Grado en Ingeniería Informática

Trabajo de Final de Grado

Securización protocolo MQTT

Autor:
Anas Alsaqqa Duwairi

Supervisor:
Kamel Arrafet Cruz
Tutor académico:
Jorge Vicente Castellet Marques

Fecha de lectura: 18 de Junio de 2024
Curso académico 2023/2024

Resumen

El proyecto se centra en fortalecer la seguridad del protocolo MQTT entre dispositivos IoT y un servidor, considerando soluciones como el uso de VPN y la implementación de TLS/SSL. Estas medidas se implementarán debido a que el servidor MQTT ya no está dentro de la red de la empresa, sino fuera de ella. Cada opción será evaluada para determinar la combinación óptima de seguridad y rendimiento para la comunicación externa del protocolo MQTT. El objetivo principal es identificar la solución más efectiva y eficiente para asegurar esta comunicación externa.

Palabras clave

Seguridad, MQTT, TLS/SSL, VPN, IoT.

Keywords

Security, MQTT, TLS/SSL, VPN, IoT.

Agradecimientos

Quiero empezar dando las gracias a mi familia, especialmente a mis padres, por estar siempre ahí para mí durante toda mi carrera académica. Les agradezco de corazón su paciencia, sacrificio y motivación, sin los cuales este logro no habría sido posible.

También quiero expresar mi profundo agradecimiento a mi tutor, Jorge Vicente Castellet Marques. Su guía y consejos han sido invaluables durante todo el proceso de investigación. Aprecio enormemente su tiempo y dedicación para ayudarme a dar lo mejor de mí mismo.

Finalmente, quiero agradecer a mi supervisor, Kamel Arrafet Cruz. Su liderazgo y dirección fueron fundamentales para llevar a cabo este proyecto de manera exitosa. Estoy realmente agradecido por su apoyo y orientación a lo largo de este camino.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este documento presenta el **TFG** (Trabajo fin de grado) en Ingeniería Informática del Alumno **Anas Alsaqqa Duwairi**, cursando el grado en Ingeniería Informática en la UJI (Universidad Jaume I), situada en Castellón, España.

1.1. Contexto y Motivación del Proyecto

El proyecto se desarrolla en el entorno de una empresa dedicada a soluciones para los sectores de la industria. La empresa ha decidido expandir sus operaciones hacia una infraestructura más globalizada, lo que implica alojar el servidor MQTT fuera de sus instalaciones locales.

La **motivación principal** surge de **la necesidad de securizar** la comunicación MQTT entre los dispositivos Raspberry Pi 4 y el servidor MQTT remoto, ahora ubicado fuera de la red corporativa. Esta transición nos presenta desafíos adicionales en términos de seguridad de la comunicación. Con la información ahora viajando a través de Internet, es crucial estar atentos a **posibles amenazas** que podrían comprometer la integridad de los datos.

El objetivo principal es investigar y evaluar diferentes métodos para **securizar la comunicación MQTT** en este nuevo contexto. Se considerarán opciones como el uso de MQTT sobre TLS (MQTTS) y la implementación de una conexión VPN. La elección de la solución adecuada se basará en la mejor combinación de **seguridad y eficiencia** para garantizar una comunicación segura y confiable en entornos de IoT.

Este proyecto busca contribuir al conocimiento en el campo de la seguridad en redes de dispositivos conectados, proporcionando una guía práctica para asegurar la comunicación MQTT en entornos empresariales distribuidos. Al abordar este desafío, se espera mejorar la seguridad y la integridad de los datos transmitidos, garantizando el cumplimiento de los estándares de seguridad de la empresa.

El proyecto se desarrolla en la solución de ImesPyme, que más adelante explicaremos con más detalle.

1.1.1. Entorno Profesional y Departamento de Desarrollo en la Empresa

IT.Backing, el entorno profesional donde se desarrolla el proyecto de la securización del protocolo MQTT. Es una empresa situada en la ciudad Castellón de la Plana, España. El objetivo principal de esta es "ayudar a optimizar los del negocio para que se más rentable y competitivo" [6]. Su logo se muestra en la Figura 1.1.

Figura 1.1: Logo de IT.Backing

Tiene **4 ramas** principales de soluciones que son: ImesPyme (Digitalización de procesos productivos) una de las soluciones más modernas en el entorno Industria 4.0, Tiles 360 (ERP para el sector cerámico), Microsoft Dynamics 365 Business Central (ERP Dynamics 365 Business Central), WII Locations (Software de gestión de almacenes (SGA)).

Nosotros nos centraremos en la solución de ImesPyme, que es donde se esta llevando a cabo el proyecto. Se muestra su logo en la figura 1.2.

El nombre ImesPyme se compone de 3 partes:

1 - I, que viene de la primera letra de IT.Backing.

2 - Mes (Manufacturing Execution System), un software que actúa como sistema de control y monitoreo que se encarga de gestionar la información de procesos tanto en producción como en mantenimiento y parados.

3 - PYME "Empresa mercantil, industrial, etcétera, compuesta por un número reducido de trabajadores, y con un moderado volumen de facturación" [4].

Figura 1.2: Logo de ImesPyme

Dentro de esta solución nos encontramos con 7 modulos, 2 de ellos fundamentales y los 5 modulos adicionales.

Los 2 modulos fundamentales son: InfoPlanta y ProdControl, y los módulos adicionales 5: Notify, Quality, GMAO, LinePrinter, BI.

1.1.2. Modelo de negocio

IT.Backing es una empresa dirigida a soluciones IT para otras empresas, es decir, un modelo de negocio B2B (Business to Business). Este modelo de negocio consiste que las empresas vendan productos o servicios a otras empresas en lugar de vender al consumidor final [1].

1.1.3. Seguridad en entornos industriales

La **seguridad en entornos industriales** a lo largo de los años hemos visto que tiene un papel fundamental, ya sea por la gran información que se esta transmitiendo o por el impacto en la producción. Además esta también tiene una estrecha relación con la seguridad de los trabajadores, ya que si algún intruso cambio los parámetros de alguna maquinaria pesada, como puede ser un robot industrial o linea de ensamblaje, esta puede cambiar su comportamiento poniendo en peligro la integridad física de los trabajadores y la del entorno, como por ejemplo en el ataque a la central nuclear de Irán que más adelante explicaremos.

Vamos a hablar un poco del caso del ataque del **gusano Stuxnet** que atacó a la central nuclear de Natanz, Irán en 2010.

El gusano llamado Stuxnet tomo el control total de alrededor de **1000 maquinas**. Stuxnet fuera directo a los PLC (Programmable Logic Controller) que controlaban la velocidad de las centrifugadores. El gusano aprovechó varias debilidades, algunas de ellas fueron agujeros de seguridad en el sistema operativo Windows y otras el usó de impresoras desactualizadas para llegar a una capa mas profunda.

Lo que hizo Stuxnet fue **cambiar los parámetros de velocidad de las centrifugadores**, que estas se usaban para aislar el uranio enriquecido, para que funcionasen a altas velocidades durante un tiempo lo bastante prolongado para causar daños. Otro ataque fue desacelerar las centrifugadoras. Estos ataques se repitieron durante varios meses.

Comentamos antes que tenia un **control total** de la centrifugadora ya que el gusano tenia código malicioso que hasta tal punto que no dejaba pulsar el interruptor de emergencia para parar estas maquinas.

Para ver la magnitud del problema, **alrededor de un 20 % de las centrifugadoras** quedaron **inutilizadas**, es decir, fuera de servicio y tuvieron que ser retiradas.

Podemos encontrar más información sobre el incidente en la pagina web de BBC [2].

1.2. Objetivo y alcance del proyecto

El objetivo principal de este proyecto es **fortalecer la seguridad del protocolo MQTT** en un **entorno empresarial distribuido**. Esto implica asegurar la comunicación entre dispositivos IoT y un servidor MQTT remoto, considerando la transmisión de datos a través de

Internet.

1.2.1. Objetivos específicos

Los objetivos específicos del proyecto son:

- **Investigar y evaluar** diferentes métodos de securización para el protocolo MQTT en entornos empresariales distribuidos.
- **Implementar** una solución de seguridad para la comunicación MQTT entre dispositivos IoT y un servidor MQTT remoto.
- Realizar **pruebas exhaustivas** para garantizar la eficacia y fiabilidad de la solución implementada.
- **Comparar y analizar** el rendimiento de las diferentes opciones de securización para determinar la más adecuada en términos de seguridad y eficiencia.

1.2.2. Alcance del proyecto

El alcance del proyecto abarca los siguientes aspectos:

- Se implementará una **solución de securización** para el protocolo MQTT, considerando opciones como MQTT sobre TLS (MQTTS) y el uso de VPN.
- La solución se aplicará específicamente a la comunicación entre **dispositivos Raspberry Pi 4 y un servidor MQTT remoto** ubicado fuera de la red corporativa.
- Se realizarán **pruebas de rendimiento y seguridad** para evaluar la efectividad de la solución implementada.
- Se elaborará un **informe detallado** que incluya la metodología utilizada, los resultados obtenidos y **las conclusiones del proyecto**.

1.3. Descripción detallada del proyecto

El protocolo **MQTT** (**Message Queuing Telemetry Transport**) es un protocolo ligero de mensajería por suscripción/publicación ampliamente utilizado en la comunicación entre dispositivos IoT (**Internet of Things**) y servidores. Su simplicidad, eficiencia y bajo consumo de ancho de banda lo convierten en una opción ideal para aplicaciones de IoT que requieren una comunicación constante y confiable.

Sin embargo, MQTT no es **intrínsecamente seguro**, lo que significa que es vulnerable a una variedad de ataques cibernéticos. Estos ataques pueden incluir interceptación de datos,

suplantación de identidad y alteración de mensajes. Para proteger la comunicación MQTT en entornos empresariales distribuidos, es **crucial** implementar medidas de seguridad adicionales.

Este proyecto **se centra en la investigación, evaluación e implementación de diferentes métodos** para securizar la comunicación MQTT entre dispositivos IoT y un servidor MQTT remoto. Se considerarán opciones como **MQTT sobre TLS (MQTTS)** y el uso de **VPN (Virtual Private Network)** para identificar la mejor combinación de seguridad y eficiencia.

1.4. Estructura de la memoria

Esta memoria se divide en varios capítulos:

- **Capitulo 2: Planificación del proyecto**, en este capítulo se hablará sobre la metodología empleada para llevar a cabo este proyecto. También podemos encontrarnos con el desglose de las fases y el EDT.
- **Capitulo 3: Análisis del sistema**, en este capítulo se hablará sobre los objetivos del sistema y las especificaciones necesarias para cumplir con esos objetivos. Se investigara las diferentes formas de securizar el protocolo MQTT, como funcionan las credenciales, las ACLs, etc.
- **Capitulo 4: Diseño**, en este capítulo se estudiara a fondo el diseño de la implementación del proyecto, el cambio debido a la evolución de la infraestructura.
- **Capitulo 5: Implementación y pruebas**, en este capítulo se pondrán a pruebas las diferentes soluciones planteadas anteriormente, se medirá el rendimiento de las VPNs, se asegurara de que los soluciones planteadas ofrezcan una robusta seguridad.
- **Capitulo 6: Mantenimiento**, en este capítulo abordaremos las prácticas esenciales para mantener la infraestructura de tecnología de la información en óptimas condiciones. Desde la gestión de certificados de seguridad hasta la supervisión activa de las conexiones VPN y la administración de las listas de control de acceso (ACL).
- **Capitulo 7: Conclusiones**, en este ultimo capítulo, se dará una conclusión del proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Para este proyecto de **securización del protocolo MQTT**, se ha seleccionado la **metodología predictiva o en cascada**. La elección de esta metodología se justifica por su capacidad para abordar de manera estructurada y secuencial los requisitos específicos del proyecto.

En la metodología en cascada, cada etapa del proyecto se divide en **fases secuenciales**, donde el progreso fluye de manera unidireccional, similar al flujo de una cascada. Esto significa que cada fase debe completarse antes de pasar a la siguiente, y una vez completada, no se puede volver atrás. En la siguiente figura 2.1 se mostrara las fases de la **metodología predictiva**.

