nuestro caso Mensaje secreto, se agrupa en bloques de 16 bytes (128 bits) en una matriz de 4 x 4:

M	a	s	e
e	j	e	t
n	e	c	o
s		r	.

cada caracter
se traduce
a su código hexadecimal
→

4d	61	73	65
65	6a	65	74
6e	65	63	6f
73	20	72	2e

Figura 4.9 - Encriptado AES (Mensaje a hexadecimal).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

1. Primer paso

El primer paso es hacer un XOR ($\oplus$) entre cada célula del state y de la clave (key). Las claves pueden ser de 128, 192 o 256 bits. Se utilizar una clave de 128 bits:

State:	$\oplus$	Key ₀ :	=	State luego de XOR con Key ₀ :
4d 61 73 65		2b 28 ab 9		66 49 d8 6c
65 6a 65 74		7e ae f7 cf		1b c4 92 bb
6e 65 63 6f		15 d2 15 4f		7b b7 76 20
73 20 72 2e		16 a6 88 3c		65 86 fa 12

Figura 4.10 -Encriptado AES (Operación XOR).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

2. Próximos pasos

Ahora se realizan 4 operaciones:

- SubBytes
- ShiftRows
- MixColumns
- XOR

Las cuales se explican a continuación.

2a. SubBytes

State del paso anterior:

66	49	d8	6c
1b	c4	92	bb
7b	b7	76	20
65	86	fa	12

Resultado de SubBytes:

33	3b	61	50
af	1c	4f	ea
21	a9	38	b7
4d	44	2d	c9

Figura 4.11 -Encriptado AES (Operación SubBytes)

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Algoritmo SubBytes

Esta operación sustituye cada uno de los bytes en el state por valores de una tabla llamada Rijndael S-BOX (se genera por un proceso matemático, [detalles...](#)). La Figura 4.12 muestra cómo se busca la sustitución para el primer y segundo byte.

Primer byte 66:

1. El primer dígito de 66 es 6, se utiliza para buscar en la línea 6.
2. El segundo es 6, busca en la columna 6.
3. La célula en la que coincide es el valor buscado (33).

Segundo byte 1b:

1. El primer dígito de 1b es 1, se utiliza para buscar en la línea 1.
2. El segundo es b, busca en la columna b.
3. La célula en la que coincide es el valor que buscamos (af).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	48	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 4.12 - Encriptado AES (Tabla S-BOX).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

2b. Shift Rows

State del paso anterior:

33	3b	61	50
af	1c	4f	ea
21	a9	38	b7
4d	44	2d	c9

Resultado de ShiftRows:

33	3b	61	50
1c	4f	ea	af
38	b7	21	a9
c9	4d	44	2d

Figura 4.13 -:Encriptado AES (Operación ShiftRows).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Algoritmo ShiftRows

- En este paso la primera línea queda igual.
- En la segunda línea, los bytes 2, 3 y 4 rotan 1 byte a la izquierda. El primer byte pasa a ser el último.
- En la tercera línea, los bytes 3 y 4 rotan 2 bytes a la izquierda. Los primeros 2 bytes pasan a ser los últimos.
- En la cuarta línea, el byte 4 rota 3 bytes a la izquierda. Los primeros 3 bytes pasan a ser los últimos.

Movimientos en cada línea:

33	3b	61	50
af	1c	4f	ea
21	a9	38	b7
4d	44	2d	c9

33	3b	61	50
1c	4f	ea	af
38	b7	21	a9
c9	4d	44	2d

Figura 4.14 - Encriptado AES (Operación ShiftRows movimiento de línea).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

2c. MixColumns

State del paso anterior:

33	3b	61	50
1c	4f	ea	af
38	b7	21	a9
c9	4d	44	2d

Resultado de MixColumns:

b3	5d	82	ce
8a	2a	89	d8
1f	d6	5	c1
f8	2f	e0	ac

Figura 4.15 - Encriptado AES (Operación MixColumns).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Algoritmo MixColumns

Este es el paso más complejo. Los bytes de cada columna del state, son módulos multiplicados por una matriz dada usando un método conocido por *Rijndael Galois Field*.

Esta imagen ayuda a entender el proceso (Figura 4.26):

State				M			
33	3B	61	50	02	03	01	01
1C	4F	EA	AF	01	02	03	01
38	B7	21	A9	01	01	02	03
C9	4D	44	2D	03	01	01	02

Figura 4.16 - Encriptado AES (Tabla State y M, Método Rijndael Galois Field).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

- La tabla State es el state antes de la operación.
- La tabla M es la matriz dada.

Se aplica el MixColumns a la primera célula del state (33). Para eso se debe multiplicar la columna azul del state por la línea azul de la tabla M y hacer XOR entre los resultados de la siguiente forma:

$$so = 33 * 2 \oplus 1c * 3 \oplus 38 * 1 \oplus c9 * 1$$

Las multiplicaciones son algo especial. Las tablas utilizadas contienen los resultados calculados previamente.

Multiplicación 33 * 2:

Esta es la tabla para multiplicar por 2. El resultado que se busca debe ser de forma igual al buscado en la tabla S-BOX:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	02	04	06	08	0a	0c	0e	10	12	14	16	18	1a	1c	1e
1	20	22	24	26	28	2a	2c	2e	30	32	34	36	38	3a	3c	3e
2	40	42	44	46	48	4a	4c	4e	50	52	54	56	58	5a	5c	5e
3	60	62	64	66	68	6a	6c	6e	70	72	74	76	78	7a	7c	7e
4	80	82	84	86	88	8a	8c	8e	90	92	94	96	98	9a	9c	9e
5	a0	a2	a4	a6	a8	aa	ac	ae	b0	b2	b4	b6	b8	ba	bc	be
6	c0	c2	c4	c6	c8	ca	cc	ce	d0	d2	d4	d6	d8	da	dc	de
7	e0	e2	e4	e6	e8	ea	ec	ee	f0	f2	f4	f6	f8	fa	fc	fe
8	1b	19	1f	1d	13	11	17	15	0b	09	0f	0d	03	01	07	05
9	3b	39	3f	3d	33	31	37	35	2b	29	2f	2d	23	21	27	25
A	5b	59	5f	5d	53	51	57	55	4b	49	4f	4d	43	41	47	45
B	7b	79	7f	7d	73	71	77	75	6b	69	6f	6d	63	61	67	65
C	9b	99	9f	9d	93	91	97	95	8b	89	8f	8d	83	81	87	85
D	bb	b9	bf	bd	b3	b1	b7	b5	ab	a9	af	ad	a3	a1	a7	a5
E	db	d9	df	dd	d3	d1	d7	d5	cb	c9	cf	cd	c3	c1	c7	c5
F	fb	f9	ff	fd	f3	f1	f7	f5	eb	e9	ef	ed	e3	e1	e7	e5

Figura 4.17 - Encriptado AES (Tabla S-BOX).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

$$33 * 2 = 66$$

Multiplicación 1c * 3:

Esta es la tabla para multiplicar por 3:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	03	06	05	0c	0f	0a	09	18	1b	1e	1d	14	16	12	11
1	30	33	36	35	3c	3f	3a	39	28	2b	2e	2d	24	27	22	21
2	60	63	66	65	6c	6f	6a	69	78	7b	7e	7d	74	77	72	71
3	50	53	56	55	5c	5f	5a	59	48	4b	4e	4d	44	47	42	41
4	c0	c3	c6	c5	cc	cf	ca	c9	d8	db	de	dd	d4	d7	d2	d1
5	f0	f3	f6	f5	fc	ff	fa	f9	e8	eb	ee	ed	e4	e7	e2	e1
6	a0	a3	a6	a5	ac	af	aa	a9	b8	bb	be	bd	b4	b7	b2	b1
7	90	93	96	95	9c	9f	9a	99	88	8b	8e	8d	84	87	82	81
8	9b	98	9d	9e	97	94	91	92	83	80	85	86	8f	8c	89	8a
9	ab	a8	ad	ae	a7	a4	a1	a2	b3	b0	b5	b6	bf	bc	b9	ba
A	fb	f8	fd	fe	f7	f4	f1	f2	e3	e0	e5	e6	ef	ec	e9	ea
B	cb	c8	cd	ce	c7	c4	c1	c2	d3	d0	d5	d6	df	dc	d9	da
C	5b	58	5d	5e	57	54	51	52	43	40	45	46	4f	4c	49	4a
D	6b	68	6d	6e	67	64	61	62	73	70	75	76	7f	7c	79	7a
E	3b	38	3d	3e	37	34	31	32	23	20	25	26	2f	2c	29	2a
F	0b	08	0d	0e	07	04	01	02	13	10	15	16	1f	1c	19	1a

Figura 4.18 - Encriptado AES (Tabla S-B0X).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

$$1c * 3 = 24$$

La multiplicación por 1 no altera el valor, por lo que podemos ir al último paso:

$$66 \oplus 24 \oplus 38 \oplus c9 = b3$$

b3 es el nuevo valor de la célula 0.

2d. XOR con Key1

Resultado de MixColumns	⊕	Key ₁	=	XOR:
b3 5d 82 ce		a0 88 23 2a		13 d5 a1 e4
8a 2a 89 d8		fa 54 a3 6c		70 7e 2a b4
1f d6 5 c1		fe 2c 39 76		e1 fa 3c b7
f8 2f e0 ac		17 b1 39 5		ef 9e d9 a9

Figura 4.19 - Encriptado AES (Operación XOR con Key1).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Este XOR se hace con una segunda clave (Key1) derivada de la clave original (se explicará el proceso de generación de claves más adelante). En total, se generan 10 claves partiendo de la clave original.

3. Las vueltas...

Ahora, se repite 9 veces la secuencia de procesos ya explicados, siguiendo este orden: *SubBytes*, *ShiftRows*, *MixColumns* y XOR. Cada vuelta usa una clave diferente.

Resultados después de cada operación en vuelta 2

SubBytes:	ShiftRows:	MixColumns:	$\oplus$	Key ₂	=	XOR:
7d 3 32 69	7d 3 32 69	cc 44 1b 39		f2 7a 59 73		3e 3e 42 4a
51 f3 e5 8d	f3 e5 8d 51	75 ed 2b 89		c2 96 35 59		b7 7b 1e d0
f8 2d eb a9	eb a9 f8 2d	2d d5 49 3d		95 b9 80 f6		b8 6c c9 cb
df b 35 d3	d3 df b 35	22 ec 35 ad		f2 43 7a 7f		d0 af 4f d2

Figura 4.20 - Encriptado AES (Operación Vuelta 2).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Resultados después de cada operación en vuelta 3

SubBytes:	ShiftRows:	MixColumns:	$\oplus$	Key ₃	=	XOR:
b2 b2 2c d6	b2 b2 2c d6	74 86 dd 83		3d 47 1e 6d		49 c1 c3 ee
a9 21 72 70	21 72 70 a9	39 7 1 eb		80 16 23 7a		b9 11 22 91
6c 50 dd 1f	dd 1f 6c 50	f6 6e f 48		47 fe 7e 88		b1 90 71 c0
70 79 84 b5	b5 70 79 84	40 40 9a 8b		7d 3e 44 3b		3d 7e de b0

Figura 4.21 - Encriptado AES (Operación Vuelta 3).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Resultados después de cada operación en vuelta 4

SubBytes:	ShiftRows:	MixColumns:	$\oplus$	Key ₄	=	XOR:
3b 78 2e 28	3b 78 2e 28	af c3 ff d7		ef a8 b6 db		40 6b 49 c
56 82 93 81	82 93 81 56	3d b7 87 39		44 52 71 b		79 e5 f6 32
c8 60 a3 ba	a3 ba c8 60	d6 ed 2a 99		a5 5b 25 ad		73 b6 f 34
27 f3 1d e7	e7 27 f3 1d	b9 ef c6 74		41 7f 3b 0		f8 90 fd 74

Figura 4.22 - Encriptado AES (Operación Vuelta 4).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Los resultados de la vueltas continúan hasta la 9, a modo de simplificar presentamos el resultado de la vuelta 9:

SubBytes:	ShiftRows:	MixColumns:	$\oplus$	Key ₉	=	XOR:
f5 ac 91 cc	f5 ac 91 cc	1f 22 df 40		ac 19 28 57		b3 3b f7 17
99 7e 84 6c	7e 84 6c 99	9e f3 7a 35		77 fa d1 5c		e9 9 ab 69
bd dc f0 5d	f0 5d bd dc	cf 74 b6 1c		66 dc 29 0		a9 a8 9f 1c
ab ef af 9c	9c ab ef af	a9 7b bc 4f		f3 21 41 6e		5a 5a fd 21

Figura 4.23 - Encriptado AES (Operación Vuelta 9).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

4. Últimos pasos

Resultados después de cada operación y texto cifrado

La vuelta final no hace el MixColumns. Por lo demás es igual a las anteriores.