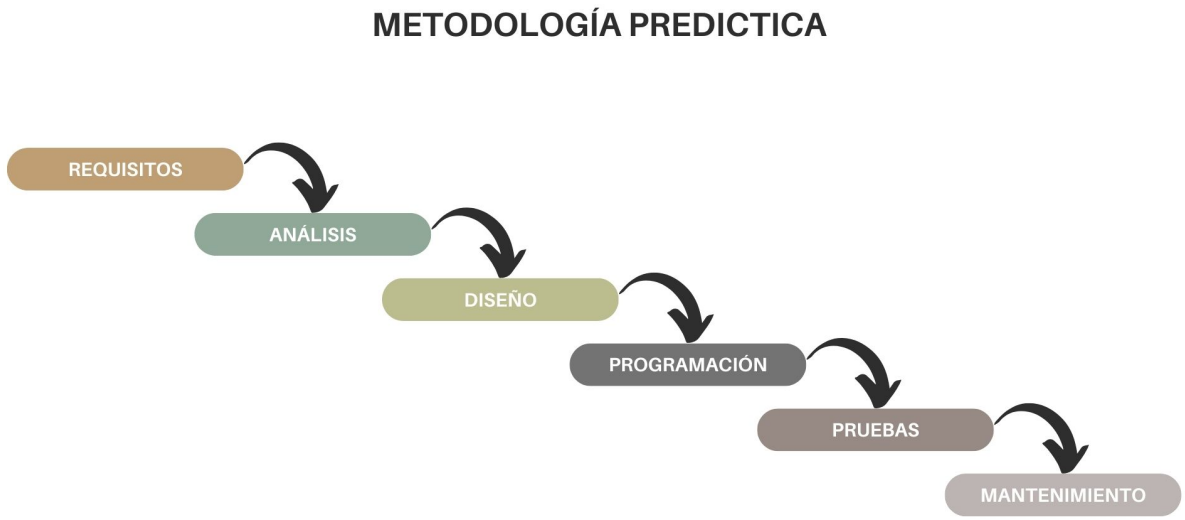


Figura 2.1: Fases metodología predictiva

A continuación, se explicarán las fases de la metodología predictiva.

Inicialmente, se realizará una **fase de los requisitos** de seguridad del sistema MQTT, identificando las vulnerabilidades potenciales y los requisitos de seguridad necesarios.

Durante esta fase, se llevará a cabo un análisis exhaustivo de los requisitos identificados en la fase anterior. Se examinarán en detalle **las necesidades de seguridad del sistema**, considerando tanto las vulnerabilidades potenciales como los requisitos específicos de seguridad.

Luego, se procederá con la **fase de diseño**, donde se planificarán y diseñarán las soluciones de seguridad, como la implementación de VPN, TLS/SSL, ACL, entre otros.

Después de la fase de diseño, se llevará a cabo **la implementación de las medidas de seguridad planificadas**, siguiendo estrictamente el diseño establecido. Como en cada implementación hay que asegurarse del funcionamiento de esta, **las pruebas se realizaran en conjunto**. Es decir, cuando se configure las credenciales, se realizarán las pruebas de esta. Cuando se configure las ACLs, se realizarán las pruebas de esta. Así procederemos con todas las implementaciones.

Posteriormente, **procederemos la implementación en el código actual**, que esta usando la empresa para las comunicaciones mediante el protocolo MQTT, la solución elegida para securizar dicha conexión. Después de la implementación, se procederá a hacer pruebas de dicha implementación.

Finalmente, se establecerá **un plan de mantenimiento** para abordar posibles necesidades futuras, como la renovación de certificados o la resolución de caídas en la VPN (según la solución que se haya escogido) y la actualización de los temas (topics) para ajustar los ACLs (Access Control Lists).

Es importante tener en cuenta que, una vez que se complete una fase y se avance a la siguiente, no se puede retroceder. Esto resalta la importancia de una planificación cuidadosa y una ejecución precisa en cada etapa del proyecto.

2.2. Planificación

En esta sección explicaremos las horas totales del proyecto, el desglose de las horas necesitadas en cada fase, una estructura de desglose de trabajo (EDT) y su respectivo diccionario, teniendo este el objetivo, responsable de la tarea, fecha de inicio y fin, descripción de la tarea.

2.2.1. Horario de trabajo

En el contrato de estancia de practicas se acordó **300 horas**, en las cuales se dividieron en 24 horas cada semana. De lunes a jueves se realizaron 5 horas, excepto los viernes que fueron 4 horas. Mi estancia **empezó el 6 de febrero hasta el 30 de abril**. En total son 12 semanas de estancia. En las tres ultimas hice 2 horas más los miércoles y jueves, ya que si multiplicamos

12 semanas x 24 horas a la semana, nos da un resultado de 288. Entonces suplimos esas 12 horas restantes haciendo esas 4 horas más a la semana, en las ultimas 3 semanas, que nos da un resultado de 300 horas totales.

2.2.2. Desglose de las fases

1. **Requisitos:**
    a) Reunión con supervisor
    b) Mecanismos de seguridad
2. **Análisis**
    a) Investigar protocolo MQTT
    b) Investigar Bridge
    c) Investigar autenticación basada en credenciales
    d) Investigar ACLs
    e) Investigar VPNs
    f) Investigar servidor MQTT mosquitto
3. **Diseño**
    a) Diseño de la implementación actual
    b) Diseño de la implementación futura
    c) Diseño de la implementación del proyecto
    d) Diseño de la implementación TLS/SSL
    e) Diseño de la implementación VPN
    f) Requisitos de diseño del sistema
4. **Programación**
    a) Pasos previos a la implementación
    b) Implementación mínima MQTT
    c) Implementación credenciales MQTT
    d) Implementación ACLs MQTT
    e) Implementación certificados TLS/SSL
    f) Implementación VPNs
5. **Pruebas**
    a) Pruebas desde el exterior del entorno de la empresa.
6. **Mantenimiento**
    a) Gestión de certificados
    b) Verificación VPN
    c) Ajustes ACL

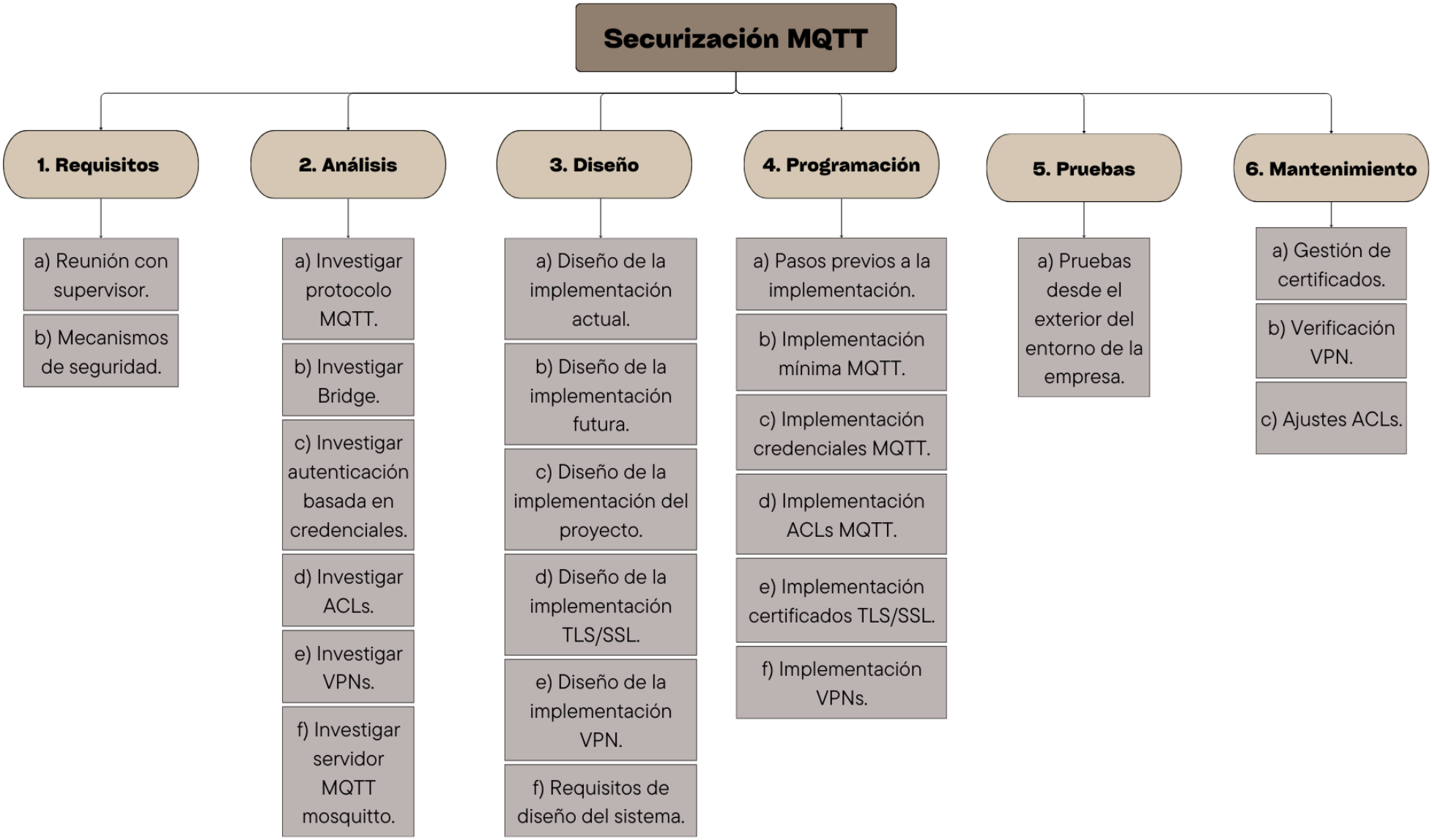


Figura 2.2: Esquema EDT de las tareas de las fases

2.2.3. Esquema EDT

En los apartados de **la planificación del proyecto**, también esta la **EDT (estructura de desglose de trabajo)**, que muestra de una manera esquematizada las tareas de las fases que se deben de realizar. En la figura 2.2 se muestra este esquema.

2.2.4. Diccionario de la EDT

El diccionario EDT trata de explicar cada tarea. Este es muy importante tanto para comenzar con tu nuevo proyecto para guiarte a encontrar con facilidad los requisitos de la tareas a realizar. En la figura 2.2 se muestra las fases para hacer este diccionario. A continuación procederemos a explicar el diccionario EDT desde la tabla 2.1 hasta la tabla 2.6, procederemos a poner una tabla por fase. El mantenimiento se ha incluido pero en este trabajo no se contemplara.

Código fase:	1
Nombre de tarea:	Requisitos.
Objetivo:	Definir los requisitos para la securización del protocolo MQTT.
Descripción:	Se realizarán todas las tareas de la fase de requisitos.
Fecha:	Del 6 de febrero al 6 de febrero del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor. **Requisitos a cumplir:** - Reunión con supervisor. - Mecanismos de seguridad. **Forma de aceptación:** La propuesta se indicará como “aprobada” o “no aprobada”.
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.1: Fase de requisitos.

Código fase:	2
Nombre de tarea:	Análisis.
Objetivo:	Realizar un análisis detallado de los requisitos de seguridad identificados en la fase anterior y planificar las soluciones correspondientes.
Descripción:	Se llevará a cabo un análisis exhaustivo de los requisitos de seguridad identificados en la fase de Requisitos.
Fecha:	Del 7 de febrero al 28 de febrero del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor. **Requisitos a cumplir:** - Investigar protocolo MQTT. - Investigar Bridge. - Investigar autenticación basada en credenciales. - Investigar ACLs. - Investigar VPNs. - Investigar servidor MQTT mosquitto. **Forma de aceptación:** La propuesta se indicará como “aprobada” o “no aprobada”.
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.2: Fase de Análisis.

Código fase:	3
Nombre de tarea:	Diseño.
Objetivo:	Planificar y diseñar las soluciones de seguridad, como la implementación de VPN, TLS, ACL, entre otros.
Descripción:	Se realizarán todas las tareas de la fase de diseño.
Fecha:	Del 29 de febrero al 13 de marzo del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor. **Requisitos a cumplir:** - Diseño de la implementación actual. - Diseño de la implementación futura. - Diseño de la implementación del proyecto. - Diseño de la implementación TLS/SSL. - Diseño de la implementación VPN. - Requisitos de diseño del sistema. **Forma de aceptación:** La propuesta se indicará como "aprobada" o "no aprobada".
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.3: Fase de Diseño - Planificación y diseño de soluciones de seguridad.

Código fase:	4
Nombre de tarea:	Programación.
Objetivo:	Implementar las soluciones de seguridad planificadas en la fase de Análisis.
Descripción:	Se realizarán todas las tareas de la fase de programación.
Fecha:	Del 14 de marzo al 23 de abril del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor. **Requisitos a cumplir:** - Pasos previos a la implementación. - Implementación mínima MQTT. - Implementación credenciales MQTT. - Implementación ACLs MQTT. - Implementación certificados TLS/SSL. - Implementación VPNs. **Forma de aceptación:** La propuesta se indicará como "aprobada" o "no aprobada".
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.4: Fase de Programación - Implementación de soluciones de seguridad.

Código fase:	5
Nombre de tarea:	Pruebas.
Objetivo:	Validar que las medidas de seguridad implementadas funcionen correctamente y cumplan con los requisitos establecidos.
Descripción:	Se realizarán todas las tareas de la fase de pruebas.
Fecha:	Del 24 de abril al 29 de abril del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor **Requisitos a cumplir:** - Pruebas desde el exterior del entorno de la empresa. **Forma de aceptación:** La propuesta se indicará como "aprobada" o "no aprobada".
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.5: Fase de Pruebas - Validación de las medidas de seguridad.

Código fase:	6
Nombre de tarea:	Mantenimiento.
Objetivo:	Citar las posibles tareas de mantenimiento que necesitará el sistema después de la implementación de las solución escogida.
Descripción:	Se realizarán todas las tareas de la fase de mantenimiento.
Fecha:	Del 30 de abril al 30 de abril del 2024.
Criterios de aceptación	**Responsable de aceptación:** Tutor **Requisitos a cumplir:** - Gestión de certificados. - Verificación VPN. - Ajustes ACLs. **Forma de aceptación:** La propuesta se indicará como "aprobada" o "no aprobada".
Responsabilidad:	**Responsable:** Anas. **Aprueba:** Tutor. **Consultado:** Supervisor. **Informado:** Supervisor.

Cuadro 2.6: Fase de Mantenimiento - Establecimiento de un plan de mantenimiento.

2.2.5. Planificación temporal

Fase	Subtarea	Duración (horas)
Requisitos	Reunión con supervisor	2
	Mecanismos de seguridad	3
Análisis	Investigar protocolo MQTT	15
	Investigar Bridge	5
	Investigar autenticación basada en credenciales	15
	Investigar ACLs	15
	Investigar VPNs	15
	Investigar servidor MQTT mosquitto	15
Diseño	Estudio del diseño de la implementación actual	10
	Diseño de la implementación futura	10
	Diseño de la implementación del proyecto	10
	Diseño de la implementación TLS/SSL	10
	Diseño de la implementación VPN	10
	Requisitos de diseño del sistema	10
Programación	Pasos previos a la implementación	5
	Implementación mínima MQTT	20
	Implementación credenciales MQTT	25
	Implementación ACLs MQTT	25
	Implementación certificados TLS/SSL	30
	Implementación VPNs	30
Pruebas	Pruebas desde el exterior del entorno de la empresa	15
Mantenimiento	Gestión de certificados	5
	Verificación VPN	
	Ajustes ACL	

Cuadro 2.7: Horas dedicadas a cada fase y subtarea del proyecto

Sumando las horas dedicadas a cada fase y subtarea nos da un **total de 300 horas**.

2.2.6. Diagrama de Gantt

En las figuras 2.3 y 2.4 se muestra el diagrama de Gantt.

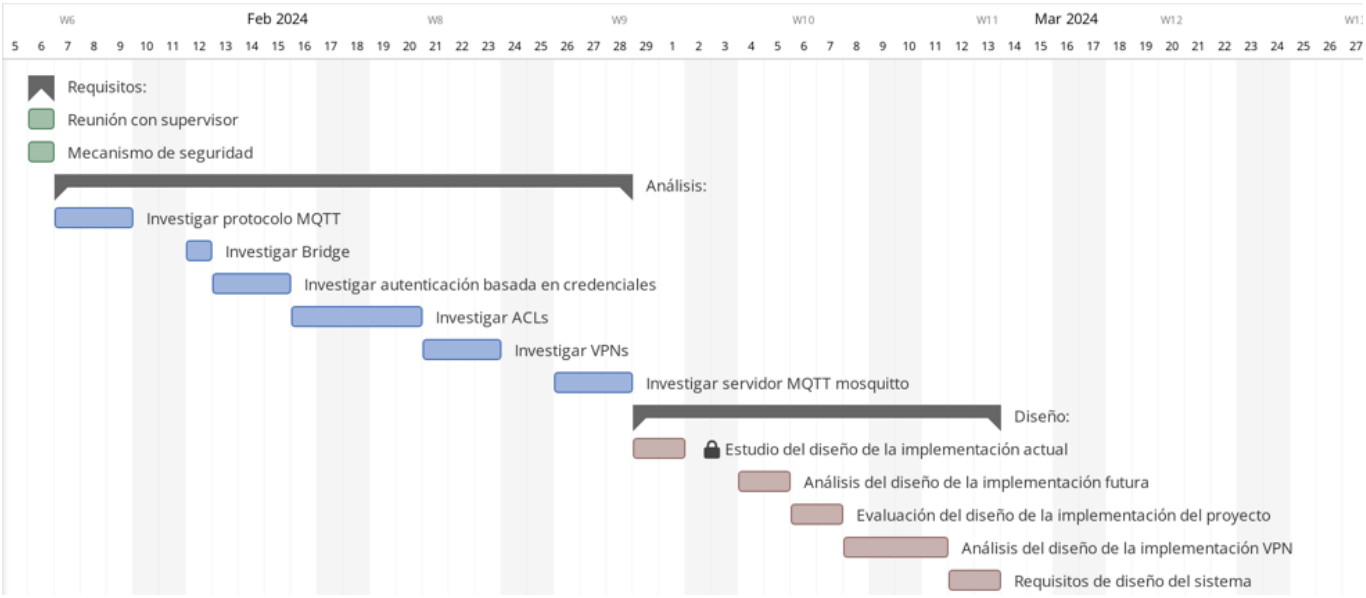


Figura 2.3: Diagrama de Gantt proyecto

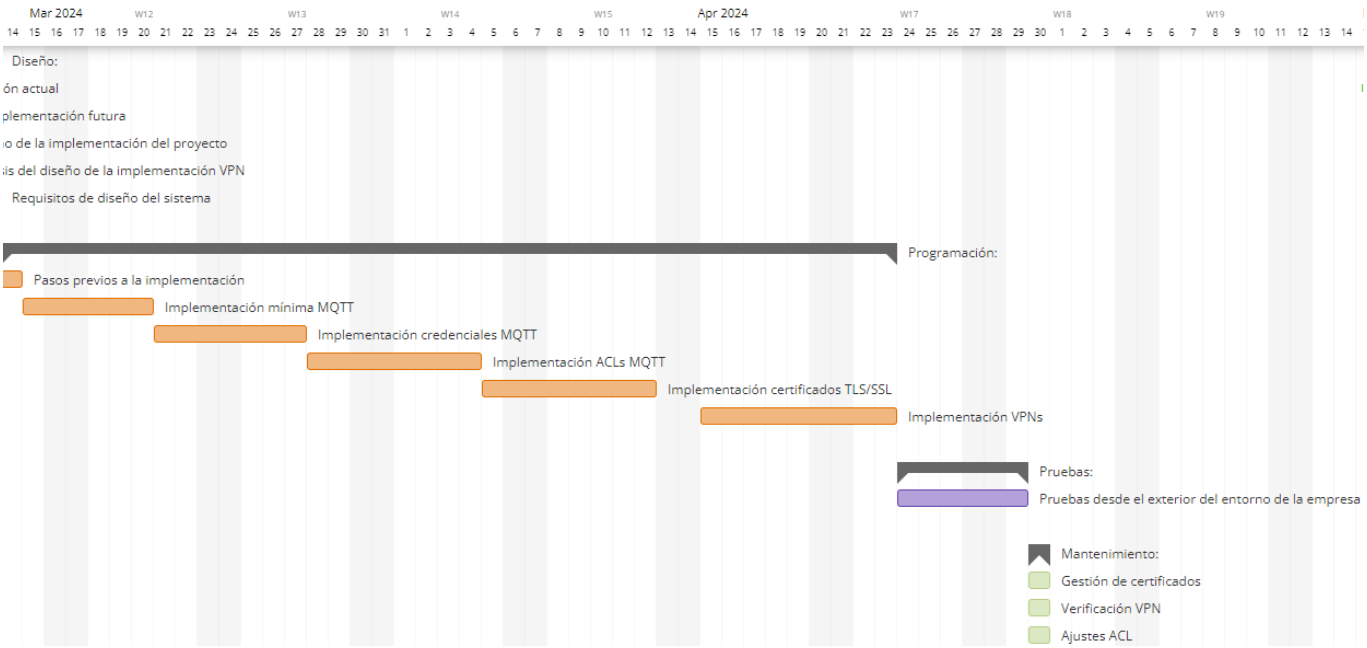


Figura 2.4: Diagrama de Gantt proyecto

2.2.7. Seguimiento del proyecto

Durante el desarrollo del proyecto, establecí un proceso de seguimiento **mediante informes quincenales**. En estos informes, detallaba las actividades realizadas la semana anterior, describía el trabajo en curso y delineaba los planes para la próxima semana. También mencionaba los cambios que fueron sucediendo a lo largo de mi instancia, como la decisión de no proceder con la implementación de la VPN IPsec, que fue comunicada a través de estos informes. A pesar de estos cambios, el proyecto continuó avanzando hacia sus objetivos.

La planificación temporal se cumplió, aunque con algunos ajustes menores, lo cual es completamente normal en proyectos de esta naturaleza.

2.3. Costes

2.3.1. Recursos Hardware

Para la conexión del protocolo MQTT, se han necesito **solo 2 dispositivos**. Un dispositivo que actuara como cliente (en este caso se usó la Raspberry pi 4) y el otro dispositivo como servidor (en este caso se instalo una maquina virtual en el servidor de la empresa).

La maquina virtual fue instalada y usada solamente en el último mes, para realizar las pruebas externas desde mi casa a la empresa.

Producto	**Precio**
Raspberry Pi 4	35 euros
Tarjeta mini SD (64GB)	15 euros
2 cables HDMI	10 euros
Ordenador de empresa	600 euros
Pantalla 22”	150 euros
Teclado y ratón	30 euros
Servidor MQTT para realizar las pruebas externas	5 euros/mes
Total Recursos Hardware	**845 euros**

2.3.2. Recursos Humanos

Se estima el salario de un **programador junior en 21.000 euros al año** y el **salario de un programador senior (supervisor) en 59.500 euros al año**.

Considerando que un año tiene aproximadamente 243 días laborables, el costo de los recursos humanos sería:

$$\text{Costo por hora (junior)} = \frac{21,000 \text{ euros/año}}{243 \text{ días laborables/año} \times 8 \text{ horas/día}} \approx 10{,}80 \text{ euros/hora}$$

$$\text{Costo por hora (supervisor)} = \frac{59,500 \text{ euros/año}}{243 \text{ días laborables/año} \times 8 \text{ horas/día}} \approx 30{,}60 \text{ euros/hora}$$

A continuación se muestra el resumen de los costos de recursos humanos:

Recurso	**Cantidad de horas (h)**	**Costo por hora (€/h)**	**Total (€)**
Empleado junior	300	10.80	3.240
Supervisor	50	30.60	1.530
Total			**4.770**

Cuadro 2.8: Costos de recursos humanos para el proyecto

2.3.3. Recursos Software

Como se utilizaron herramientas de **software libre y gratuito**, no hay un costo monetario directo asociado.

A continuación se presentan algunas de las herramientas utilizadas en el proyecto:

- Software para medir la latencia de una red: Iperf3.
- Software VPNs: WireGuard, OpenVPN.
- Maquinas virtuales: Oracle VM VirtualBox.

2.3.4. Costes Indirectos

Consumo de Electricidad

El costo de la electricidad es de 0,15 €/kWh.

Dispositivo	**Potencia (W)**	**Horas de uso**	**Costo Total (€)**
Raspberry Pi 4	5	300	0,225
Ordenador de empresa	200	300	9
Pantalla 22”	30	300	1,35
Servidor MQTT	100	720 (30 días, 24 h/día)	10,8
Total	335	1620	21,375

Cuadro 2.9: Costos de consumo de electricidad

Para calcular el costo total de la electricidad, multiplicamos la suma de la potencia de los dispositivos por las horas de uso, y luego por el costo por kWh (0,15 €). En este caso:

$$335\,\mathrm{W} \times 1620\,\mathrm{h} \times 0,15\,€/\mathrm{kWh} = 21,375\,€$$

Alquiler del Local

Costo mensual: 500 €

El costo mensual del alquiler del local corresponde a un espacio ubicado en las proximidades de la Universidad Jaume I. Este local proporciona un entorno adecuado y bien equipado para llevar a cabo el proyecto.

Internet

Costo mensual: 50 €

La tarifa de Internet incluye una velocidad de conexión de 300 Mbps y acceso ilimitado a la red.

Sumando los costes indirectos, obtenemos la siguiente tabla:

Concepto	**Duración**	**Costo (€)**
Costo eléctrico (Raspberry Pi 4)	300 horas	0,225
Costo eléctrico (Ordenador de empresa)	300 horas	9
Costo eléctrico (Pantalla 22")	300 horas	1,35
Costo eléctrico (Servidor MQTT)	1 mes	10,8
Alquiler del local	3 meses	1500
Internet	3 meses	150
Total Costes Indirectos		**1671,375**

Cuadro 2.10: Costes Indirectos del Proyecto

2.3.5. Costos Totales

Para calcular **los costos totales** del proyecto, sumaremos los costos de los recursos hardware, los costos de los recursos humanos y los costos indirectos.