SubBytes:	ShiftRows:	MixColumns:	$\oplus$	Key ₁₀	=	Finalmente llegamos al texto cifrado tras el último XOR:
6d e2 68 f0	6d e2 68 f0	No se usa MixColumns en la última vuelta		d0 c9 e1 b6		bd 2b 89 46
1e 1 62 f9	1 62 f9 1e			14 ee 3f 63		15 8c c6 7d
d3 c2 db 9c	db 9c d3 c2			f9 25 c c		22 b9 df ce
be be 54 fd	fd be be 54			a8 89 c8 a6		55 37 76 f2

Figura 4.24 - Encriptado AES (Resultado final).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Generación de claves

Se necesitan 10 claves adicionales a la original. Por lo tanto se generan siguiendo este proceso:

Primeros 4 bytes de la nueva clave (bytes 0 - 3)

Para generar los primeros 4 bytes de la nueva clave “key 1” (ver figura 4.25):

1. La clave original (“key0” aparece en el número 1).
2. Reorganiza los últimos 4 bytes de la clave.
3. Sustituye cada byte usando la tabla S-BOX.

4. Se hace un XOR con los 4 bytes de “rcon1” (explicación más adelante).
5. Resultado del XOR anterior.
6. Realiza otro XOR con los primeros 4 bytes de la clave original.
7. Finalmente, se obtienen los primeros 4 bytes de la nueva clave (key 1).

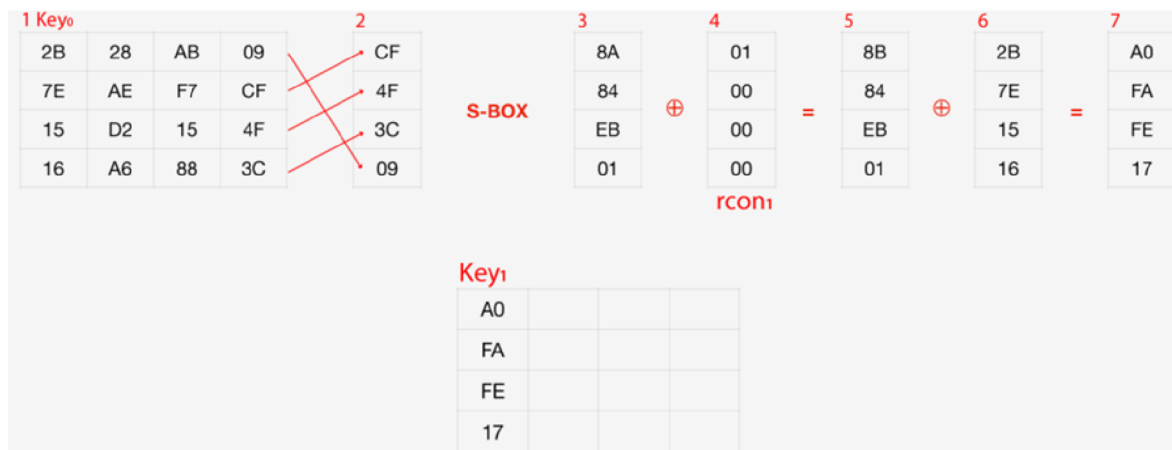


Figura 4.25 - Encriptado AES (Generación de claves).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Bytes 4 a 15

Las próximos bytes se generan de manera mucho más sencilla (ver figura 4.26):

- Bytes 4 a 7 (2da columna): XOR entre los bytes 4 a 7 de la clave original y los 4 bytes anteriores de la nueva clave (en rojo).
- Bytes 8 a 11 (3ra columna): XOR entre los bytes 8 a 11 de la clave original y los 4 bytes anteriores de la nueva clave (en azul).
- Bytes 12 a 15 (4ta columna): XOR entre los bytes 12 a 15 de la clave original y los 4 bytes anteriores de la nueva clave (en verde).

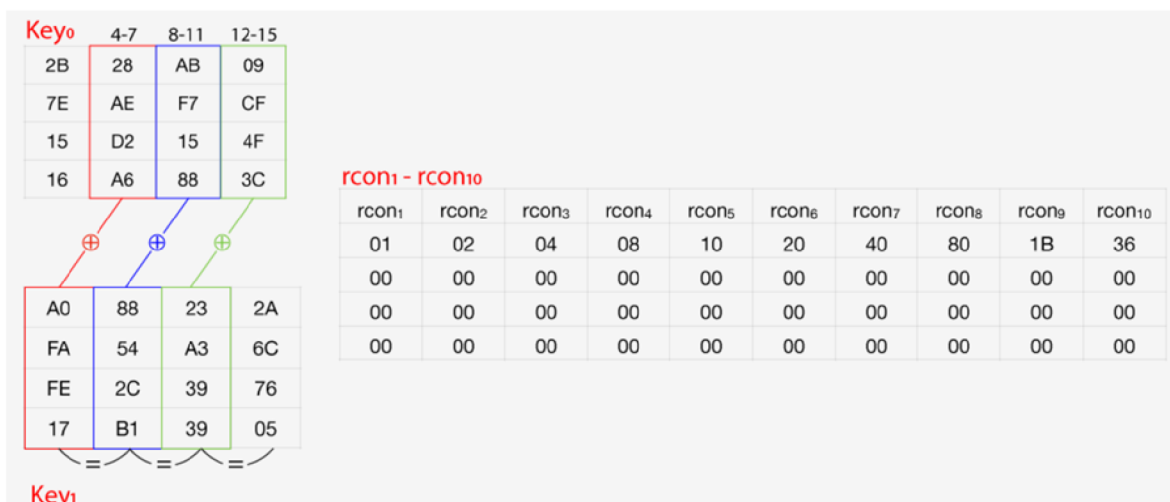


Figura 4.26 - Encriptado AES (Generación de claves).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Claves key2 a key10

Se necesitan 10 claves adicionales a la original:

- Para generar las demás claves se parte siempre de la clave anterior (key2 se genera partiendo de key1).
- Los valores de “rcon” cambian para cada clave (“rcon2” para generar “key2”, “rcon3” para “key3”, etc.). Se puede ver los valores de “rcon” (hexadecimales) en la imagen anterior.

4.10.2 AES Decifrado

1. Primer paso

El primer paso es realizar un XOR ($\oplus$) entre cada célula del state y de la clave 10 (key) seguido del inverso de ShiftRows y SubBytes:

State (texto cifrado)	$\oplus$	Key ₁₀ :	=																																																
<table> <tr><td>bd</td><td>2b</td><td>89</td><td>46</td></tr> <tr><td>15</td><td>8c</td><td>c6</td><td>7d</td></tr> <tr><td>22</td><td>b9</td><td>df</td><td>ce</td></tr> <tr><td>55</td><td>37</td><td>76</td><td>f2</td></tr> </table>	bd	2b	89	46	15	8c	c6	7d	22	b9	df	ce	55	37	76	f2		<table> <tr><td>d0</td><td>c9</td><td>e1</td><td>b6</td></tr> <tr><td>14</td><td>ee</td><td>3f</td><td>63</td></tr> <tr><td>f9</td><td>25</td><td>c</td><td>c</td></tr> <tr><td>a8</td><td>89</td><td>c8</td><td>a6</td></tr> </table>	d0	c9	e1	b6	14	ee	3f	63	f9	25	c	c	a8	89	c8	a6																	
bd	2b	89	46																																																
15	8c	c6	7d																																																
22	b9	df	ce																																																
55	37	76	f2																																																
d0	c9	e1	b6																																																
14	ee	3f	63																																																
f9	25	c	c																																																
a8	89	c8	a6																																																
State $\oplus$ con Key ₁₀	ShiftRows	SubBytes																																																	
<table> <tr><td>6d</td><td>e2</td><td>68</td><td>f0</td></tr> <tr><td>1</td><td>62</td><td>f9</td><td>1e</td></tr> <tr><td>db</td><td>9c</td><td>d3</td><td>c2</td></tr> <tr><td>fd</td><td>be</td><td>be</td><td>54</td></tr> </table>	6d	e2	68	f0	1	62	f9	1e	db	9c	d3	c2	fd	be	be	54	<table> <tr><td>6d</td><td>e2</td><td>68</td><td>f0</td></tr> <tr><td>1e</td><td>1</td><td>62</td><td>f9</td></tr> <tr><td>d3</td><td>c2</td><td>db</td><td>9c</td></tr> <tr><td>be</td><td>be</td><td>54</td><td>fd</td></tr> </table>	6d	e2	68	f0	1e	1	62	f9	d3	c2	db	9c	be	be	54	fd	<table> <tr><td>b3</td><td>3b</td><td>f7</td><td>17</td></tr> <tr><td>e9</td><td>9</td><td>ab</td><td>69</td></tr> <tr><td>a9</td><td>a8</td><td>9f</td><td>1c</td></tr> <tr><td>5a</td><td>5a</td><td>fd</td><td>21</td></tr> </table>	b3	3b	f7	17	e9	9	ab	69	a9	a8	9f	1c	5a	5a	fd	21	
6d	e2	68	f0																																																
1	62	f9	1e																																																
db	9c	d3	c2																																																
fd	be	be	54																																																
6d	e2	68	f0																																																
1e	1	62	f9																																																
d3	c2	db	9c																																																
be	be	54	fd																																																
b3	3b	f7	17																																																
e9	9	ab	69																																																
a9	a8	9f	1c																																																
5a	5a	fd	21																																																

Figura 4.27 - Decifrado AES (XOR ($\oplus$) entre cada célula del state y de la clave 10 (key)).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

¿Cómo se realiza el inverso de los procesos XOR, ShiftRows y SubBytes?

XOR

El XOR es sencillo:

- Si $a \oplus b = c$
- Entonces $c \oplus b = a$

En otras palabras, la misma operación devuelve el original.

❖ ShiftRows

Se invierten los movimientos.

❖ SubBytes

Ahora se busca el valor en la tabla, y de ahí se saca la línea y columna. 6D (de la tabla ShiftRows precedente) está en la línea B y la columna 3. Por lo tanto el resultado es B3 (en la tabla SubBytes).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 4.28 - Decifrado AES (Tabla S-BOX).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

Proceso inverso Vuelta 9 (Comienza de manera inversa)

Key₉

ac	19	28	57
77	fa	d1	5c
66	dc	29	0
f3	21	41	6e

SubBytes $\oplus$ Key₉

1f	22	df	40
9e	f3	7a	35
cf	74	b6	1c
a9	7b	bc	4f

MixColumns

f5	ac	91	cc
7e	84	6c	99
f0	5d	bd	dc
9c	ab	ef	af

ShiftRows

f5	ac	91	cc
99	7e	84	6c
bd	dc	f0	5d
ab	ef	af	9c

SubBytes

77	aa	ac	27
f9	8a	4f	b8
cd	93	17	8d
e	61	1b	1c

Figura 4.29 - Decifrado AES (Vuelta 9 proceso inverso).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

¿Cómo se realiza el inverso del proceso MixRows?

Para el proceso de MixRows se utiliza la siguiente matriz:

0e	0b	0d	09
09	0e	0b	0d
0d	09	0e	0b
0b	0d	09	0e

Figura 4.30 - Decifrado AES (Matriz de proceso inverso MixRows).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

2. Paso final

Luego de realizar toda la inversa y llegar finalmente a la “Vuelta 1”, se procede a decodificar el texto:

Key₀

2b	28	ab	9
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

SubBytes ⊕ Key₀

4d	61	73	65
65	6a	65	74
6e	65	63	6f
73	20	72	2e

=

Texto decodificado

M	a	s	e
e	j	e	t
n	e	c	o
s		r	.

Figura 4.31 - Decifrado AES (Operación final de decodificación del mensaje).

Fuente: [https://www.teoria.com/jra/aes/encrypt.html\(2023\)](https://www.teoria.com/jra/aes/encrypt.html(2023))

4.10.3 AES Ejemplo práctico

A continuación se presenta un ejemplo del algoritmo AES implementado en Arduino IDE para este trabajo, el cual tiene el fin de cifrar una cadena de datos, en este caso de tipo string.

```
#include "mbedtls/aes.h"
#include <cstring>

void encrypt(char * plainText, char * key, unsigned char * outputBuffer){
```

```
mbdtdls_aes_context aes;
mbdtdls_aes_init(&aes);
mbdtdls_aes_setkey_enc(&aes, (const unsigned char*) key, strlen(key) * 8);

unsigned char paddedPlainText[16];
memcpy(paddedPlainText, plainText, strlen(plainText));
memset(paddedPlainText + strlen(plainText), 0, sizeof(paddedPlainText) -
strlen(plainText));

    mbdtdls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, paddedPlainText,
outputBuffer);
    mbdtdls_aes_free(&aes);
}
void setup() {
    Serial.begin(115200);
}
void loop() {
    char myString[] = "Hola pirulo";
    char * key = "abcdefghijklmnp";
    unsigned char cipherTextOutput[16];
    encrypt(myString, key, cipherTextOutput);
    char encryptedData[33]; // Para almacenar el resultado cifrado como
cadena hexadecimal
    encryptedData[32] = '\0'; // Terminador de la cadena
    for (int i = 0; i < 16; i++) {
        sprintf(&encryptedData[i*2], "%02x", (int)cipherTextOutput[i]); //
Almacenar cada byte cifrado como dos caracteres hexadecimales
    }
    Serial.println(encryptedData);
    delay(5000);
}
```

Inicialmente, el código incluye las bibliotecas necesarias (*mbdtdls/aes.h* y *cstring*) para realizar el cifrado AES.