Concepto	**Costo (€)**
Costo total hardware	845
Costo total recursos humanos	4770
Costo total costes indirectos	1671,375
Costo total del proyecto	**7286,375**

Cuadro 2.11: Costos Totales del Proyecto

Por lo tanto, el costo total del proyecto sería de **7.286,375 euros**.

Capítulo 3

Requisitos y análisis del protocolo MQTT

3.1. Definición de requisitos

En esta sección se establecen los **objetivos del sistema y las especificaciones** necesarias para cumplir con esos objetivos. Se identifican los requisitos funcionales y no funcionales que el sistema debe satisfacer.

3.1.1. Objetivos del sistema

Los objetivos del sistema son:

- Seleccionar e implementar **un método de securización** para el protocolo MQTT.
- Garantizar **la confidencialidad, integridad y autenticidad** de los datos transmitidos a través de MQTT.
- Minimizar el impacto en el rendimiento del sistema debido a las medidas de seguridad implementadas.

3.1.2. Requisitos funcionales

Los requisitos funcionales del sistema incluyen:

1. El sistema debe ser capaz de **establecer conexiones seguras** entre los dispositivos IoT y el servidor MQTT.
2. Debe **garantizar la autenticación** de los dispositivos y del servidor MQTT.

3. Debe **permitir la encriptación de los datos transmitidos** a través del protocolo MQTT.

3.1.3. Requisitos no funcionales

Los requisitos no funcionales del sistema son:

- El sistema debe ser **fácil de configurar y administrar**.
- Debe tener un **impacto mínimo en el rendimiento de la red** y de los dispositivos IoT.
- Debe ser **compatible** con una variedad de plataformas y dispositivos.

3.1.4. Reunión con supervisor

Durante **mi primera reunión** con mi supervisor en la empresa, discutimos la necesidad de fortalecer la seguridad del protocolo MQTT. Se destacó que la implementación básica de MQTT carece de medidas de seguridad, como encriptación o autenticación mediante contraseña para publicar o suscribirse a los temas. Esto **destacó la importancia** de explorar diferentes métodos para securizar el protocolo MQTT en el contexto de nuestro proyecto.

3.1.5. Mecanismos de seguridad

En la reunión con mi supervisor, exploramos diversas alternativas para mejorar la seguridad del protocolo MQTT (el supervisor tenia conocimientos sobre algunos de estos, pero no los tenia implementados). Se debatió sobre los siguientes mecanismos:

1. **Encriptación de datos:** Se propuso la implementación de encriptación para salvaguardar la confidencialidad de los mensajes MQTT durante su transmisión.
2. **Autenticación de clientes:** Consideramos la viabilidad de establecer un sistema de autenticación para verificar la identidad de los clientes que acceden al servidor MQTT.
3. **Control de acceso basado en ACL:** Se analizó la opción de utilizar listas de control de acceso (ACL) para limitar los temas a los que los clientes pueden acceder, restringiendo así el acceso no autorizado a los datos.
4. **Seguridad de la capa de transporte (TLS/SSL):** Evaluamos la posibilidad de implementar TLS/SSL para agregar una capa adicional de seguridad mediante el cifrado de la comunicación entre los clientes MQTT y el servidor.
5. **Implantación de VPNs:** Se consideró la configuración de redes privadas virtuales (VPNs) para establecer conexiones seguras entre dispositivos IoT y el servidor MQTT, protegiendo así la comunicación frente a posibles amenazas externas.

3.2. Análisis de los requisitos de seguridad del protocolo MQTT

En esta sección se realiza un **análisis detallado** de los requisitos del sistema, identificando posibles conflictos o inconsistencias y proponiendo soluciones.

3.2.1. Análisis de los objetivos del sistema

Se analiza cada uno de los **objetivos del sistema** y se verifica su viabilidad y relevancia para el proyecto.

3.2.2. Análisis de los requisitos funcionales

Se verifica que los requisitos funcionales **sean claros, coherentes y completos**, y se identifican posibles dependencias entre ellos.

3.2.3. Análisis de los requisitos no funcionales

Se evalúa la factibilidad de cumplir con los requisitos no funcionales y se identifican posibles limitaciones técnicas o de recursos.

3.3. Análisis protocolo MQTT

En esta sección analizaremos todos los aspectos del protocolo MQTT, desde el análisis de su implementación básica hasta su securización. Además, en estos subapartados, mostraremos una figura para facilitar el entendimiento de dicha implementación.

3.3.1. Distribución de puertos

Antes de entrar en la investigación del protocolo MQTT, es importante entender los dos protocolos principales de la capa de transporte en Internet: TCP (Protocolo de Control de Transmisión) y UDP (Protocolo de Datagrama de Usuario).

El protocolo **TCP** es un protocolo orientado a la conexión que proporciona una comunicación fiable entre aplicaciones. Utiliza un sistema de confirmación de entrega de paquetes para garantizar que los datos se transmitan correctamente y en orden. Muchos servicios importantes, como la navegación web (HTTP), el correo electrónico (SMTP) y la transferencia de archivos (FTP), utilizan TCP para su comunicación.

En el contexto de MQTT, el protocolo TCP proporciona una base sólida para la comunicación confiable entre clientes y brokers MQTT. Al utilizar TCP, MQTT garantiza la entrega confiable de mensajes en el orden correcto, lo que es esencial para aplicaciones donde la integridad de los datos es crítica.

Por otro lado, el protocolo **UDP** es un protocolo sin conexión que ofrece una comunicación no fiable y no ordenada entre aplicaciones. A diferencia de TCP, UDP no establece una conexión antes de enviar datos y no proporciona confirmaciones de entrega ni retransmisiones de paquetes perdidos. Esto hace que UDP sea más rápido y eficiente en aplicaciones donde la velocidad es más importante que la integridad de los datos, como la transmisión de audio y vídeo en tiempo real, la telefonía IP (VoIP) y los juegos en línea.

Distribución de puertos en redes. Los puertos de red se dividen en tres rangos principales:

- **Puertos bien conocidos (Well-Known Ports):** Estos puertos van del 0 al 1023. Están reservados para servicios comúnmente utilizados y están asignados por la Internet Assigned Numbers Authority (IANA). Algunos ejemplos de puertos bien conocidos son el puerto 80 para HTTP, el puerto 443 para HTTPS y el puerto 22 para SSH.
- **Puertos registrados (Registered Ports):** Estos puertos van del 1024 al 49151. Están asignados por la IANA para aplicaciones específicas o servicios registrados por los desarrolladores. Estos puertos pueden ser utilizados por aplicaciones y servicios no tan comunes pero aún necesarios. Por ejemplo, algunos protocolos de comunicación, como el protocolo NFS para el intercambio de ficheros en red local o en Internet (Puerto 2049), utilizan puertos registrados.
- **Puertos dinámicos o privados (Dynamic or Private Ports):** Estos puertos van del 49152 al 65535. Son puertos disponibles para ser utilizados por aplicaciones y servicios de manera temporal y no están reservados por la IANA ni por ningún otro organismo. Se utilizan principalmente para conexiones salientes y son asignados dinámicamente por el sistema operativo cuando una aplicación establece una conexión.

Podemos encontrar más información sobre la distribución de puertos en redes en el siguiente enlace [5].

3.3.2. Investigar protocolo MQTT

En esta sección, llevaremos a cabo un análisis detallado del protocolo MQTT para comprender su funcionamiento, características y aplicaciones.

MQTT, abreviatura de **Message Queuing Telemetry Transport**, es un protocolo de comunicación diseñado para la conectividad M2M (Machine-to-Machine) y la comunicación en entornos de IoT (Internet of Things). Su principal objetivo es **facilitar la comunicación eficiente y fiable** entre dispositivos distribuidos en una red, con un enfoque en la simplicidad y el **bajo consumo de recursos**.

Este protocolo se ha convertido en una **opción popular** para una amplia gama de dispositivos y aplicaciones en el ámbito de la IoT, incluyendo sensores inteligentes, dispositivos portátiles, sistemas de monitoreo remoto y más. Su flexibilidad y escalabilidad lo hacen adecuado para entornos donde se requiere una comunicación eficiente y confiable entre dispositivos con recursos limitados. Podemos obtener más información en el siguiente enlace [7].

MQTT utiliza el puerto TCP 1883 de forma predeterminada para las comunicaciones no cifradas y el puerto TCP 8883 para comunicaciones cifradas utilizando TLS/SSL.

El protocolo MQTT no implementa cifrado de forma nativa, en su lugar, se requiere implementar medidas de seguridad adicionales, como TLS/SSL.

En nuestro diseño implementaremos TLS/SSL mediante certificados para garantizar la seguridad de las comunicaciones MQTT.

Aunque estos son los **puertos estándar** para el protocolo MQTT, en diversos casos los administradores pueden optar por utilizar puertos diferentes por razones de seguridad u otros motivos. Por lo tanto, una buena practica es verificar con el administrador del sistema la elección de los puertos que no estén reservados para otros servicios conocidos o que estén asignados por organismos de estandarización para evitar conflictos y garantizar la compatibilidad.

Ahora procederemos a destacar algunas de sus características:

- **Naturaleza liviana y eficiente:** Una de las características distintivas de MQTT es su naturaleza liviana y su eficiencia en el uso de ancho de banda. Esto esto el porque se utiliza mucho en entornos industriales. Para ver la cantidad mínima de recursos que utiliza, un mensaje de control MQTT mínimo puede tener tan solo dos bytes de datos, ademas de tener una cabecera muy reducida para optimar el ancho de banda de la red.
- **Arquitectura Pub/Sub:** El modelo de publicación/suscripción desacopla a los publicadores de los suscriptores, permitiendo una comunicación asíncrona y escalable. Los mensajes se publican en temas específicos y los suscriptores interesados los reciben sin necesidad de establecer conexiones directas con cada publicador.
- **Implementación sencilla:** La propia implementación del protocolo MQTT es ligera y requiere poca cantidad de código, lo que la hace ideal para dispositivos con recursos limitados, como en nuestro caso, que como hemos mencionado anteriormente, se utilizan Raspberry Pi 4 de apenas 4Gb de RAM.
- **Fiabilidad:** MQTT define tres niveles de calidad de servicio (QoS) para garantizar la entrega confiable de mensajes en función de las necesidades específicas de cada aplicación IoT:

    1. **QoS 0 (Máximo una vez):** El mensaje se envía al broker una vez, sin garantía de entrega o reintentos. Este nivel es adecuado para datos no críticos que toleran pérdidas ocasionales.
    2. **QoS 1 (Al menos una vez):** El broker confirma la recepción del mensaje al dispositivo. Si no se recibe la confirmación, el dispositivo reenvía el mensaje hasta que se reciba correctamente. Este nivel es adecuado para datos importantes que requieren una entrega garantizada, pero toleran un cierto retraso.

3. **QoS 2 (Exactamente una vez):** El broker garantiza que el mensaje se entregue exactamente una vez al suscriptor designado. Este nivel es el más alto en términos de confiabilidad y se utiliza para datos críticos que no pueden tolerar pérdidas ni duplicaciones.

Después de enumerar y explicar alguna de sus características, procederemos a explicar en que consiste el protocolo.

MQTT utiliza un modelo de **publicación/suscripción** (Publish/Subscribe) en el que los dispositivos pueden actuar como publicadores (publishers) que envían mensajes y suscriptores (subscribers) que reciben los mensajes relevantes para ellos. Y entre ellos esta el servidor MQTT, que es quien maneja todas las peticiones.

Para el servidor MQTT hemos escogido mosquitto, ya que era el servidor que tenia implementado la empresa. Al servidor mosquitto también se le suele llamar **broker o servidor MQTT**, pero nosotros **le llamaremos servidor MQTT** para facilitar la comprensión. El servidor MQTT se encarga de gestionar la comunicación entre los dispositivos IoT y las aplicaciones que necesitan acceder a sus datos.

A continuación, en la figura 3.1 mostramos un ejemplo sencillos donde ponemos en practica estas definiciones.

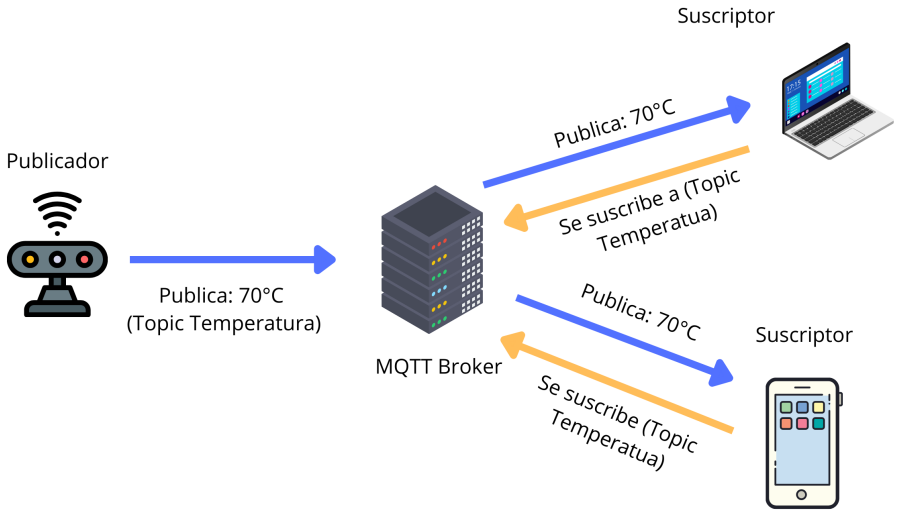


Figura 3.1: Esquema MQTT suscriptor/publicador

Ahora procederemos a explicar la foto.

Como vemos tenemos 3 elementos:

1. **Publicador:** En este caso es un sensor.
2. **MQTT Broker:** En este caso es un servidor mosquitto.
3. **Suscriptor:** En este caso, un portátil y un dispositivo móvil.

El sensor publica la temperatura al servidor MQTT, este mensaje se tiene que publicar en un **tema (topic)**. Este topic es llamado temperatura. Mientras que el sensor va publicando en dicho tema, están los suscriptores que se suscriben al tema temperatura, entonces el servidor MQTT se encarga de enviarles el mensaje de la temperatura, ya que estos están suscritos a dicho tema.

Varios publicadores pueden publicar al mismo tema y ademas varios suscriptores pueden suscribirse al mismo tema. Además de que los publicadores pueden publicar en diferentes temas a la vez, al igual que los suscriptores pueden suscribirse a diferentes temas a la vez.

3.3.3. Bridge

En MQTT, **un bridge** es un componente que permite la interconexión entre dos brokers MQTT distintos. Su función principal es facilitar la comunicación y el intercambio de mensajes entre dos redes o dominios MQTT separados.

La principal característica de un bridge MQTT es su capacidad para suscribirse y publicar mensajes en ambos brokers simultáneamente. Esto significa que un bridge puede recibir mensajes de un broker, reenviarlos al otro y viceversa, actuando como un puente entre los dos brokers.

El **uso de bridges** en MQTT es útil en diversas situaciones, como:

1. **Interconexión de redes MQTT:** Permite la comunicación entre dispositivos y aplicaciones que se encuentran en diferentes redes MQTT, incluso si están ubicadas en diferentes ubicaciones geográficas o administradas por diferentes entidades.
2. **Escalabilidad:** Facilita la expansión de la infraestructura MQTT al conectar múltiples brokers y redes, lo que permite distribuir la carga de trabajo y mejorar la escalabilidad del sistema.
3. **Integración de sistemas:** Permite la integración de sistemas heterogéneos que utilizan diferentes implementaciones o versiones de brokers MQTT, facilitando la interoperabilidad entre ellos.
4. **Federación de datos:** Permite la federación de datos entre diferentes dominios MQTT, lo que facilita el intercambio de información entre organizaciones, proveedores de servicios o dispositivos de IoT pertenecientes a diferentes entidades.

La **interconexión de redes MQTT** es un aspecto fundamental en entornos donde los clientes y el broker están ubicados en diferentes redes, incluso en diferentes ubicaciones geográficas o administradas por diferentes entidades. La implementación de un bridge MQTT permite que los clientes en la red local de la empresa se comuniquen de manera segura con el broker que está fuera de esta red.

Este puente actúa como un enlace vital entre las redes locales y externas, garantizando una **conectividad segura y confiable** en entornos industriales y empresariales. La interconexión de redes MQTT facilita la **comunicación fluida** entre dispositivos y aplicaciones, lo que es

crucial para la integridad y la seguridad de las comunicaciones en entornos donde la **protección de los datos y la privacidad** son de suma importancia.

Ahora que hemos explicado el protocolo, procederemos a explicar sus métodos de securización. En el siguiente enlace podemos encontrar una listado de posibilidades para securizar nuestro protocolo MQTT [3].

3.3.4. Autenticación basada en credenciales

En MQTT, **la autenticación con usuario y contraseña** es una característica de seguridad que permite a los clientes MQTT conectarse a un servidor MQTT (como Mosquitto) proporcionando credenciales de autenticación. Esto significa que antes de que un cliente pueda publicar o suscribirse a temas en el servidor MQTT, debe proporcionar un nombre de usuario y una contraseña válidos.

Cuando un cliente MQTT intenta conectarse a un servidor MQTT que requiere autenticación, envía un mensaje CONNECT que incluye un campo de nombre de usuario y un campo de contraseña. El servidor MQTT valida estas credenciales comparándolas con las credenciales almacenadas en su base de datos de usuarios.

Si el nombre de usuario y la contraseña son correctos, el servidor MQTT permite al cliente conectarse y realizar acciones como publicar o suscribirse a temas. Si las credenciales no son válidas, el servidor MQTT rechaza la conexión y el cliente no puede acceder al servidor.

Esta función de autenticación de usuario y contraseña proporciona una capa adicional de seguridad al protocolo MQTT, ayudando a proteger los recursos y datos de MQTT contra accesos no autorizados. Es particularmente útil en entornos donde se requiere un nivel de seguridad más alto, como en aplicaciones industriales o de IoT donde la integridad de los datos es crítica. Además, evita que personas no autorizadas se conecten al servidor MQTT y comiencen a publicar o suscribirse a temas, lo que podría comprometer la seguridad o inundar el sistema con datos no deseados. En la siguiente figura 3.2 podemos ver un esquema sencillo de esta.

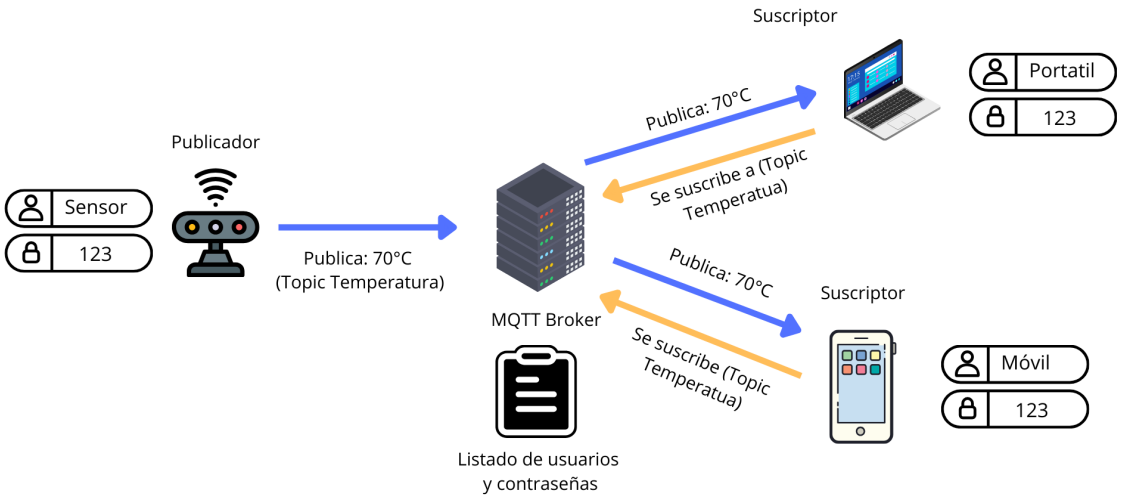


Figura 3.2: Esquema MQTT suscriptor/publicador con autenticación basada en credenciales

3.3.5. ACLs

En MQTT, **las Listas de Control de Acceso (ACLs)** son una característica importante para controlar y gestionar el acceso de los clientes MQTT a los temas (topics) en el servidor MQTT. Las ACLs permiten al administrador del servidor MQTT definir reglas específicas que determinan que clientes tienen permiso para publicar o suscribirse a ciertos temas.

Al igual que con la autenticación basada en credenciales (**si se desea implementar las ACLs, la autenticación debe de estar implementada**), las ACLs añaden una capa adicional de seguridad al protocolo MQTT, ya que permiten un control granular sobre quién puede acceder a qué datos en el servidor MQTT. Esto es crucial en entornos donde se necesitan políticas de acceso estrictas para garantizar la integridad y la seguridad de los datos.

Las reglas de ACL pueden basarse en **varios criterios**, como el nombre de usuario, el identificador del cliente, el nombre del tema y el tipo de acción (publicar o suscribirse). Por ejemplo, se pueden configurar reglas que permitan a ciertos usuarios publicar en un tema específico, pero no suscribirse a él, o viceversa.

La configuración de ACLs suele realizarse en el servidor MQTT, donde **el administrador** define las reglas y políticas de acceso según los requisitos de seguridad y las necesidades del sistema.

Pondremos algunas restricciones para la anterior figura 3.2. En la siguiente figura 3.3 se muestran dichas restricciones.

1. Sensor **solo** tendrá permitido publicar en el tema de Temperatura y no podrá suscribirse

a nada.

2. Móvil tendrá permitido suscribirse a cualquier tema pero no publicar.
3. A portátil no le pondremos ninguna ACL ya que es el administrador.

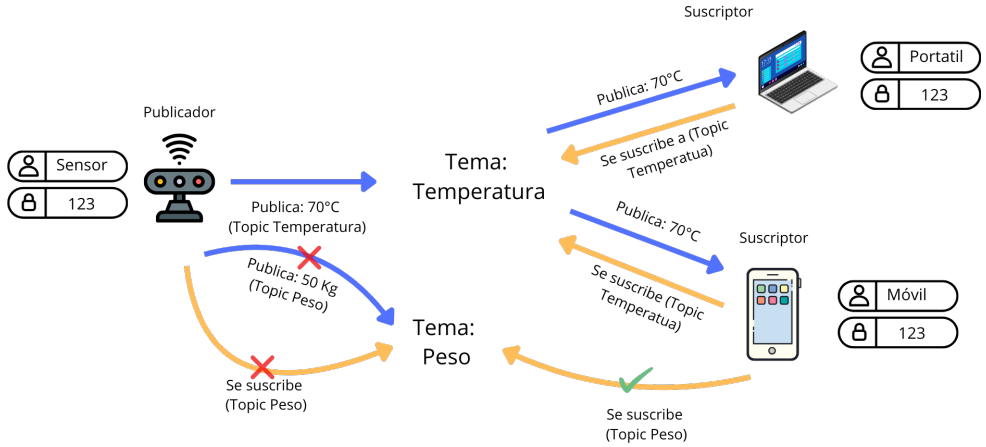


Figura 3.3: Esquema MQTT suscriptor/publicador con ACLs

3.4. Investigar VPNs

Las redes privadas virtuales (VPN) se han convertido en una herramienta esencial para garantizar la seguridad y la privacidad en las comunicaciones en línea. Al crear un túnel seguro entre un dispositivo y un servidor VPN, las VPN permiten a los usuarios navegar por internet, acceder a recursos remotos y comunicarse de manera confidencial, incluso en redes públicas no seguras.

En este apartado, analizaremos tres de las VPN más populares y utilizadas: WireGuard, OpenVPN e IPsec, evaluando sus características, ventajas y desventajas para determinar cuál se adapta mejor a nuestras necesidades.

1. **WireGuard:**
   - **Características:**
     - Protocolo VPN moderno y ligero, diseñado para la simplicidad y el rendimiento. Utiliza criptografía de última generación, como ChaCha20-Poly1305 y Curve25519, para garantizar la seguridad y la velocidad.
     - Implementación eficiente en términos de recursos, ideal para dispositivos con recursos limitados.
     - Configuración sencilla y fácil de usar.

- **Ventajas:**
  - Alto rendimiento y baja latencia, ideal para juegos en línea y streaming de vídeo.
  - Mayor eficiencia en el uso de recursos, lo que lo hace adecuado para dispositivos móviles y con poca potencia de procesamiento.
  - Facilidad de implementación y gestión.
- **Desventajas:**
  - Protocolo relativamente nuevo, con una comunidad de usuarios y soporte técnico menos establecidos que OpenVPN o IPsec.
  - Menor compatibilidad con algunos sistemas operativos y dispositivos más antiguos.

2. **OpenVPN:**
   - **Características:**
     - Protocolo VPN maduro y ampliamente utilizado, con una gran comunidad de usuarios y soporte técnico.
     - Altamente configurable y personalizable, lo que permite adaptarlo a diversas necesidades.
     - Compatible con una amplia gama de plataformas, sistemas operativos y dispositivos. Soporta una gran variedad de protocolos de cifrado y autenticación.
   - **Ventajas:**
     - Amplia compatibilidad y soporte técnico.
     - Alta flexibilidad y capacidad de personalización.
     - Amplia gama de opciones de configuración para diferentes necesidades.
   - **Desventajas:**
     - Puede ser más complejo de configurar y administrar que WireGuard.
     - Mayor consumo de recursos, lo que puede afectar el rendimiento en dispositivos con recursos limitados.