Se define una función **encrypt**, la cual toma un texto en plano (*plainText*), una clave de cifrado (*key*) y un buffer de salida (*outputBuffer*). Esta función realiza el cifrado AES del texto en claro utilizando la clave proporcionada y guarda el resultado en el buffer de salida.

Dentro de la función **encrypt**, inicializa el contexto AES, configura la clave de cifrado y rellena el texto en claro con ceros si es necesario para alcanzar el tamaño de bloque requerido por AES (16 bytes). Luego, realiza el cifrado AES y libera el contexto AES.

La función **setup()** inicializa la comunicación serie a una tasa de baudios de 115200.

La función **loop()** se ejecuta continuamente. En cada iteración, define una cadena y una clave, realiza el cifrado AES de la cadena utilizando la función **encrypt** y almacena el resultado en un buffer de salida.

Luego, el programa convierte el texto cifrado (en formato binario) a una cadena de caracteres hexadecimales y la imprime en el puerto serie.

Finalmente, el programa espera 5 segundos (5000 milisegundos) antes de comenzar la próxima iteración del bucle.

4.11 Algoritmo de cifrado XOR

El algoritmo de cifrado XOR es un algoritmo simple, su funcionamiento consta de:

1. **Operación XOR:** La operación XOR compara dos bits y devuelve 1 si los bits son diferentes y 0 si son iguales. Por ejemplo, si se tiene dos bits A y B, entonces la operación XOR se representaría como $A \text{ XOR } B$. Aquí están todas las posibilidades:
 - $0 \text{ XOR } 0 = 0$
 - $0 \text{ XOR } 1 = 1$
 - $1 \text{ XOR } 0 = 1$
 - $1 \text{ XOR } 1 = 0$
2. **Cifrado:** Para cifrar un mensaje con XOR, se toma un carácter del mensaje y un carácter de la clave, y los combina con la operación XOR. Esto te da el carácter cifrado. Este proceso se repite para cada carácter del mensaje.
3. **Descifrado:** Para descifrar un mensaje con XOR, realizas el mismo proceso que para el cifrado. Esto se debe a que la operación XOR tiene la propiedad de que $A \text{ XOR } B \text{ XOR } B = A$. Por lo tanto, si se toma el carácter cifrado y se combina con el mismo carácter de la clave que se utilizó para cifrar, se obtiene el carácter original.

La seguridad del cifrado XOR depende completamente de la clave. Si la clave es tan larga como el mensaje y se utiliza una vez, entonces se tiene lo que se llama un "cifrado de un solo uso", que es teóricamente irrompible. Sin embargo, si la clave es más corta que el

mensaje y se repite, o si se utiliza la misma clave para cifrar varios mensajes, entonces un atacante puede usar técnicas de análisis de frecuencias y otros métodos para romper el cifrado.

Se escogió este algoritmo con el fin de poder realizar una implementación personal de cifrado, probar nuevas funcionalidades que otorgar EMQX versión 5 (como el uso de scripts de java para el broker) y realizar la prueba de que tan eficiente es el mismo a nivel de seguridad en el encriptado de mensajes.

4.12 Python

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general.

El mismo posee características tales como:

- **Sintaxis simple y clara:** El diseño de Python se centra en la legibilidad del código, que utiliza un formato de espacios en blanco y una sintaxis que permite a los programadores expresar conceptos en menos líneas de código que sería posible en otros lenguajes como C++ o Java.
- **Tipado dinámico y gestión automática de memoria:** Python es dinámicamente tipado, lo que significa que no se requiere definir el tipo de variable al declararla. Además, Python maneja la memoria automáticamente mediante un recolector de basura, que libera la memoria utilizada por los objetos que ya no están en uso.
- **Interpretado:** Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea, lo que facilita la depuración y hace que el código sea más portable ya que puede ser ejecutado en cualquier plataforma que tenga un intérprete de Python.
- **Multiparadigma:** Python admite varios paradigmas de programación, incluyendo programación procedural, orientada a objetos y funcional.
- **Extensibilidad:** Python puede integrarse con otros lenguajes de programación como C y C++, lo que permite aprovechar la velocidad de estos lenguajes cuando es necesario.

- **Gran biblioteca estándar y ecosistema de terceros:** Python tiene una amplia biblioteca estándar que incluye módulos para manejar archivos, conexiones a Internet, protocolos de web, y mucho más. Además, hay un enorme ecosistema de bibliotecas de terceros para Python que permiten todo, desde el desarrollo web y de aplicaciones hasta el análisis de datos y el aprendizaje automático.

Python se utiliza en una amplia variedad de aplicaciones, incluyendo desarrollo web, análisis de datos, aprendizaje automático, inteligencia artificial, automatización, desarrollo de software y mucho más. Es particularmente popular en la comunidad científica y de datos por su facilidad de uso y la disponibilidad de bibliotecas como NumPy, Pandas y PyCrypto.

4.12.1 ¿Porque python?

A continuación se presenta un cuadro comparativo entre las características principales de los lenguajes Python, Java, C++ y JavaScript, indicando también las diferencias sobre el soporte de bibliotecas de criptografía y su uso.

Características	Python	Java	C++	JavaScript
Tipado	Dinámico	Estático	Estático	Dinámico
Sintaxis	Muy limpia	Verbose	Compleja	Moderada
Aprendizaje	Fácil	Moderado	Difícil	Moderado
Uso general	Análisis de datos, IA, ML, desarrollo web, scripting	Desarrollo de aplicaciones empresariales, móviles y web	Desarrollo de sistemas, juegos	Desarrollo web
Bibliotecas Criptografía	PyCrypto, cryptography, M2Crypto, hashlib	Java Cryptography Extension (JCE), BouncyCastle	Crypto++, Botan	CryptoJS, bcrypt.js, Node.js Crypto module
Seguridad en Criptografía	Alta (Buenas bibliotecas y buen soporte comunitario)	Alta (Ampliamente usada en aplicaciones)	Alta (El control de más bajo nivel puede proporcionar)	Media (Dependiente del entorno, buen soporte en)

		empresariales seguras)	seguridad adicional si se usa correctamente)	entornos Node.js)
Facilidad de Uso en Criptografía	Alta (Sintaxis sencilla y legible, gran cantidad de bibliotecas)	Media (Sintaxis más compleja, pero buenas bibliotecas)	Media/Baja (Sintaxis compleja y nivel de abstracción más bajo)	Media/Alta (Sintaxis sencilla, pero enfoque principal en la web)

Python se escogió para este trabajo, ya que se destaca por su sintaxis sencilla y legible, así como por su gran colección de bibliotecas de criptografía. Esto lo hace ideal para la implementación de diversos protocolos criptográficos y su respectivo cifrado. Las bibliotecas de criptografía en Python son bastante robustas y se mantienen actualizadas, lo que proporciona una seguridad confiable para la aplicación.

4.12.2 Scripts Python

Python es una excelente opción para la creación y utilización de scripts. Entre las razones se encuentra su facilidad de uso. Python es conocido por su sintaxis clara y fácil de entender, que se asemeja al inglés. Esto facilita su lectura y comprensión en comparación con muchos otros lenguajes de programación.

Además, Python cuenta con una gran cantidad de bibliotecas y módulos integrados que resultan útiles para diversas tareas. Estos pueden asistir en actividades que van desde el manejo de archivos hasta la realización de cálculos matemáticos complejos, como por ejemplo la criptografía.

Otro beneficio de Python es su portabilidad. Los scripts de Python pueden ejecutarse en cualquier sistema operativo que tenga instalado un intérprete de Python, lo que aporta mucha flexibilidad.

Python también es excelente para la automatización de tareas. Puede ser muy eficaz para automatizar acciones que de otra forma serían tediosas o complicadas, como mover o renombrar archivos en masa, extraer información de la web o realizar cálculos complejos.

La gran comunidad de soporte de Python es otro punto fuerte. Si se encuentran problemas, hay numerosos recursos disponibles, incluyendo tutoriales, foros y libros que pueden brindarte ayuda.

Implementación del algoritmo de cifrado

5.1 Puesta en funcionamiento del esquema

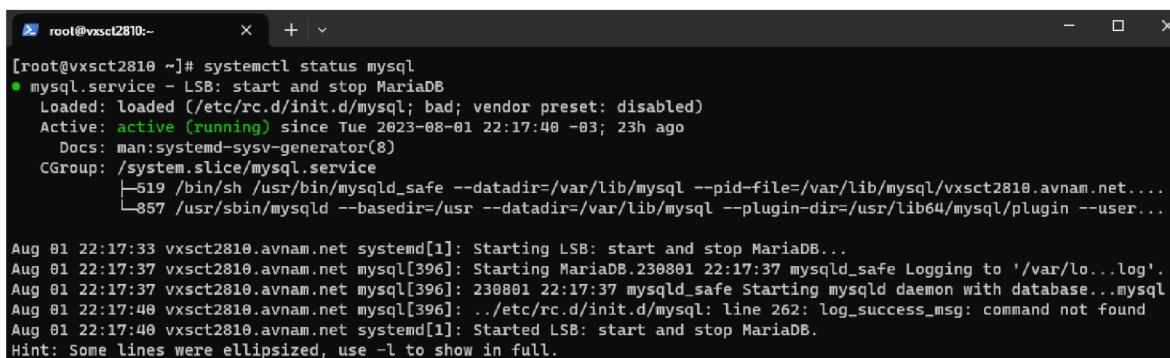
Como primer paso tenemos que poner en funcionamiento todos los servicios configurados requeridos para recibir nuestro mensaje en el servidor.

Como aclaración principal, todo está instalado y corriendo en un servidor proporcionado por la UNAJ, el cual funciona como máquina virtual.

5.1.1 MySQL inicio

Verificamos que nuestra base de datos esté encendida y funcionando con el siguiente comando (El estado debe encontrarse en “**active (running)**”):

- `systemctl status mysql`



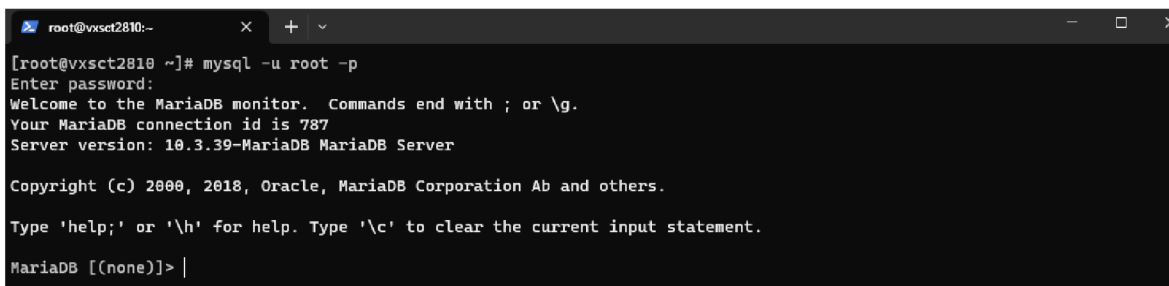
```
root@vxsc2810:~  
[root@vxsc2810 ~]# systemctl status mysql  
● mysql.service - LSB: start and stop MariaDB  
   Loaded: loaded [/etc/rc.d/init.d/mysql; bad; vendor preset: disabled]  
   Active: active (running) since Tue 2023-08-01 22:17:40 -03; 23h ago  
     Docs: man:systemd-sysv-generator(8)  
   CGroup: /system.slice/mysql.service  
           └─819 /bin/sh /usr/bin/mysqld_safe --datadir=/var/lib/mysql --pid-file=/var/lib/mysql/vxsc2810.avnam.net...  
           └─857 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-dir=/usr/lib64/mysql/plugin --user...  
  
Aug 01 22:17:33 vxsc2810.avnam.net systemd[1]: Starting LSB: start and stop MariaDB...  
Aug 01 22:17:37 vxsc2810.avnam.net mysql[396]: Starting MariaDB.230801 22:17:37 mysqld_safe Logging to '/var/lo...log'.  
Aug 01 22:17:37 vxsc2810.avnam.net mysql[396]: 230801 22:17:37 mysqld_safe Starting mysqld daemon with database...mysql  
Aug 01 22:17:40 vxsc2810.avnam.net mysql[396]: ../etc/rc.d/init.d/mysql: line 262: log_success_msg: command not found  
Aug 01 22:17:40 vxsc2810.avnam.net systemd[1]: Started LSB: start and stop MariaDB.  
Hint: Some lines were ellipsized, use -l to show in full.
```