3. **IPsec:**
   - **Características:**
     - Suite de protocolos de seguridad de red que incluye VPN.
     - Estándar ampliamente utilizado e integrado en la mayoría de los sistemas operativos modernos.
     - Ofrece un alto nivel de seguridad y confiabilidad.
     - Soporta una amplia gama de algoritmos de cifrado y autenticación.
   - **Ventajas:**
     - Alto nivel de seguridad y confiabilidad.
     - Amplia compatibilidad e integración con sistemas operativos.
     - Soporte para una variedad de dispositivos y plataformas.
   - **Desventajas:**
     - Puede ser mucho más complejo de configurar que WireGuard o OpenVPN.
     - Mayor consumo de recursos, lo que puede afectar el rendimiento en dispositivos con recursos limitados.

Característica	**WireGuard**	**OpenVPN**	**IPsec**
Protocolo	Moderno	Maduro	Estándar
Rendimiento	Alto	Medio	Alto
Latencia	Baja	Media	Alta
Consumo de recursos	Bajo	Medio	Alto
Facilidad de uso	Fácil	Media	Difícil
Compatibilidad	Media	Alta	Alta
Soporte técnico	Medio	Alto	Alto

Cuadro 3.1: Comparación de VPNs

Si observamos la anterior tabla, podemos sacar las siguientes conclusiones: WireGuard se posiciona como la opción más destacada entre las VPNs evaluadas. Su protocolo moderno ofrece un rendimiento excepcionalmente alto, con una latencia baja y un consumo de recursos reducido. Además, su facilidad de uso lo hace accesible para una amplia gama de usuarios. En términos de seguridad, WireGuard ofrece un enfoque sólido con su diseño simplificado y eficiente, debido a sus unicamente 4000 lineas de código, lo que la hace fácil de auditar.

Como hemos citado anteriormente, WireGuard usa en su naturaleza criptografía de ultima generación, como puede ser Curve25519. Este es un algoritmo ECC (Criptografía de curva elíptica). Es verdad que en las otras VPNs también se puede usar ECC, pero no están de manera nativa en su diseño, habría que hacer cambios para que usasen este, mientras que WireGuard si que lo implementa. Esto tiene una gran importancia ya que ECC usa un algoritmo ECDSA (Algoritmo de firma digital de curva elíptica), que este puede lograr un cifrado más fuerte con menos potencia informática y ancho de banda. Esto es crucial y muy ventajoso para dispositivos IoT (Internet Of Things) que tienen recursos limitados. Podemos encontrar más información sobre la criptografía de curva elíptica en el siguiente enlace [9].

3.5. Investigar servidor MQTT mosquitto

Mosquitto es un servidor de código abierto implementado en el protocolo MQTT. Es uno de los servidores MQTT más populares y ampliamente utilizados debido a su simplicidad, rendimiento y escalabilidad. Desarrollado por Eclipse Foundation, Mosquitto está escrito en C y se puede ejecutar en una variedad de sistemas operativos, incluidos Linux, Windows y macOS.

Entre sus características principales se incluyen la compatibilidad con el protocolo MQTT versión 3.1 y 3.1.1, soporte para TLS/SSL para conexiones seguras, autenticación basada en usuario y contraseña, ACLs para control de acceso, y muchas más.

Como hemos dicho anteriormente, hemos seleccionado este servidor MQTT ya que la empresa lo tenia en uso lo que nos permitía aprovechar la experiencia previa y el conocimiento adquirido por parte del equipo técnico en su implementación y gestión.

Capítulo 4

Diseño

Aunque la implementación de la solución no se llevará a cabo en el entorno real en el momento actual, esta propuesta de arquitectura está diseñada para un proyecto futuro. Este documento proporciona todos los detalles necesarios para implementar la seguridad cuando sea necesario en el futuro. Se presenta una visión general de la arquitectura propuesta para el sistema MQTT, incluyendo la estructura de componentes, los módulos principales, las interacciones entre ellos y los flujos de datos. Aunque la implementación real aún no se ha realizado, esta descripción sienta las bases para la implementación futura, proporcionando una guía clara para la seguridad en el entorno MQTT.

En nuestro caso, usaremos como sistema operativo del servidor, usaremos Debian 11 y en el cliente Raspberry Pi OS. Estos son los sistemas operativos que se están usando actualmente en la empresa.

4.1. Diseño de la implementación actual

La siguiente imagen 4.1 representa **el diseño actual** de implementación en la empresa. Como vemos el servidor MQTT está alojado dentro de la empresa. Los circuitos que vemos en la imagen son los dispositivos Raspberry Pi 4. Cuando estos se comunican con el servidor, lo hacen localmente y no hay mucho riesgo de robo de datos, lectura de paquetes con herramientas como Wireshark ya que no viajan a través de Internet.

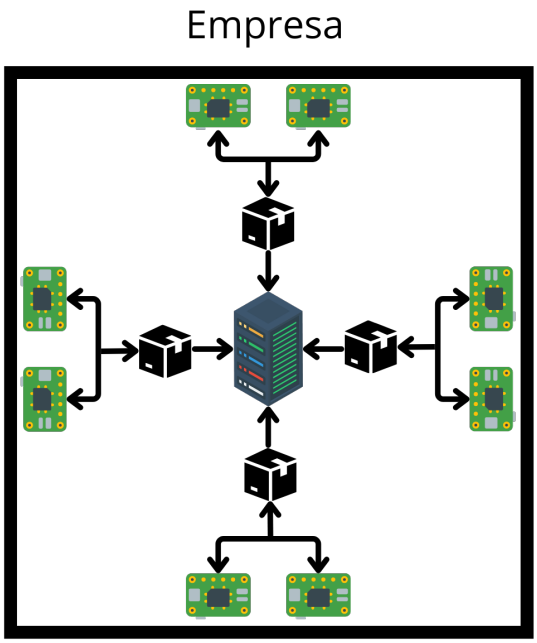


Figura 4.1: Diseño actual de la implementación

4.2. Diseño de la implementación futura

Pero en el **futuro diseño** en el que se quiere implementar el protocolo MQTT, como observamos en la siguiente imagen 4.2, los paquetes MQTT enviados por las Raspberry Pi 4, viajan a través de Internet, exponiendo así la información que contienen estos. Para no exponer dicha información, se tendrá que securizar dicho protocolo.

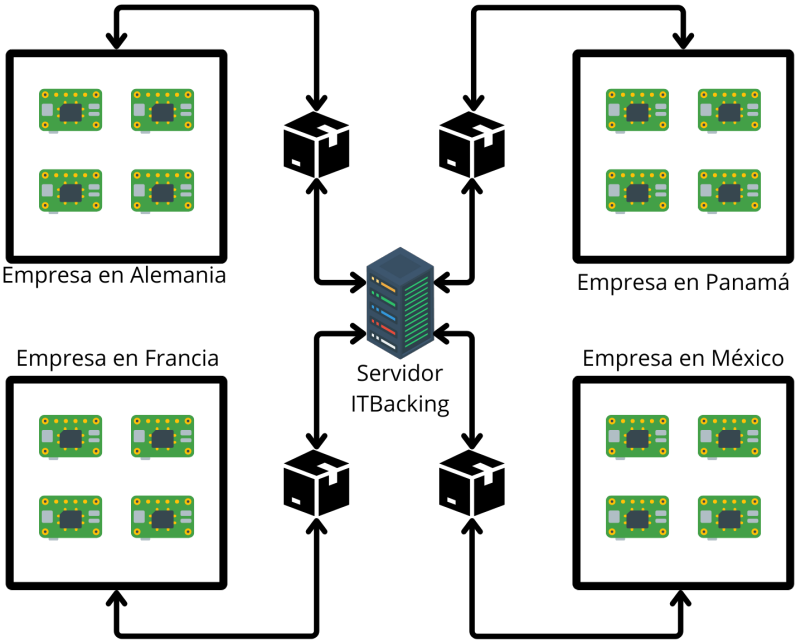


Figura 4.2: Diseño de la implementación a futuro

4.3. Diseño de la implementación del proyecto

En el diseño de la implementación del proyecto, dado que la infraestructura aún no está establecida, voy a configurar un entorno desde mi casa hacia la empresa para simular la infraestructura futura. En este escenario, el servidor estará ubicado en la empresa, mientras que el cliente estará alojado en mi casa.

4.4. Diseño de la implementación TLS/SSL

En la siguiente figura 4.3 podemos ver como procederemos con el diseño de la implementación de TLS/SSL.

Con TLS (Transport Layer Security), la información se **cifra punto a punto** mientras se transmite a través de Internet. Esto significa que cuando dos dispositivos se comunican utilizando TLS, los datos se cifran en el origen y se descifran en el destino, lo que proporciona **seguridad durante la transferencia de datos a través de una red no confiable como Internet**.

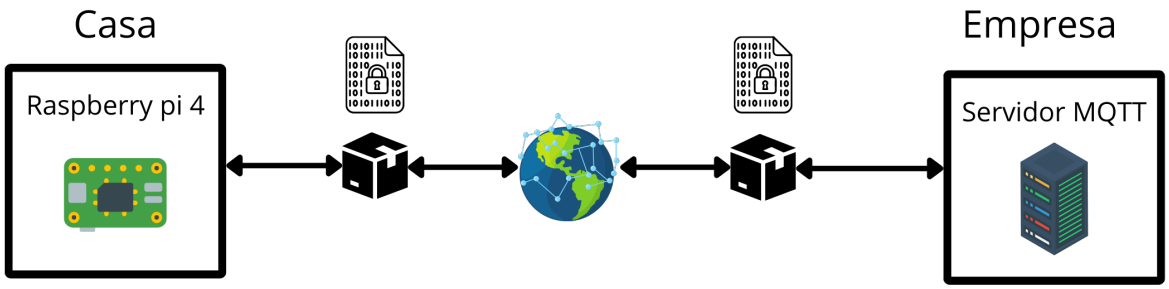


Figura 4.3: Diseño de la implementación TLS/SSL

4.5. Diseño de la implementación VPN

En la siguiente figura 4.4 podemos ver como procederemos con el diseño de la implementación VPNs.

Con una VPN (Red Privada Virtual), la información también **se cifra**, pero en lugar de cifrarla y descifrarla punto a punto, se encapsula en un túnel seguro que atraviesa Internet. Dentro de este túnel, los datos están protegidos mediante cifrado, pero en lugar de descifrarlos en cada extremo del túnel, se mantienen cifrados hasta llegar al destino final de la red privada.

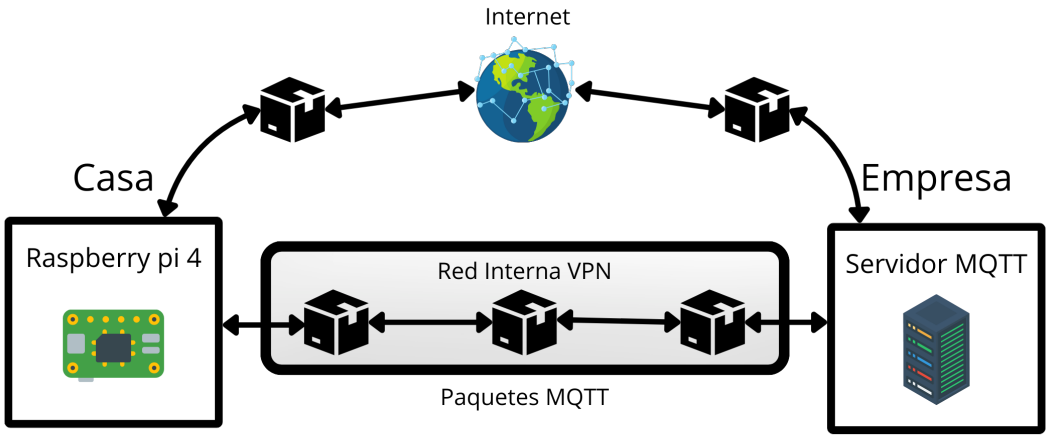


Figura 4.4: Diseño de la implementación VPN

4.6. Requisitos del diseño

Vamos a enumerar los requisitos del diseño del sistema para su correcto funcionamiento.

- El servidor MQTT tenga acceso con credenciales.
- El servidor MQTT tenga ACLs.
- La Raspberry pi 4 tenga un bridge con el servidor MQTT.
- Se use TLS/SSL para la comunicación entre la Raspberry pi 4 y el servidor MQTT.
- Los mensajes de las Raspberry pi 4 hacia el servidor MQTT lleguen a través del bridge.