Figura 5.1 - MySql (Runnig)

Fuente: Propia

Una vez verificado esto pasamos a ingresar a la base de datos con nuestro usuario root, con el siguiente comando:

- `mysql -u root -p 04nsUPTfC1tjy5q8`



```
root@vxsc2810:~# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 787
Server version: 10.3.39-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> |
```

Figura 5.2 - MySql (Conexion)

Fuente: Propia

Seleccionamos la BD creada para este trabajo, llamada “mqtt”, donde se encuentran los usuarios autenticados, con los siguientes comandos siguiendo el orden de los mismos:

1. `SHOW databases;`
2. `SELECT user,Host FROM mysql.db WHERE db = "mqtt";`
3. `SHOW tables;`
4. `SELECT * FROM mqtt_user;`

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| db_mysql |
| dbmysql78 |
| information_schema |
| test_group_con |
| mqtt |
| mysql |
| performance_schema |
| proyecto_sondas |
| test_crypto |
| vxset1808_avnan_net_roundcube |
+-----+
10 rows in set (0.054 sec)

MariaDB [(none)]> use mqtt;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Figura 5.3 - MySql (Base de datos)

Fuente: Propia

Figura 5.4 - MySql (Usuarios)

Fuente: Propia

5.1.2 Emqx inicio

- `emqx_ctl status`

Figura 5.5 - Emqx(Status)

Fuente: Propia

Una vez que comprobamos que el mismo está corriendo vamos a conectarnos a el dashboard proporcionado, para verificar que todo funciona de manera correcta:

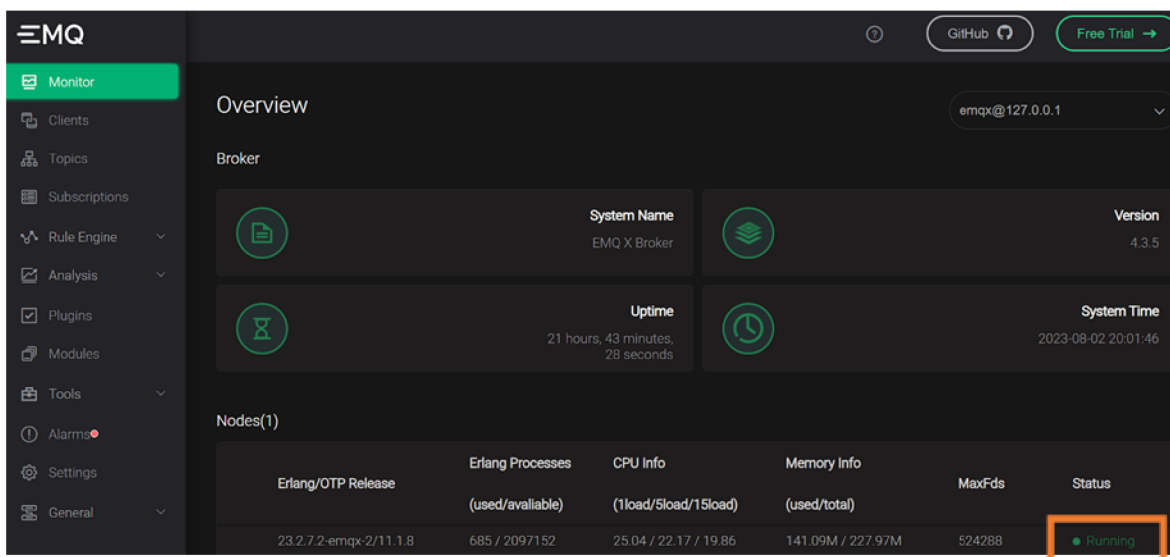


Figura 5.6 - Emqx(Dashboard)

Fuente: Propia

Verificamos que el proceso “Status” se encuentre en modo “Running” como se observa en la (Figura 5.6), esto indica que nuestro broker está activo.

Seguido de esto verificamos los puertos MQTT “8083” y “8084 (Dedicados a la conexión segura con certificado SSL)” encargados de recibir los mensajes estén habilitados, para esto utilizamos el comando:

- `netstat -tuln | grep 8083`
- `netstat -tuln | grep 8084`

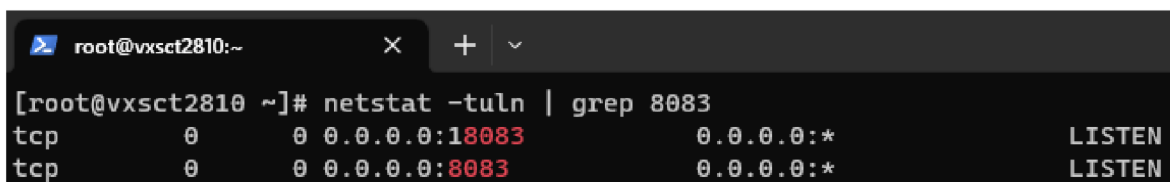


Figura 5.7 - Puerto (8083)

Fuente: Propia

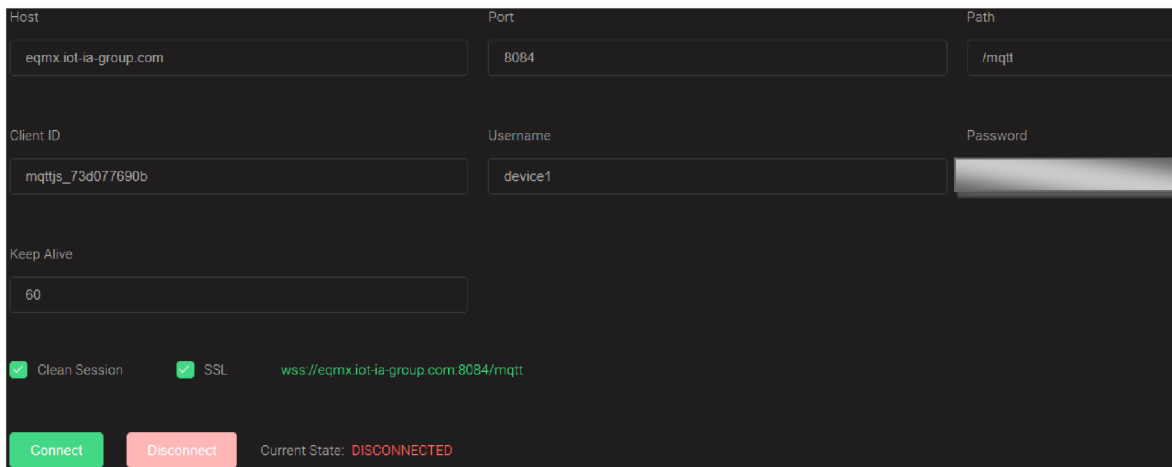


Figura 5.8 - Puerto (8084)

Fuente: Propia

Como podemos observar los puertos se encuentran en estado “LISTEN” o "escuchando" en español, lo cual significa que los mismos están habilitados para recibir mensajes.

Para el siguiente paso tenemos que conectarnos al WebSocket, para ello vamos a realizar una conexión segura mediante SSL, el usuario a utilizar es “**device1**”, el cual posee permisos para poder conectarse mediante certificación gracias a la autenticación de MySQL. La configuración de lo mencionado anteriormente se encuentra en la sección (*Autenticación con MySQL(Emqx) Pag 22*).



The screenshot shows a configuration window for an MQTT client. The fields are as follows:

Host	Port	Path
eqmx.iot-ia-group.com	8084	/mqtt

Client ID	Username	Password
mqttjs_73d077690b	device1	[Redacted]

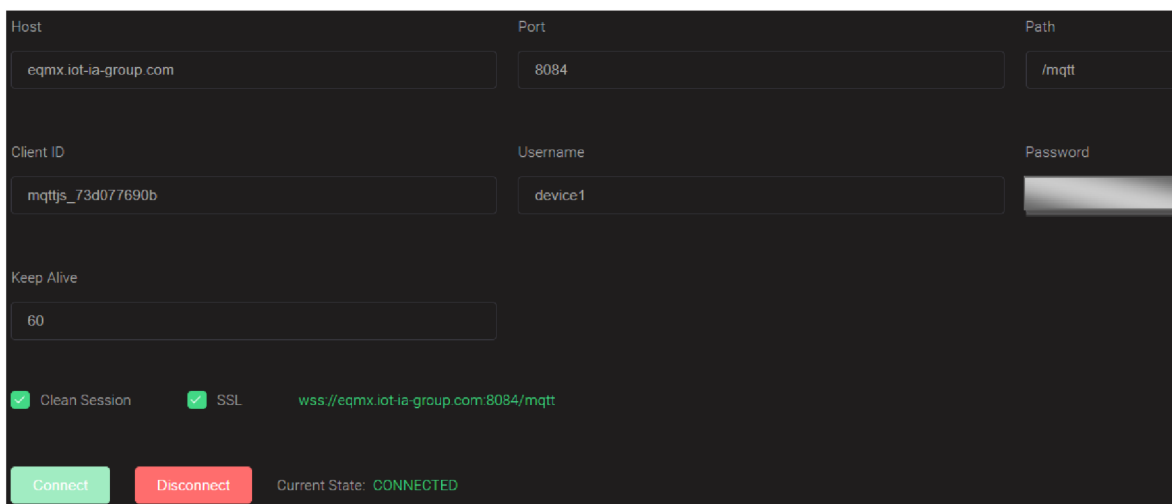
Keep Alive: 60

☒ Clean Session ☒ SSL wss://eqmx.iot-ia-group.com:8084/mqtt

 Current State: DISCONNECTED

Figura 5.9 - Conexión a WebSocket SSL.

Fuente: Propia



This screenshot is identical to the one in Figure 5.9, but the 'Current State' at the bottom right is now 'CONNECTED'.

 Current State: CONNECTED

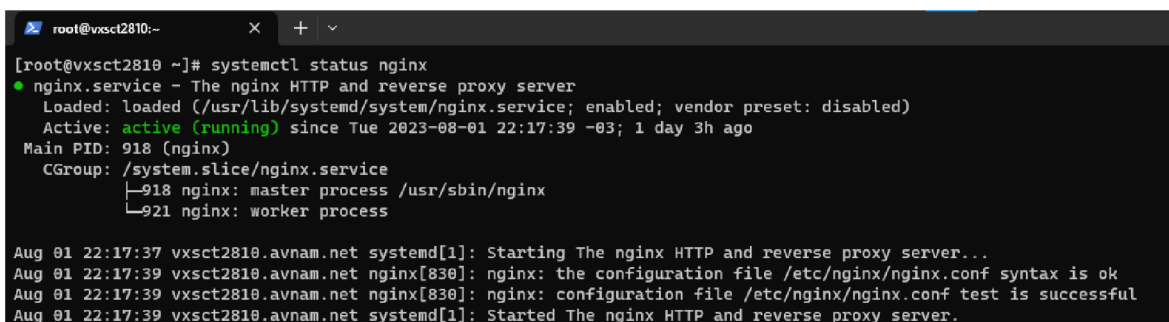
Figura 5.10 - Conexión a WebSocket SSL (Exitosa).

Fuente: Propia

Hasta aquí llegaremos con este punto, ya que la visualización de los datos en el dashboard se mostrará en la sección de resultados.

5.1.3 Nginx inicio

Este punto es sencillo ya que solo se debe verificar que Nginx esté activo o en estado “Running”, lo cual se muestra en la (Figura 5.11).



```
root@vxsc2810:~  
[root@vxsc2810 ~]# systemctl status nginx  
● nginx.service - The nginx HTTP and reverse proxy server  
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset: disabled)  
   Active: active (running) since Tue 2023-08-01 22:17:39 -03; 1 day 3h ago  
     Main PID: 918 (nginx)  
    CGroup: /system.slice/nginx.service  
            └─918 nginx: master process /usr/sbin/nginx  
              └─921 nginx: worker process  
  
Aug 01 22:17:37 vxsc2810.avnam.net systemd[1]: Starting The nginx HTTP and reverse proxy server...  
Aug 01 22:17:39 vxsc2810.avnam.net nginx[830]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok  
Aug 01 22:17:39 vxsc2810.avnam.net nginx[830]: nginx: configuration file /etc/nginx/nginx.conf test is successful  
Aug 01 22:17:39 vxsc2810.avnam.net systemd[1]: Started The nginx HTTP and reverse proxy server.
```

Figura 5.11 - Nginx inicio.

Fuente: Propia

Para esto se utilizó el comando:

- `systemctl status nginx`

En caso de que la aplicación se hubiese detenido, solo se debe usar el siguiente comando para iniciarla nuevamente:

- `systemctl start nginx`

5.1.4 NodeRed inicio

Para el inicio de NodeRed es de suma importancia verificar que el puerto “1880” ya que es el que viene por defecto para poder acceder a la aplicación. En la siguiente imagen podemos observar como el puerto se encuentra abierto, indicando así “LISTEN”:



```
[root@vxsc2810 ~]# netstat -tuln | grep 1880  
tcp        0      0 0.0.0.0:1880          0.0.0.0:*           LISTEN
```

Figura 5.12 - Puerto (1880)

Fuente: Propia

Finalmente con el siguiente comando iniciamos el programa:

- `node-red`

```
Welcome to Node-RED
=====

3 Aug 02:34:58 - [info] Node-RED version: v2.2.2
3 Aug 02:34:58 - [info] Node.js version: v17.8.0
3 Aug 02:34:58 - [info] Linux 5.4.73-1-pve x64 LE
3 Aug 02:35:01 - [info] Loading palette nodes
3 Aug 02:35:17 - [info] Dashboard version 3.1.7 started at /ui
3 Aug 02:35:17 - [info] Settings file : /root/.node-red/settings.js
3 Aug 02:35:17 - [info] Context store : 'default' [module=memory]
3 Aug 02:35:17 - [info] User directory : /root/.node-red
3 Aug 02:35:17 - [warn] Projects disabled : editorTheme.projects.enabled=false
3 Aug 02:35:17 - [info] Flows file : /root/.node-red/flows.json
3 Aug 02:35:17 - [error] Uncaught Exception:
3 Aug 02:35:17 - [error] Error: listen EADDRINUSE: address already in use 0.0.0.0:1880
    at Server.setupListenHandle [as _listen2] (node:net:1372:16)
    at listenInCluster (node:net:1420:12)
    at doListen (node:net:1559:7)
    at processTicksAndRejections (node:internal/process/task_queues:84:21)
```

Figura 5.13 - NodeRed inicio

Fuente: Propia

5.1.5 Grafana inicio

Para este paso simplemente verificamos que Grafana esté corriendo en nuestro servidor, o en estado “Running”, lo cual se muestra en la (Figura 5.14).

```
root@vxsc2810:~
[root@vxsc2810 ~]# systemctl status grafana-server
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2023-08-02 10:54:11 -03; 1 day 16h ago
     Docs: http://docs.grafana.org
    Main PID: 13459 (grafana)
    CGroup: /system.slice/grafana-server.service
            └─13459 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile:
              └─13464 /var/lib/grafana/plugins/grafana-google-sheets-datasource/gpx_sheets_linux_amd64
                └─13470 /var/lib/grafana/plugins/hadesarchitect-cassandra-datasource/cassandra-plugin_lin

Aug 04 02:54:11 vxsc2810.avnam.net grafana[13459]: logger=plugins.update.checker t=2023-08-04T02:54:11.185444039-03
Aug 04 03:04:11 vxsc2810.avnam.net grafana[13459]: logger=grafana.update.checker t=2023-08-04T03:04:11.185444039-03
Aug 04 03:04:11 vxsc2810.avnam.net grafana[13459]: logger=plugins.update.checker t=2023-08-04T03:04:11.185444039-03
Aug 04 03:14:11 vxsc2810.avnam.net grafana[13459]: logger=cleanup t=2023-08-04T03:14:11.185444039-03
Aug 04 03:14:11 vxsc2810.avnam.net grafana[13459]: logger=grafana.update.checker t=2023-08-04T03:14:11.185444039-03
Aug 04 03:14:11 vxsc2810.avnam.net grafana[13459]: logger=plugins.update.checker t=2023-08-04T03:14:11.185444039-03
Aug 04 03:24:11 vxsc2810.avnam.net grafana[13459]: logger=cleanup t=2023-08-04T03:24:11.185444039-03
Aug 04 03:24:11 vxsc2810.avnam.net grafana[13459]: logger=grafana.update.checker t=2023-08-04T03:24:11.185444039-03
Aug 04 03:24:12 vxsc2810.avnam.net grafana[13459]: logger=plugins.update.checker t=2023-08-04T03:24:11.185444039-03
```

Figura 5.14 - Grafana (Running)

Fuente: Propia

Para esto se utilizó el comando:

- `systemctl status nginx`

5.2 Algoritmo de cifrado XOR e implementación de librería mbedtls

5.2.1 Algoritmo de cifrado XOR (Cifrado de datos)

A continuación se presenta el algoritmo XOR implementado en Arduino IDE , el cual tiene el fin de encriptar un mensaje de 16 caracteres que se genera de manera aleatoria, simulando una cadena de mensajes x que podría enviar el dispositivo. Luego este mismo finaliza enviando el mensaje cifrado al broker EMQX versión 5 (**MQTTX**) de prueba.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "Seeed_mbedtls.h"

const char* mqttServer = "broker.emqx.io";
const int mqttPort = 1883;
const char* mqttTopic = "Salida/01";
const char* ssid = "*****";
const char* password = "*****";

WiFiClient espClient;
PubSubClient client(espClient);

// Clave de cifrado
char key[] = "0123456789ABCDEF";
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }
  Serial.println("Conectado a la red WiFi");
  client.setServer(mqttServer, mqttPort);
  while (!client.connected()) {
    if (client.connect("ESP32Client")) {
      Serial.println("Conectado al broker MQTT");
    } else {
      Serial.print("Error de conexión al broker MQTT: ");
      Serial.println(client.state());
      delay(2000);
    }
  }
}

void loop() {
  if (!client.connected()) {
```

```

    reconnect();
}
client.loop();
// Generar un dato aleatorio de hasta 16 caracteres
char randomData[17];
for (int i = 0; i < 16; i++) {
    randomData[i] = random(33, 126); // Caracteres imprimibles ASCII
}
randomData[16] = '\0'; // Terminador de cadena
// Cifrar el dato
char encryptedData[16];
encryptData(randomData, encryptedData);
// Publicar el dato cifrado en el tópico
client.publish(mqttTopic, encryptedData);
Serial.println("El dato original es:");
Serial.println(randomData);
Serial.println("El cifrado es:");
Serial.println(encryptedData);
delay(5000);
}
void encryptData(const char* data, char* encryptedData) {
    int dataLen = strlen(data);
    for (int i = 0; i < dataLen; i++) {
        encryptedData[i] = data[i] ^ key[i % 16];
    }
    encryptedData[dataLen] = '\0'; // Terminador de cadena
}
void reconnect() {
    while (!client.connected()) {
        Serial.println("Intentando conexión MQTT...");
        if (client.connect("ESP32Client")) {
            Serial.println("Conectado al broker MQTT");
        } else {
            Serial.print("Error de conexión al broker MQTT: ");
            Serial.println(client.state());
            delay(2000);
        }
    }
}
}

```

Para cada carácter del dato a cifrar (“**data**”), el algoritmo realiza una operación XOR (bit a bit) con un carácter correspondiente de la clave (“**key**”). En este caso, la clave se repite cada 16 caracteres, ya que tiene una longitud de 16 caracteres.

La operación XOR se realiza en esta línea de código: “**encryptedData[i] = data[i] ^ key[i % 16]**”;

El siguiente carácter, “^” es el operador XOR en C++. Este operador compara cada bit de sus dos operandos. Si los bits son diferentes, el resultado es 1 (verdadero); si los bits son iguales, el resultado es 0 (falso).

“**i % 16**” se utiliza para repetir la clave cada 16 caracteres. “%” es el operador de módulo en C++, que da el residuo de la división de “i” entre 16.

Luego, cada carácter cifrado se almacena en `encryptedData`.

Finalmente, el algoritmo añade un carácter de terminación de cadena ('\0') al final de “**encryptedData**”.

Luego de todo este proceso se publica el mensaje cifrado al tópico, en la línea “**client.publish(mqttTopic, encryptedData);**”.

5.2.2 Algoritmo de cifrado XOR (Descifrado de datos)

El algoritmo de descifrado que se muestra a continuación, es el utilizado para descifrar el mensaje enviado al broker MQTTX en la sección anterior (*Algoritmo de cifrado XOR (Cifrado de datos) Pag 90.*)

```
const key = "0123456789ABCDEF";

function encryptData(data) {
  const encryptedData = [];
  for (let i = 0; i < data.length; i++) {
    const encryptedChar = String.fromCharCode(data.charCodeAt(i) ^
key.charCodeAt(i % 16));
    encryptedData.push(encryptedChar);
  }
  return encryptedData.join('');
}

execute(encryptData)
```

const key = "0123456789ABCDEF"; define la clave que se utilizará para el cifrado. Esta clave se repetirá si el texto de entrada es más largo que 16 caracteres.

“**function encryptData(data)**” define la función que descifra la cadena de entrada.

“**const encryptedData = []**”; inicializa un arreglo vacío que almacenará los caracteres descifrados.

“**for (let i = 0; i < data.length; i++)**” es un bucle que recorre cada carácter de la cadena de entrada.

Dentro del bucle, “**const encryptedChar = String.fromCharCode(data.charCodeAt(i) ^ key.charCodeAt(i % 16))**”; descifra cada carácter. Primero obtiene el código ASCII de un carácter de la cadena de entrada y de un carácter de la clave, luego realiza una operación XOR en estos códigos ASCII, y finalmente convierte el resultado de nuevo a un carácter. Este carácter descifrado se almacena en “**encryptedChar**”.

“**encryptedData.push(encryptedChar)**”; añade el carácter descifrado al final del arreglo “**encryptedData**”.

Una vez que todos los caracteres de la cadena de entrada han sido descifrados, “**return encryptedData.join('')**”; convierte el arreglo “**encryptedData**” de nuevo a una cadena y la devuelve.

Finalmente “**execute(encryptData)**” ejecuta la función de descifrado para poder leer el mensaje original.

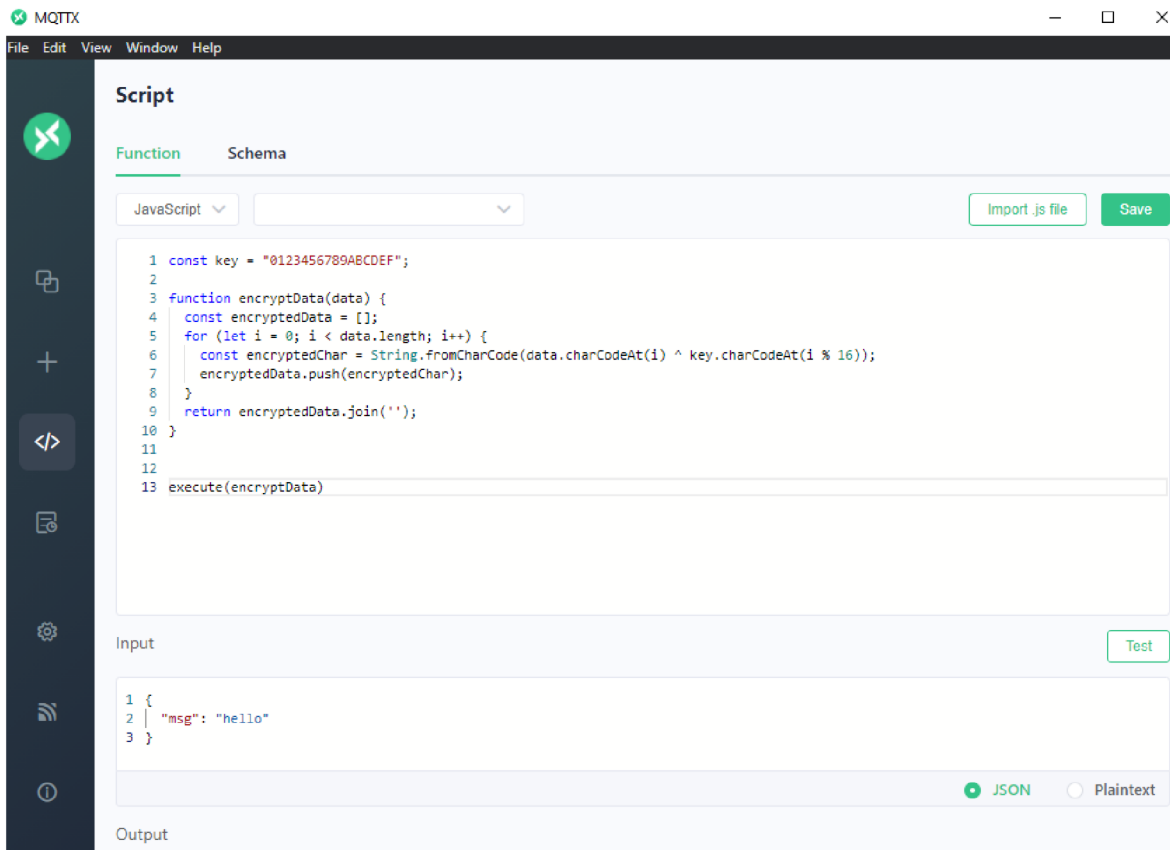


Figura 5.15 - Script descifrado MQTTX

Fuente: Propia

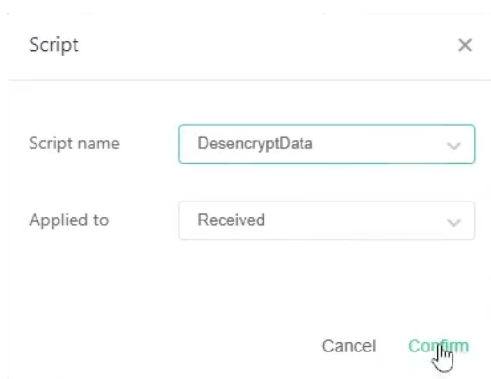


Figura 5.16 - Script descifrado MQTTX (Puesto en funcionamiento)

Fuente: Propia

En la sección de (Resultados Pag 106.) se procederá a mostrar el mensaje descifrado.

5.2.3 Implementación de librería mbedtls Cifrado

A continuación se presenta el algoritmo AES implementado en Arduino IDE, a través de la librería “mbedtls/aes.h” el cual tiene el fin de encriptar un mensaje de 16 caracteres, que para esta prueba se simuló como una cadena de un mensaje previamente escrito, con el fin de poder verificar que el mismo se descifra de manera correcta.