Capítulo 5

Implementación y pruebas

En este apartado explicaremos en detalle como implementar tanto el protocolo MQTT hasta su securización, pasando también por las VPNs. A medido que vaya implementando el protocolo, se irán haciendo pruebas para comprobar el funcionamiento de este mismo.

5.1. Pasos previos a la implementación

Antes de proceder con la implementación del protocolo MQTT, es crucial realizar algunos pasos previos para garantizar una comunicación segura, especialmente al trabajar desde casa. Además de abrir los puertos necesarios para MQTT, también se deben abrir otros puertos para permitir el acceso remoto a servicios adicionales, como SSH.

El puerto 22 se utiliza comúnmente para establecer conexiones SSH, lo que permite trabajar de forma remota en el servidor alojado en la empresa. Abrir este puerto facilitará la conexión y el acceso a recursos y servicios en el entorno empresarial desde casa.

Es importante verificar **la configuración del firewall y del enrutador** para asegurarse de que los puertos relevantes estén abiertos tanto en el lado del cliente como en el lado del servidor. Además, se deben seguir las mejores prácticas de seguridad al abrir estos puertos, como restringir el acceso solo a las direcciones IP necesarias y utilizar autenticación adecuada para proteger la red contra posibles amenazas.

En este caso, el servidor MQTT de prueba que estaba **alojado en la empresa** me fue facilitado por el administrador de sistemas de la empresa. Este servidor era gestionado exclusivamente por el administrador de sistemas, ya que no tenía acceso para administrarlo ni para abrir puertos. En su lugar, comunicaba mis necesidades al administrador, quien se encargaba de realizar las configuraciones correspondientes.

Caso servidor:

Estos son las configuraciones que el administrador aplicó al servidor alojado en la empresa:

- Solamente otorgó permiso a mi dirección IP pública para acceder él.

- Abrió diversos puertos en el firewall para facilitar la comunicación con el servidor MQTT.

  Los puertos que el administrador abrió en el firewall del servidor son lo siguientes:

  1. El puerto 22 (SSH).
  2. El puerto 1885 (MQTT sin encriptación, es verdad que el puerto estándar es el 1883, pero en este caso estaba ocupado).
  3. El puerto 8883 (MQTT con TLS/SSL).
  4. El puerto 51194 (WireGuard VPN).
  5. El puerto 1195 (OpenVPN, es verdad que el puerto estándar es el 1194, pero en este caso estaba ocupado).
  6. El puerto 4500 (IPSec NAT-T).
  7. El puerto 500 (Para el intercambio de claves en Internet, lo necesitamos para el correcto funcionamiento de IPSec IKE).

Caso cliente:

En mi caso, como yo hacia las peticiones desde mi casa hacia el servidor alojado en la empresa, no tenia que abrir todos los puertos, solo los del MQTT y las VPNs.

En el anexo C detallaremos este proceso.

5.2. Implementación mínima MQTT

Empezaremos primero con la instalación mínima del protocolo MQTT, mas adelante haremos una implementación mas completa, implementado TLS/SSL, ACLs, etc.

Para conectarnos al servidor del cliente utilizaremos un programa llamado Putty, que es un cliente para conectarse mediante SSH. Introducimos la IP publica y nos conectamos al servidor.

Cuando estemos en la consola, introducimos el comando **sudo apt install mosquitto** y **sudo apt install mosquitto-clients**. El primer comando es para el servidor mosquitto y el segundo comando son las herramientas de lineas de comando que utilizaremos más tarde.

Podemos ver en la siguiente imagen 5.1 la estructura de ficheros que se crearon. Accedemos a la ruta donde nos hemos descargado mosquitto, en este caso es /etc/mosquitto para ver todos los ficheros. De momento para configurar las funciones básicas sin TLS/SSL, modificaremos el fichero de configuración mosquitto.conf. Los demás ficheros mientras vayamos usándolos los iremos nombrando.

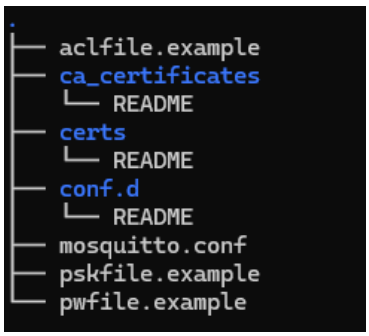


Figura 5.1: Estructura de ficheros servidor mosquitto

Usaremos el editor de texto nano, una herramienta sencilla para editar ficheros.

Abrimos el fichero de configuración mosquitto.conf y introducimos las siguientes líneas mostradas en la imagen 5.2.

```
  GNU nano 5.4                /etc/mosquitto/mosquitto.conf *
allow_anonymous true
listener 1885

persistence true
persistence_location /var/lib/mosquitto/
```

Figura 5.2: Configuración básica MQTT

El parámetro `allow_anonymous` con el valor `true` permite aceptar cualquier petición con un usuario desconocido. Por el momento, lo dejaremos así. Cuando implementemos credenciales, cambiaremos el parámetro a `false`.

El listener 1885 es el puerto por el cual el protocolo MQTT estará escuchando. Si ejecutamos el comando `lsof -i:1885`, podemos ver qué proceso está escuchando en dicho puerto. Esto es muy importante, ya que a veces el puerto estará ocupado y tendremos que detener ese proceso para poder ejecutar el servidor MQTT. En la figura 5.3 podemos ver este proceso.

```
anas@anas:/etc/mosquitto$ /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
1715563644: mosquitto version 2.0.11 starting
1715563644: Config loaded from /etc/mosquitto/mosquitto.conf.
1715563644: Opening ipv4 listen socket on port 1885.
1715563644: Error: Address already in use
anas@anas:/etc/mosquitto$ sudo lsof -i:1885
COMMAND     PID      USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
mosquitto 57990 mosquitto    4u  IPv4 202029      0t0  TCP *:1885 (LISTEN)
mosquitto 57990 mosquitto    5u  IPv6 202030      0t0  TCP *:1885 (LISTEN)
anas@anas:/etc/mosquitto$ sudo kill 57990
anas@anas:/etc/mosquitto$ /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
1715563666: mosquitto version 2.0.11 starting
1715563666: Config loaded from /etc/mosquitto/mosquitto.conf.
1715563666: Opening ipv4 listen socket on port 1885.
1715563666: Opening ipv6 listen socket on port 1885.
1715563666: mosquitto version 2.0.11 running
```

Figura 5.3: Matar proceso que ocupa un puerto

Si deseamos que los mensajes que recibimos o enviamos se guarden, podemos configurar el parámetro `persistence_location` para que guarde dichos mensajes en una base de datos.

Ahora vamos a la parte del cliente, nos descargamos mosquitto y editamos el fichero de configuración mosquitto.conf y introducimos lo mismo que la implementación del servidor mosquitto. En la imagen 5.2 tenemos el código a implementar.

Ahora procederemos a explicar los comandos básicos para publicar y suscribise a un tema (topic).

5.2.1. mosquitto_pub

```
mosquitto_pub -h <servidor> -p <puerto> -t <tema> -m <mensaje>
```

Parámetros:

- `-h <servidor>`: Especifica el nombre o la dirección IP del servidor MQTT al que deseas conectarte.
- `-p <puerto>`: Especifica el puerto del servidor MQTT al que deseas conectarte. El puerto predeterminado es 1883.
- `-t <tema>`: Especifica el tema al que deseas publicar el mensaje.
- `-m <mensaje>`: Especifica el contenido del mensaje que deseas publicar en el tema especificado.

5.2.2. mosquitto_sub

```
mosquitto_sub -h <servidor> -p <puerto> -t <tema>
```

Parámetros:

- `-h <servidor>`: Especifica el nombre o la dirección IP del servidor MQTT al que deseas conectarte.
- `-p <puerto>`: Especifica el puerto del servidor MQTT al que deseas conectarte. El puerto predeterminado es 1883.
- `-t <tema>`: Especifica el tema al que deseas suscribirte para recibir mensajes.

Otro comando importante para mostrar es el comando para visualizar el registro de Mosquitto:

```
/usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Este comando ejecutará Mosquitto con el archivo de configuración especificado ('/etc/mosquitto/mosquitto.conf').

Cuando ya tengamos el servidor y el cliente listos, procederemos a hacer un pequeño test para probar la implementación básica.

Ejecutamos el registro en el servidor y nos suscribimos al tema "test"para realizar un pequeño intercambio de mensajes. Anteriormente hemos explicado el comando para suscribirse a un tema, el comando `mosquitto_pub`. Ahora en el lado del cliente publicaremos un mensaje al tema "test"poniendo como IP la del servidor. En el log del servidor observamos que alguien se ha conectado (que seremos nosotros porque nos hemos suscrito a nuestra IP) y veremos otro mensaje de conexión que será el publicador, y en el suscriptor miraremos que mensaje ha enviado.

- Ejecutamos el log en el servidor. En la figura 5.4 podemos ver la ejecución de este.

```
anas@anas:/etc/mosquitto$ /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
1715564663: mosquitto version 2.0.11 starting
1715564663: Config loaded from /etc/mosquitto/mosquitto.conf.
1715564663: Opening ipv4 listen socket on port 1885.
1715564663: Opening ipv6 listen socket on port 1885.
1715564663: mosquitto version 2.0.11 running
```

Figura 5.4: Log mosquitto MQTT servidor

- Nos suscribimos desde el servidor a nuestra propia IP , al tema "testz al puerto 1885. En la figura 5.5 podemos observar como nos suscribimos.

```
anas@anas:~$ mosquitto_sub -h 212.230.233.50 -p 1885 -t test
```

Figura 5.5: Suscriptor desde el servidor MQTT al servidor MQTT

- Miramos el log y observamos que hemos recibido una conexión desde nuestra propia IP. En la figura 5.4 podemos ver el log de este.

```
1715564831: New connection from 212.230.233.50:42794 on port 1885.
1715564831: New client connected from 212.230.233.50:42794 as auto-18BFD1FE-EDE8-B
8EE-D7BF-182A0D9D889E (p2, c1, k60).
```

Figura 5.6: Conexión del suscriptor en el log

- Publicamos desde el cliente al servidor un mensaje de prueba, ponemos la IP pública del servidor, el tema "test", puerto 1885 y un mensaje "hola". En la figura 5.7 vemos como publicar dicho mensaje.

```
itb@CRIB-ENTESM:~ $ mosquitto_pub -h 212.230.233.50 -t test -p 1885 -m hola
```

Figura 5.7: Publicación del cliente hacia el servidor MQTT

- Luego nos dirigimos al log y vemos que se ha conectado una IP (es mi IP publica de casa). En la figura 5.8 vemos como mirar el log.

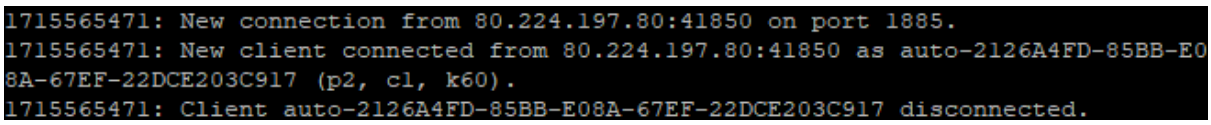


Figura 5.8: Conexión del publicador en el log

- Y finalmente nos dirigimos al suscriptor y vemos el mensaje "hola". En la figura 5.9 nos muestra que hemos recibido el mensaje.

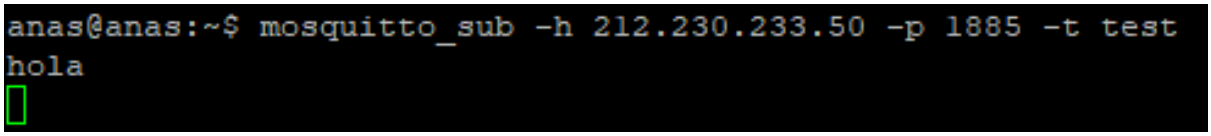


Figura 5.9: Mensaje del publicador en el suscriptor

Tras esta implementación y pruebas, ya hemos implementado lo básico protocolo MQTT.

Ahora procederemos a agregar credenciales al servidor.

5.3. Implementación credenciales MQTT

Ahora que tenemos la implementación básica, procederemos a implementar las credenciales de usuario y contraseña para el servidor.

Nos situamos en la carpeta /etc/mosquitto.

Para agregar un usuario podemos hacerlo de diversas maneras:

La primera forma es si aun no tenemos creado aun el fichero de contraseñas.