```
#include "mbedtls/aes.h"
#include <cstring>

void encrypt(char * plainText, char * key, unsigned char * outputBuffer){
    mbedtls_aes_context aes;
    mbedtls_aes_init(&aes);
    mbedtls_aes_setkey_enc(&aes, (const unsigned char*) key, strlen(key) * 8);

    unsigned char paddedPlainText[16];
    memcpy(paddedPlainText, plainText, strlen(plainText));
    memset(paddedPlainText + strlen(plainText), 0, sizeof(paddedPlainText) -
    strlen(plainText));

    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, paddedPlainText,
    outputBuffer);
    mbedtls_aes_free(&aes);
}

void setup() {
    Serial.begin(115200);
}

void loop() {
    char myString[] = "16.1";
    char * key = "abcdefghijklmno";
    unsigned char cipherTextOutput[16];
    encrypt(myString, key, cipherTextOutput);
    char encryptedData[33]; // Para almacenar el resultado cifrado como cadena
    hexadecimal
    encryptedData[32] = '\0'; // Terminador de la cadena
    for (int i = 0; i < 16; i++) {
        sprintf(&encryptedData[i*2], "%02x", (int)cipherTextOutput[i]); //
        Almacenar cada byte cifrado como dos caracteres hexadecimales
    }
    Serial.println(encryptedData);
    delay(5000);
}
```

Este código realiza un cifrado de bloques de texto utilizando el estándar de cifrado avanzado (AES,) en el modo ECB (Electronic CodeBook). En el modo ECB, cada bloque

de texto se cifra de forma independiente con la misma clave. Esto significa que el mismo bloque de texto siempre se cifrará de la misma manera.

1) En la función **encrypt**:

- Se crea un contexto de cifrado AES (**mbdts_aes_context aes;**).
- Se inicializa el contexto de cifrado AES (**mbdts_aes_init(&aes);**).
- Se configura la clave de cifrado (**mbdts_aes_setkey_enc(&aes, (const unsigned char*) key, strlen(key) * 8);**).
- Se rellena el texto a cifrar hasta llegar a los 16 bytes necesarios para AES (**memcpy y memset**).
- Se realiza el cifrado en modo ECB (**mbdts_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, paddedPlainText, outputBuffer);**).
- Se libera el contexto AES (**mbdts_aes_free(&aes);**).
- En la función setup, se inicializa el puerto serie con una velocidad de 115200 bps (**Serial.begin(115200);**).

2) En la función **loop**:

- Se define la cadena de texto a cifrar (**char myString[] = "Texto descifrado";**).
- Se define la clave de cifrado de 16 caracteres (**char * key = "abcdefghijklmnop";**).
- Se define un buffer para almacenar el texto cifrado (**unsigned char cipherTextOutput[16];**).
- Se llama a la función “**encrypt**” para cifrar el texto (**encrypt(myString, key, cipherTextOutput);**).
- Se convierte el texto cifrado (que está en formato binario) a una cadena hexadecimal para que pueda ser impresa (**sprintf(&encryptedData[i*2], "%02x", (int)cipherTextOutput[i]);**).

3) Finalmente se imprime el texto cifrado en el monitor serie (**Serial.println(encryptedData);**). Esto es con el fin de que luego se pueda capturar el dato en Python.

5.2.4 Implementación de librería mbedtls Descifrado

Este script Python usa la biblioteca “*PyCryptoDome*” para descifrar el texto cifrado que se lee desde una conexión serial, y luego publica el texto descifrado a un broker MQTT.

```
import serial
from Crypto.Cipher import AES
import binascii
import paho.mqtt.client as mqtt

def decrypt(ciphertext, key):
    cipher = AES.new(key.encode(), AES.MODE_ECB)
    plaintext = cipher.decrypt(binascii.unhexlify(ciphertext))
    return plaintext.decode('utf-8').rstrip("\0")

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

def on_publish(client, userdata, mid):
    print("Message published")

key = 'abcdefghijklmnop'

# Inicializa la conexión serial
ser = serial.Serial('COM5', 115200)
# Inicializa el cliente MQTT
client = mqtt.Client()
client.on_connect = on_connect
client.on_publish = on_publish

# Establece usuario y contraseña
client.username_pw_set("****", "****")
# Conéctate al broker
client.connect("eqmx.iot-ia-group.com", 1883, 60)
# Inicia el loop de red del cliente MQTT
client.loop_start()

while True:
    if ser.in_waiting > 0:
        line = ser.readline().decode('utf-8').rstrip() # Lee una línea de
la conexión serial
        print('Línea recibida:', line)
        plaintext = decrypt(line, key)
        # Imprime el texto descriptado
        print('Texto descriptado:', plaintext)
        # Publica el texto descriptado en el broker MQTT
        client.publish("/v1/device1/temperature", plaintext)

# Para el loop de red del cliente MQTT antes de salir
```

```
client.loop_stop()
```

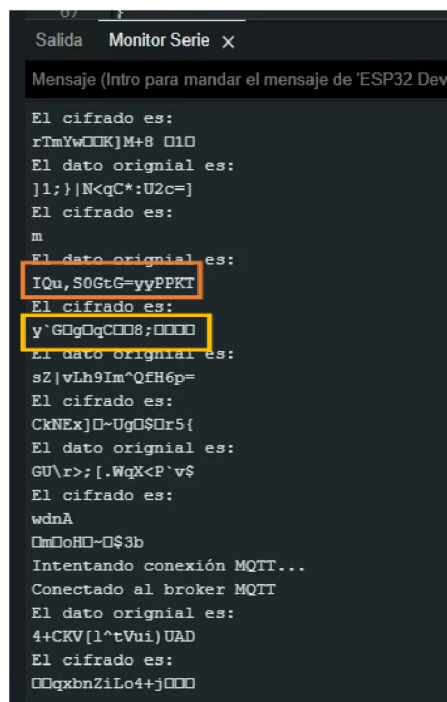
A continuación se detalla el funcionamiento del descifrado del mensaje enviado por el ESP32:

1. **ser = serial.Serial('COM5', 115200):** Esta línea establece la conexión con el ESP32 a través del puerto serial 'COM5'(En la cual se encuentra conectado el dispositivo) a una velocidad de 115200 baudios.
2. Dentro del bucle **while True**, se utiliza **if ser.in_waiting > 0:** para verificar si hay datos disponibles para ser leídos desde el puerto serial. Si hay datos esperando, se procede a leerlos.
3. **line = ser.readline().decode('utf-8').rstrip():** Esta línea lee una línea completa de datos del puerto serial (hasta el próximo carácter de nueva línea), decodifica los bytes en una cadena de texto usando el estándar de codificación de caracteres UTF-8, y luego elimina cualquier espacio en blanco o carácter de nueva línea del final de la cadena de texto.
4. **plaintext = decrypt(line, key):** Aquí es donde se realiza el descifrado del texto. Se llama a la función **decrypt**, pasando la línea de texto cifrado y la clave de cifrado. Esta función:
 - Crea un nuevo objeto de cifrado AES en modo ECB usando la clave de cifrado (**cipher = AES.new(key.encode(), AES.MODE_ECB)**).
 - Descifra el texto cifrado (**plaintext = cipher.decrypt(binascii.unhexlify(ciphertext))**). El texto cifrado se pasa a la función **binascii.unhexlify** para convertirlo de una cadena hexadecimal a bytes antes de ser descifrado.
 - La función devuelve el texto descifrado convertido a una cadena de texto UTF-8 y se eliminan los caracteres nulos ('\0') que puedan haber quedado como resultado del relleno durante el cifrado.
5. Finalmente, se imprime el texto descifrado y se publica al broker MQTT con **client.publish("Salida/01", plaintext)**.

Resultados

6.1 Resultados del algoritmo XOR

En la siguiente (Figura 6.1), se muestra el resultado de cómo el algoritmo XOR cifra el mensaje (recuadro naranja) y cuál es el mensaje original generado aleatoriamente (recuadro verde).



```

Salida Monitor Serie x
Mensaje (Intro para mandar el mensaje de 'ESP32 Dev

El cifrado es:
rTmYw00K]M+8 010
El dato original es:
]1;}|N<qC*:U2c=]
El cifrado es:
m
El dato original es:
IQa,S0GtG=yyPPKI
El cifrado es:
y`G0g0q0008;0000
El dato original es:
sZ|vLh9Im^QEH6p=
El cifrado es:
CkNEx]0~Ug0$0r5{
El dato original es:
GU\r>;[.WqX<P`v$
El cifrado es:
wdnA
0m0oH0-0$3b
Intentando conexión MQTT...
Conectado al broker MQTT
El dato original es:
4+CKV[l^tVui)UAD
El cifrado es:
00qxbnZiLo4+j000
  
```

Figura 6.1 - Mensajes cifrado y originales (Algoritmo XOR)

Fuente: Propia

A continuación se observa qué el broker MQTTX recibe el mensaje cifrado indicado en la (Figura 6.1), a través del tópico “Salida/01”.

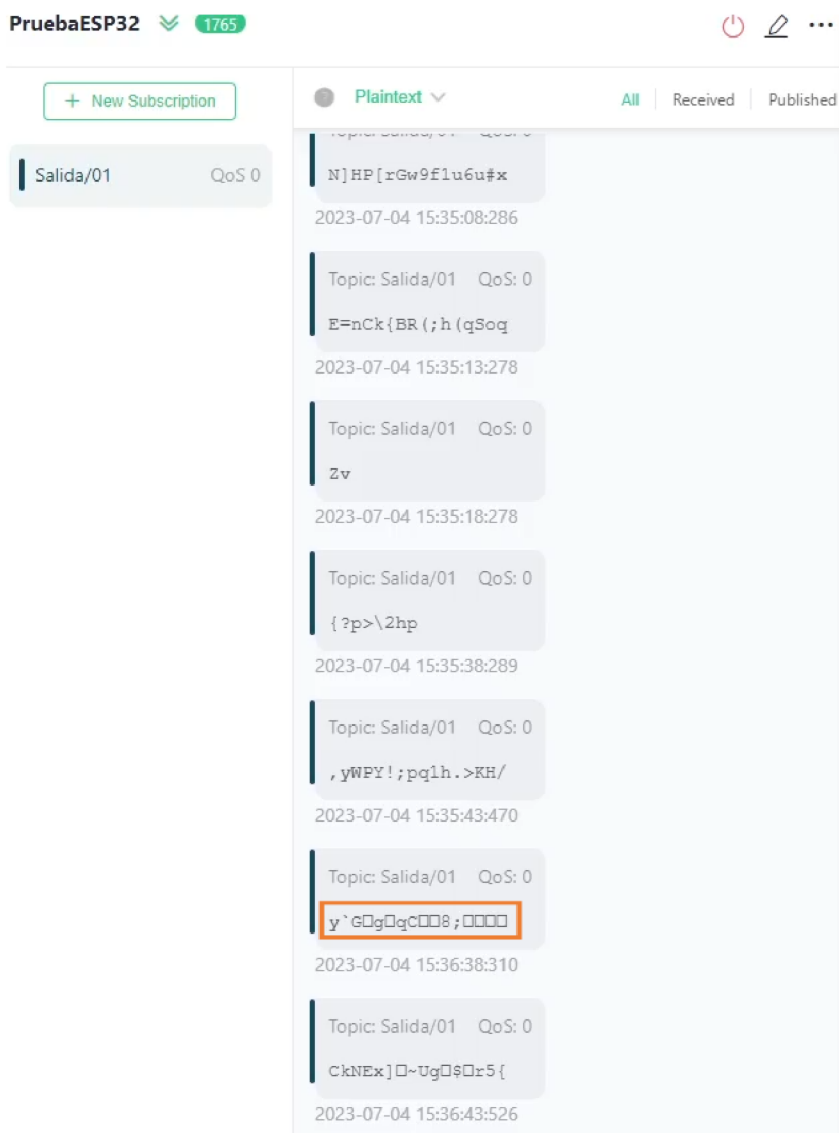


Figura 6.2 - Mensajes cifrado y originales (MQTTX)

Fuente: Propia

Luego de verificado que se reciben los mensajes, se procede a habilitar el script de descifrado, el cual está configurado en el broker MQTTX.

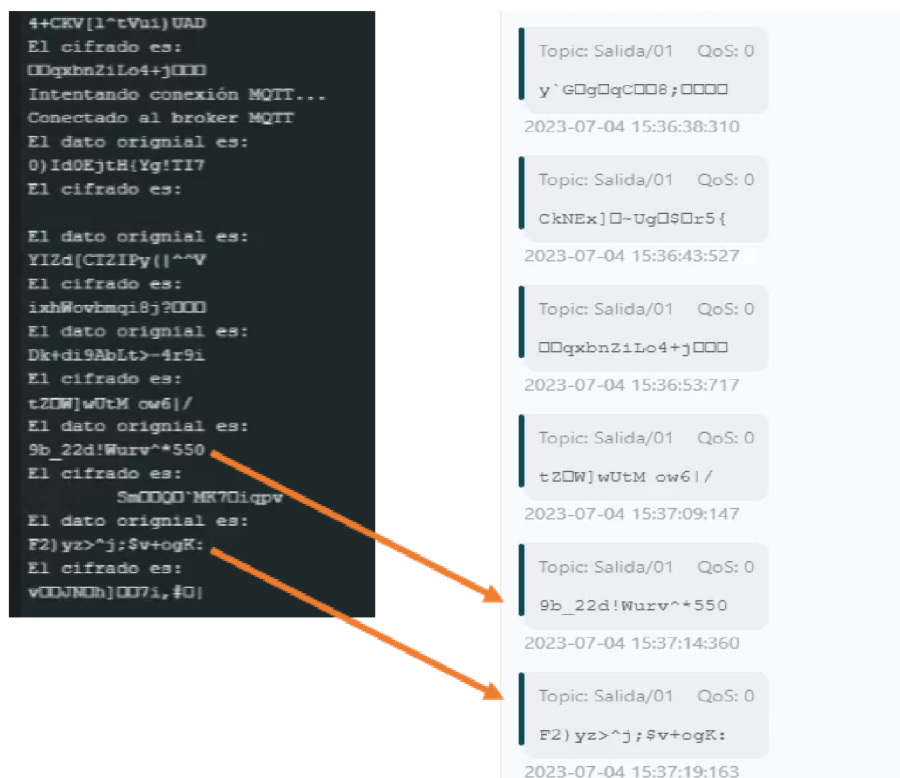


Figura 6.3 - Mensajes cifrado y originales (MQTTX) vs Monitor Serial

Fuente: Propia

Los mensajes recibidos por el broker con el script habilitado, concuerdan con los mensajes leídos por las consola serial de Arduino IDE.

El algoritmo solo cifra mensajes de largo 16 bits, ya que utiliza el operador módulo (%) para moverse a través de la clave de cifrado. Por ejemplo, **key[i % 16]** regresa al inicio de la clave después del carácter número 16. Esto funciona bien para los primeros 16 caracteres del texto plano, pero cualquier carácter adicional terminará siendo cifrado con los mismos caracteres de la clave que se usaron para los primeros 16 caracteres del texto.

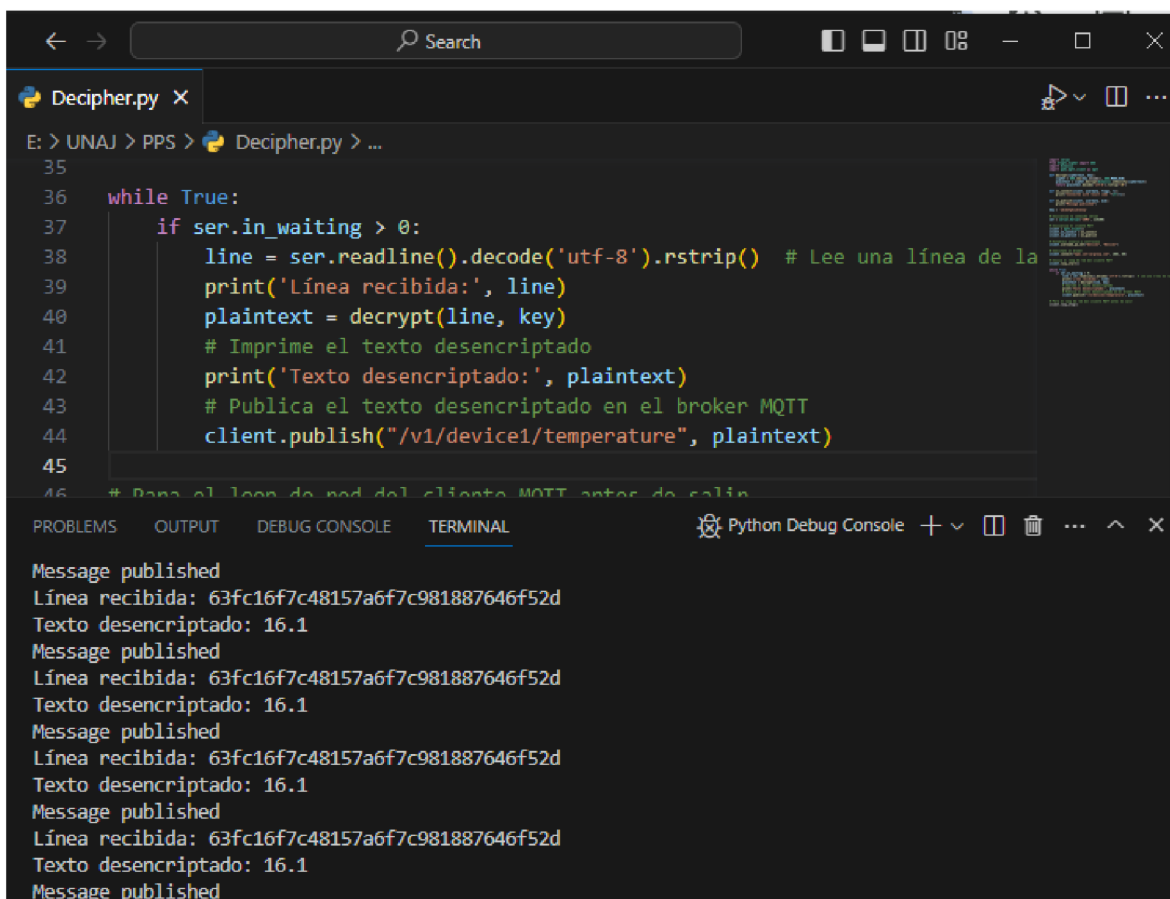
Por otro lado, si el mensaje a cifrar es mayor que 16 caracteres, hay un problema de desbordamiento de búfer en la línea **char encryptedData[16];** y en **encryptedData[i] = data[i] ^ key[i % 16];**. Aquí se está reservando solo espacio para 16 caracteres en el búfer

encryptedData. Si se intenta cifrar un texto de más de 16 caracteres, se escribirán datos más allá del final de este búfer, lo que puede provocar comportamiento impredecible.

La posible solución sería implementar el manejo de datos más largos, lo que podría implicar el cifrado en bloques de 16 caracteres a la vez o el redimensionamiento dinámico de los búferes para adaptarse a los datos más largos, en función del tamaño del texto a cifrar. Sin embargo, para este trabajo no es necesario un largo mayor al indicado, ya que solo se enviarán datos como temperatura del suelo, humedad, etc, los cuales siempre estarán en este rango, ya que por ejemplo la cadena “**Temperatura:0**” y “**Temperatura:100**” que una variación exagerada de la misma, como máximo tendrán un largo de 15 caracteres, para los cuales el algoritmo no presenta fallas.

6.2 Resultados del algoritmo AES

Como primer paso se simuló un valor de temperatura de 16.1° como el mensaje cifrado enviado por el ESP32 al broker de Emqx. Este mismo es captado por el script de python el cual se encarga de mostrar cuál es su valor de cifrado, descifrarlo y publicarlo en el tópico “/v1/device1/temperature”



```

Decipher.py X
E: > UNAJ > PPS > Decipher.py > ...
35
36 while True:
37     if ser.in_waiting > 0:
38         line = ser.readline().decode('utf-8').rstrip() # Lee una línea de la
39         print('Línea recibida:', line)
40         plaintext = decrypt(line, key)
41         # Imprime el texto descriptado
42         print('Texto descriptado:', plaintext)
43         # Publica el texto descriptado en el broker MQTT
44         client.publish("/v1/device1/temperature", plaintext)
45
46 # Para el loop de red del cliente MQTT antes de salir

```

PROBLEMS **OUTPUT** **DEBUG CONSOLE** **TERMINAL** Python Debug Console

```

Message published
Línea recibida: 63fc16f7c48157a6f7c981887646f52d
Texto descriptado: 16.1
Message published
Línea recibida: 63fc16f7c48157a6f7c981887646f52d
Texto descriptado: 16.1
Message published
Línea recibida: 63fc16f7c48157a6f7c981887646f52d
Texto descriptado: 16.1
Message published
Línea recibida: 63fc16f7c48157a6f7c981887646f52d
Texto descriptado: 16.1
Message published
Línea recibida: 63fc16f7c48157a6f7c981887646f52d
Texto descriptado: 16.1
Message published

```

Figura 6.4 - Mensajes cifrado y originales (Python)

Fuente: Propia

Como podemos observar el mensaje enviado por el dispositivo es leído de manera correcta, mostrando su valor de encriptado en el campo “**Línea recibida**”, su valor original en “**Texto descriptado**” y finalmente publicando el mensaje descriptado en el tópico. Seguido de esto pasamos a la verificación del mensaje recibido en el broker, conectándonos al mismo de forma similar a como explica la sección (5.1.1 Emqx inicio Pag 85.). A continuación procedemos a suscribirnos al tópico donde se van a enviar los mensajes.

Subscribe

Topic

QoS

0

Subscribe

Figura 6.5 - Suscripción tópico (Emqx)

Fuente: Propia

Una vez que estamos suscritos, observamos que el mensaje esperado se recibe correctamente.

Messages received 			
Messages	Topic	QoS	Time
16.1	/v1/device1/temperature	0	2023-08-04 01:16:40
16.1	/v1/device1/temperature	0	2023-08-04 01:16:35
16.1	/v1/device1/temperature	0	2023-08-04 01:16:30
16.1	/v1/device1/temperature	0	2023-08-04 01:16:25
16.1	/v1/device1/temperature	0	2023-08-04 01:16:20

Figura 6.6 - Datos recibidos (Emqx)

Fuente: Propia

Finalmente vamos a observar si el mensaje recibido en el tópico es procesado por NodeRed, el cual va a almacenarlo en la BD, para verificar esto simplemente se observa el

dashboard dedicado de Grafana, ya que el mismo está conectado para poder leer los datos guardados en MySQL.

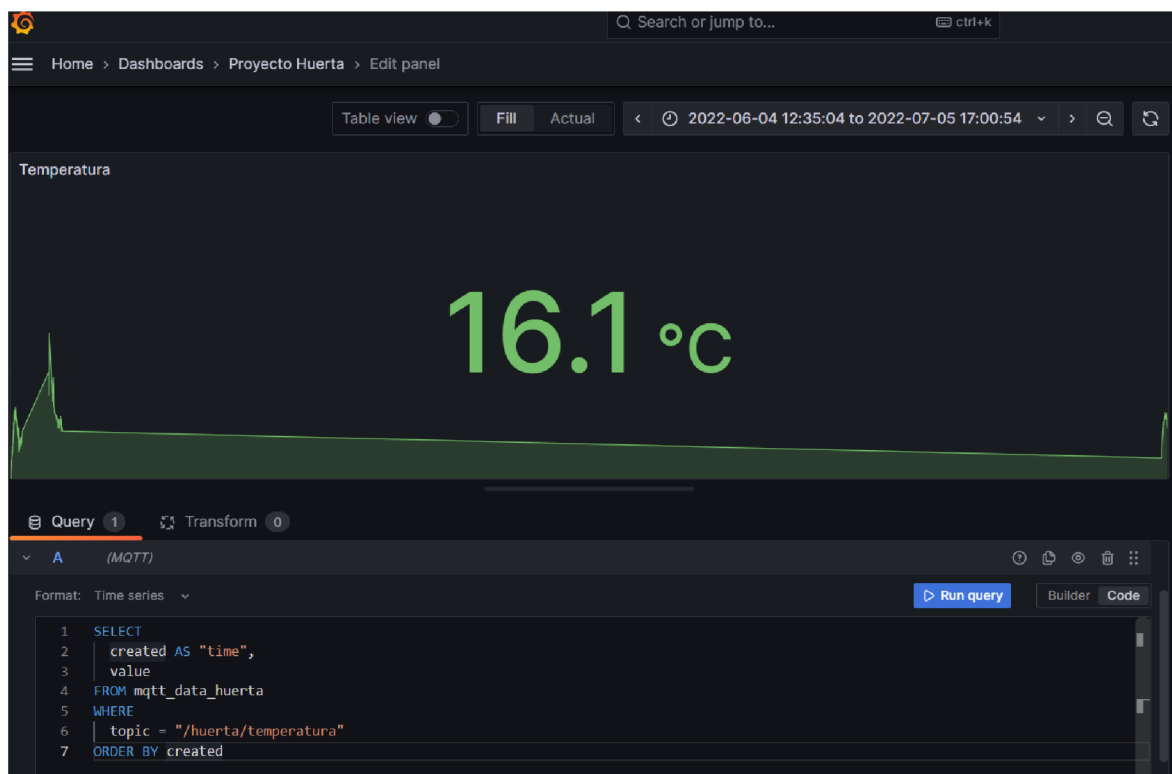


Figura 6.7 - Datos recibidos (Grafana)

Fuente: Propia

El dato se lee de manera correcta en el dashboard de grafana, en la parte inferior de la imagen se encuentra la query, la cual está encargada de leer el dato de la tabla “**mqtt_data_huerta**”, alojada en la BD “**mqtt**”.

Con esto concluimos que el ciclo del mensaje que parte cifrado del ESP32, viaja de esta manera hasta el servidor, donde se descifra con el script de Python, es procesado por NodeRed, se almacena y finalmente se lee en el Dashboard de Grafana.

Como punto importante cabe destacar que este algoritmo también cifra máximo una cadena de 16 bits para el largo de un mensaje. Ocurrirá el siguiente error, si el mensaje se extiende de este parámetro.

Figura 6.8 - Datos recibidos (Grafana)

En la sección de Análisis de resultados, veremos porque se da este problema y cual seria una posible solución al mismo.

En base a lo observado se puede analizar la situación para intentar descubrir las razones del inconveniente encontrado y los resultados obtenidos hasta el momento.

Como primer punto el algoritmo cifra correctamente el 100% de los mensajes enviados inclusive si de caracteres especiales se tratase, el cifrado se produce de manera correcta y esperada, convirtiendo el mensaje en un número hexadecimal ilegible.

Otro punto positivo es que el servidor logra soportar una gran carga de mensajes, ya que se realizó un prueba donde se envían mensajes simultáneos, en intervalos de tiempo de 1 segundo, y el mismo logró recibir los mismos de manera correcta.

Como uno de los inconvenientes encontrados está el limitante de que el cifrado es para una cadena de como máximo 16 bits, en principio esto no es un problema ya que para la aplicación desarrollada en este trabajo, la cantidad de caracteres es suficiente y permite dar una flexibilidad al envío de los datos, de tipo humedad, temperatura, etc.

Finalmente Otro de los inconvenientes es el cifrado por ECB, que aunque es un cifrado en bloques simple y eficiente, presenta las siguientes desventajas:

1. **Patrones en el texto plano se mantienen en el texto cifrado:** Dado que cada bloque se cifra de forma independiente, si dos bloques del texto plano son idénticos,

sus correspondientes bloques de texto cifrado también serán idénticos. Esto significa que los patrones en el texto plano se mantienen en el texto cifrado, lo que puede proporcionar a un atacante información sobre el contenido del mensaje.

2. **No tiene seguridad en la integridad de los datos:** Los modos de operación de AES como el ECB no proporcionan ninguna seguridad en la integridad de los datos. Eso significa que si alguien altera el texto cifrado, no se puede detectar.

En la siguiente sección se plantean algunas soluciones a futuro para los inconvenientes detallados.

6.2.2 Análisis de soluciones y mejoras para AES

Extender la cadena cifrada a más de 16 bits

Para poder cifrar más bits de una cadena, podría plantearse un enfoque general el cual divide la cadena en bloques de 16 bytes y luego cifrar cada bloque por separado. Esta es la forma en que se maneja en el cifrado por bloques. Se implementó un ejemplo el cual podría funcionar siguiendo esta idea:

```
void encrypt(const char * plainText, size_t plainTextLength, char * key,
unsigned char * outputBuffer){
    mbedtls_aes_context aes;
    mbedtls_aes_init(&aes);
    mbedtls_aes_setkey_enc(&aes, (const unsigned char*) key, strlen(key) * 8);

    size_t blockCount = (plainTextLength + 15) / 16; // Calcula cuántos
    bloques necesitamos, redondeando hacia arriba

    for (size_t i = 0; i < blockCount; i++) {
        unsigned char paddedPlainText[16];

        size_t start = i * 16;
        size_t end = start + 16 <= plainTextLength ? start + 16 :
plainTextLength;
        size_t blockLength = end - start;

        memcpy(paddedPlainText, plainText + start, blockLength);
        memset(paddedPlainText + blockLength, 0, sizeof(paddedPlainText) -
blockLength);

        mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, paddedPlainText,
outputBuffer + i * 16);
    }

    mbedtls_aes_free(&aes);
}
```

Esto dividirá la cadena de entrada en bloques de 16 bytes y cifrará cada bloque por separado.

Hay que tener en cuenta que se asume que **outputBuffer** es lo suficientemente grande como para contener todos los datos cifrados. También, es importante destacar que este método simple de relleno (*padding*) con ceros, con lo cual puede no ser adecuado para todos los usos, especialmente para datos que pueden contener ceros. Para una solución más íntegra podría considerarse el uso de un esquema de relleno más robusto como PKCS7, el cual es un estándar que describe cómo los datos deben ser ajustados (rellenados) para que se ajusten al tamaño del bloque requerido por un algoritmo de cifrado específico. La ventaja de este esquema de relleno es que cuando se recibe un mensaje, se puede observar el último byte para saber cuántos bytes de relleno se han añadido y, por tanto, cuántos se necesitan eliminar para obtener el mensaje original.

Mejora de modo ECB

Para mejorar la seguridad y evitar los problemas mencionados anteriormente, se pueden utilizar otros modos de operación de AES, como por ejemplo:

- **Modo CBC** (Cipher Block Chaining): En este modo, cada bloque de texto plano se combina con el bloque de texto cifrado anterior antes de ser cifrado. Esto significa que cada bloque de texto cifrado depende de todos los bloques de texto plano procesados hasta ese punto, lo que evita el problema de los patrones en el texto cifrado. Sin embargo, al igual que el ECB, no proporciona seguridad en la integridad de los datos.
- **Modo GCM** (Galois/Counter Mode): Este es un modo de operación más moderno que proporciona tanto la confidencialidad de los datos como la integridad de los datos. Cada bloque de texto plano se cifra de forma independiente (como en el ECB), pero también se utiliza un valor adicional llamado un "nonce" para garantizar que cada cifrado sea único. Además, GCM incluye una función de autenticación que permite verificar la integridad de los datos.

Conclusiones

La seguridad informática siempre ha atraído mi interés. A lo largo de la historia, la necesidad de encriptar y desencriptar mensajes ha desencadenado una revolución, que dio lugar a la primera computadora. Cuando se presentó la oportunidad de desarrollar este proyecto, no dudé en aceptarlo, ya que me permitía aplicar mis conocimientos en ciberseguridad, mi campo de trabajo actual, y mis conocimientos de IoT, que adquirí gracias a varias tareas en colaboración con el grupo de trabajo del Laboratorio de IoT y Sistemas Embebidos (IoT&SE) del programa TICCAPPS de la UNAJ.

A medida que avanzaba en el proyecto, fui comprendiendo diferentes técnicas de cifrado y protocolos. Estos son utilizados actualmente tanto por empresas privadas como por organismos certificados en seguridad informática. Una vez que comprendí estas técnicas y protocolos, fui capaz de aplicarlas en este proyecto para alcanzar los objetivos propuestos. Este proceso fue apasionante y emocionante.

En relación a este trabajo de Práctica Profesional Supervisada, el servidor proporcionado por la UNAJ ha sido extremadamente útil. En él se alojaron varias aplicaciones utilizadas para las pruebas de este proyecto, lo que facilitó el uso personal de recursos.

En términos de resultados, es importante destacar el papel crucial que juega la ciberseguridad en los sistemas informáticos actuales. Los ciberataques a empresas son cada vez más frecuentes, causando pérdidas millonarias. Por esta razón, en este trabajo no solo logramos nuestros objetivos, sino que también establecimos las bases para las buenas prácticas en el diseño de sistemas IoT seguros. Esto sugiere que este enfoque debe ser aplicado en todas las aplicaciones de Internet de las cosas. Además, se ha iniciado una nueva línea de investigación que permitirá explorar otros aspectos de la ciberseguridad en el futuro.

Finalmente, considero que esta etapa de formación es una excelente oportunidad para concluir mi carrera de grado. Me permite demostrar todos los conocimientos y habilidades que he adquirido a lo largo de mi trayectoria educativa.

Referencias bibliográficas

1. Teoria.com. (n.d.). Encrypt. Recuperado el 22 de Febrero de 2023, de <https://www.teoria.com/jra/aes/encrypt.html>
2. Ramió Aguirre, J. (2022). Módulo 8: Criptografía simétrica en bloque. Recuperado el 24 de Marzo de 2023, de <https://www.criptored.es/libro/cap%C3%ADtulo%208%20Criptograf%C3%ADa%20sim%C3%A9trica%20en%20bloque.pdf>
3. Seeed-Studio. (n.d.). HelloHMACSHA256.ino. Recuperado el 10 de Junio de 2023, de https://github.com/Seeed-Studio/Seeed_Arduino_mbedtls/blob/master/examples/HelloHMACSHA256/HelloHMACSHA256.ino
4. EMQ X. (n.d.). EMQ X - An open-source MQTT broker for IoT. Recuperado el 16 de marzo de 2023, de <https://www.emqx.io/docs/en/v5.0/#feature-list>
5. Grafana. (n.d.). Grafana documentation. Recuperado el 05 de Febrero de 2023, de <https://grafana.com/docs/grafana/latest/>
6. Node-RED. (n.d.). Node-RED: Documentation. Recuperado el 22 de Mayo de 2023, de <https://nodered.org/docs/>
7. MariaDB. (n.d.). Documentation - MariaDB Knowledge Base. Recuperado el 12 de Marzo de 2023, de <https://mariadb.com/kb/en/documentation/>
8. Espressif Systems. (n.d.). Getting Started with ESP32-DevKitC Guide. Recuperado el 24 de marzo de 2023, de <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>

9. NGINX. (n.d.). NGINX Docs. Recuperado el 12 de Enero de 2023, de <http://nginx.org/en/docs/>
10. MariaDB. (n.d.). Documentation - MariaDB Knowledge Base. Recuperado el 19 de Abril de 2023, de <https://mariadb.com/kb/en/documentation/>
11. Katz, J., & Lindell, Y. (2014). Introduction to Modern Cryptography (2nd ed.). CRC Press.
12. Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Wiley.
13. Sweigart, A. (2020). Beyond the Basic Stuff with Python: Best Practices for Writing Clean Code. No Starch Press.
14. López-Cerón Corredor, M. (2023). Sistemas de Producción IoT Automáticos y Seguros. (Tesis de [nivel del grado]). [Universidad Politécnica de Madrid], Madrid.
15. Greengard, S. (2020). The Internet of Things (The MIT Press Essential Knowledge series). MIT Press.
16. Mattern, F., & Floerkemeier, C. (2020). From the Internet of Computers to the Internet of Things. Springer.