```
mosquitto_passwd -c /ruta/al/archivo/passwordfile username
```

Parámetros:

- `-c /ruta/al/archivo/passwordfile`: Crea un nuevo archivo de contraseñas en la ruta especificada.
- `username`: Nombre de usuario que deseas agregar al archivo de contraseñas.

Luego nos pedirá que introduzcamos la contraseña 2 veces, después de introducirla se nos habrá creado **un fichero** con el usuario y contraseña. En la figura 5.10 podemos observar esto.

Si queremos revisar que se han introducido correctamente, haremos un cat del fichero passwf. En la figura 5.11 podemos observar como revisar el fichero passwf.

```
anas@anas:/etc/mosquitto$ sudo mosquitto_passwd -c /etc/mosquitto/passwf cliente
Password:
Reenter password:
anas@anas:/etc/mosquitto$ cat passwf
cliente:$7$101$Qb6RnJZGEFkJY3Qy$PfnGjjQUnfSSdlDjkA+0EjcYNkoE+OBf16HVaKD+KPjUyPmY
niRCWBEY+BCg6M49TzDuvrm1g25pZMK4YCpt5A==
```

Figura 5.10: Crear credenciales con código

```
anas@anas:/etc/mosquitto$ cat passwf
cliente:$7$101$Qb6RnJZGEFkJY3Qy$PfnGjjQUnfSSdlDjkA+0EjcYNkoE+OBf16HVaKD+KPjUyPmY
niRCWBEY+BCg6M49TzDuvrm1g25pZMK4YCpt5A==
```

Figura 5.11: Contenido fichero passwf

Ahora si necesitamos agregar a otro usuario lo haremos con el siguiente comando:

mosquitto_passwd –b /ruta/al/archivo/passwordfile username password

Los parámetros son casi los mismos, solamente esta -b, para decirle que queremos agregar un usuario a un fichero, y podemos introducir la contraseña en el propio comando.

En la figura 5.12 vemos que se nos ha agregado correctamente.

```
anas@anas:/etc/mosquitto$ sudo mosquitto_passwd -b /etc/mosquitto/passwf cliente
2 123
anas@anas:/etc/mosquitto$ cat passwf
cliente:$7$101$Qb6RnJZGEFkJY3Qy$PfnGjjQUnfSSdlDjkA+0EjcYNkoE+OBf16HVaKD+KPjUyPmY
niRCWBEY+BCg6M49TzDuvrm1g25pZMK4YCpt5A==
cliente2:$7$101$EbSJQEvF1BN1AnXF$rMETH6/WpOdoeX0IVimgnrHxK5lXIKMbecPaV/mki7oEe3b
3OxV9dHcZEWI4QskaSyRJ9d9VJzZSH4B5zLNq+g==
```

Figura 5.12: Contenido fichero passwf

Ahora vamos a eliminar un usuario del fichero passwf.

Utilizaremos el siguiente comando:

mosquitto_passwd –D /ruta/al/archivo/passwordfile username

Vemos en la figura 5.13 que cliente2 ya no esta en el fichero passwf.

```
anas@anas:/etc/mosquitto$ sudo mosquitto_passwd -D /etc/mosquitto/passwf cliente
2
anas@anas:/etc/mosquitto$ cat passwf
cliente:$7$101$Qb6RnJZGEFkJY3Qy$PfnGjjQUnfSSdlDjkA+0EjcYNkoE+OBf16HVaKD+KPjUyPmY
niRCWBEY+BCg6M49TzDuvrm1g25pZMK4YCpt5A==
```

Figura 5.13: Contenido fichero passwf sin usuario cliente2

Ahora que ya tenemos el usuario cliente en el fichero passwf, vamos introducimos la siguiente linea en el fichero de configuración de mosquitto. También, como hemos nombrado anteriormente, cambiamos el valor de `allow_anonymous` de `true` a `false`, ya que ahora no dejaremos entrar a ningún usuario que no tenga credenciales.

Nuestra configuración quedaría tal que en la siguiente figura 5.14:

```
  GNU nano 5.4              /etc/mosquitto/mosquitto.conf
allow_anonymous false
listener 1885
password_file /etc/mosquitto/passwf

persistence true
persistence_location /var/lib/mosquitto/
```

Figura 5.14: Configuración fichero mosquitto.conf con credenciales

Ahora cuando nos suscribamos y publiquemos tendremos que agregar las credenciales en el comando, quedaría tal que así:

```
mosquitto_pub -h <servidor> -p <puerto> -t <tema>
-u <usuario> -P <password>

mosquitto_sub -h <servidor> -p <puerto> -t <tema> -m <mensaje>
-u <usuario> -P <password>
```

Parámetros: Nos encontramos con 2 nuevos parámetros.

- `-u <usuario>`: Especifica el nombre de usuario creado en el fichero passwf.
- `-P <contraseña>`: Especifica la contraseña creada en el fichero passwf.

Para comprobar la funcionalidad de la implementación seguiremos los siguientes pasos:

1. Ejecutamos el log en el servidor.
2. Nos suscribimos desde el servidor al servidor.
3. Publicamos desde la Raspberry pi 4 hacia el servidor. En la siguiente figura 5.15 podemos observar como publicamos.

```
itb@CRIB-ENTESM:~ $ mosquitto_pub -h 212.230.233.50 -t test -p 1885 -m hola -u c
liente -P 123
```

Figura 5.15: Publicación con credenciales desde Raspberry pi 4

4. Miramos que nos haya llegado el mensaje en el suscriptor.

   En la siguiente figura 5.16 podemos ver que nos ha llegado el mensaje.

```
anas@anas:~$ mosquitto_sub -h 212.230.233.50 -p 1885 -t test -u cliente -P 123
hola
```

Figura 5.16: Suscriptor con credenciales recibe mensaje de Raspberry pi 4

5. Por ultimo miraremos los logs y analizaremos la siguiente figura 5.17.

```
anas@anas:/etc/mosquitto$ /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
1715605735: mosquitto version 2.0.11 starting
1715605735: Config loaded from /etc/mosquitto/mosquitto.conf.
1715605735: Opening ipv4 listen socket on port 1885.
1715605735: Opening ipv6 listen socket on port 1885.
1715605735: mosquitto version 2.0.11 running
1715605743: New connection from 212.230.233.50:61217 on port 1885.
1715605743: New client connected from 212.230.233.50:61217 as auto-D83DB684-E7C2
-AB0C-386D-0DFA33498704 (p2, c1, k60, u'cliente').
1715605765: New connection from 80.224.197.80:49864 on port 1885.
1715605765: New client connected from 80.224.197.80:49864 as auto-228F644C-4C73-
B3EE-8C85-B6375232400A (p2, c1, k60, u'cliente').
1715605765: Client auto-228F644C-4C73-B3EE-8C85-B6375232400A disconnected.
```

Figura 5.17: Log para el análisis de las credenciales

Como podemos comprobar, en los logs, se nos muestra el usuario cliente que hemos configurado anteriormente.

Aquí tenemos un resumen de la información adicional que nos proporciona el servidor en el log.

(p2, c1, k60, u'cliente'): Esta parte del mensaje proporciona información adicional sobre la configuración del cliente en el servidor MQTT.

- `p2`: Indica que el cliente está utilizando el protocolo MQTT versión 3.1.1.
- `c1`: Indica que el cliente es el primero en conectarse con este identificador de cliente.
- `k60`: Indica que el servidor mantendrá la conexión del cliente durante 60 segundos si no recibe ninguna comunicación del cliente.
- `u'cliente'`: Usuario cliente especificado por el cliente al conectarse al servidor MQTT. Este usuario deberá estar en el fichero de configuración de credenciales /etc/mosquitto/passwf.

Vamos a ejecutar el comando **sin las credenciales** a ver que sucede. Como observamos en la siguiente figura 5.18 nos rechaza la conexión y en log nos avisa que se ha conectado un cliente sin autorización.

Figura 5.18: Conexión rechazada. Sin autorización

5.4. Implementación ACLs MQTT

Ahora que ya hemos implementado el protocolo MQTT con credenciales, implementaremos las ACLs. Como hemos mencionado anteriormente, para implementar las ACLs tenemos que tener implementado anteriormente las credenciales, ya que sino no vamos a poder implementar estas.

En el fichero de configuración mosquitto.conf, tendremos que agregar la siguiente línea **acl_file /etc/mosquitto/acl**. También tendremos que agregar una 2 línea, que será **per_listener_settings false**. Si esta opción estuviera en en true, podríamos personalizar que cada oyente utilice un puerto, pero nuestro caso queremos que todos los oyentes utilicen el mismo puerto.

Establece la ruta a un archivo de lista de control de acceso (ACL). Si se define, el contenido del archivo se utiliza para controlar el acceso de los clientes a los temas en el broker MQTT.

Si este parámetro está definido, entonces solo los temas enumerados tendrán acceso. El acceso a los temas se agrega con líneas en el siguiente formato:

```
topic [read|write|readwrite|deny] <topic>
```

El tipo de acceso se controla utilizando `read`, `write`, `readwrite` o `deny`. Este parámetro **es opcional** (a menos que `<topic>` incluya un carácter de espacio), si no se proporciona, entonces el acceso es de lectura/escritura. `<topic>` puede contener los comodines `+` o `#` como en las suscripciones. La opción `deny` se puede usar para denegar explícitamente el acceso a un tema que de otra manera sería concedido por una declaración de lectura/escritura/lectura-escritura más amplia. Cualquier tema `deny` se maneja antes que los temas que conceden acceso de lectura/escritura.

El primer conjunto de temas se aplica a clientes anónimos, suponiendo que allow_anonymous

sea verdadero. Las ACL específicas de usuario se agregan después de una línea de usuario de la siguiente manera:

```
user <username>
```

El nombre de usuario al que se hace referencia aquí es el mismo que en password_file.

También es posible definir ACLs basadas en la sustitución de patrones dentro del tema. La forma es la misma que para la palabra clave de tema, pero usando "pattern" como la palabra clave.

```
pattern [read|write|readwrite|deny] <topic>
```

Los patrones disponibles para la sustitución son:

- `%c` para hacer coincidir el ID de cliente del cliente.
- `%u` para hacer coincidir el nombre de usuario del cliente.

Ejemplo: Queremos que el usuario cliente tenga permiso solo a sus datos del sensor.

```
pattern write sensor/%u/data
```

Quedaría de esta forma: `pattern write sensor/cliente/data`, entonces cliente solo podrá publicar en el topic data o cualquiera que descienda de este.

El patrón de sustitución debe ser el único texto para ese nivel de jerarquía. Las ACL de patrón se aplican a todos los usuarios incluso si se ha dado la palabra clave "user" previamente.

Ejemplo:

user cliente

```
pattern write sensor/%u/data
```

En este caso nos referimos a lectura cuando nos suscribimos y escritura cuando publicamos.

A nosotros se nos ha pedido el siguiente esquema:

1. Denegar todos los usuarios de escritura y lectura.
2. Que haya un usuario que tenga acceso a todos los temas y variables del sistema.
3. Que haya un usuario que pueda escribir y leer de un especifico tema, en este caso será IOTDEVICES, que será el tema el cual publicaremos en el código de la empresa donde tenemos que hacer la securización.

Esta información la hemos extraído del manual de mosquitto [8].

Nuestro fichero mosquitto.conf se vería como el código de la siguiente figura 5.19.

```
  GNU nano 5.4              /etc/mosquitto/mosquitto.conf *
per_listener_settings false
allow_anonymous false

listener 1885
password_file /etc/mosquitto/passwf
acl_file /etc/mosquitto/acl

persistence true
persistence_location /var/lib/mosquitto/
```

Figura 5.19: Configuración fichero mosquitto.conf con ACLs

Ahora editaremos nuestro fichero acl para que cumpla con los requisitos demandados.

Para que haya un usuario que tenga acceso a todos los temas y variables del sistema, creamos un usuario llamado root y le otorgaremos estos permisos. En la siguiente figura 5.20 podemos ver dichos requisitos nombrados anteriormente en el fichero acl.

```
  GNU nano 5.4                              acl *
#Reglas de acceso para todos, denegar a todos
pattern deny #

#Permitir a ciertos usuarios

#Permitir al usuario servidor acceso a todo
user root
topic readwrite #

#Permitir al usuario cliente solo al topic IOTDEVICES
user cliente
topic readwrite IOTDEVICES
```

Figura 5.20: Configuración fichero acl

Ahora probaremos la implementación de las ACLs. Procederemos a coger el usuario cliente y haremos algunas pruebas para probar el correcto funcionamiento de estas.

Haremos 2 pruebas y serán las siguientes:

Prueba 1:

1. Suscribirnos desde el servidor al servidor al topic IOT con el usuario root.

2. Publicar desde la Raspberry pi 4 al servidor al topic IOT con el usuario cliente.

Como observamos en la siguiente figura 5.21, hemos publicado con el usuario cliente en el topic IOT, pero root que estaba escuchando no ha recibido nada. Esto se debe a que cliente solo tiene permiso de escritura y lectura en el topic IOTDEVICES.

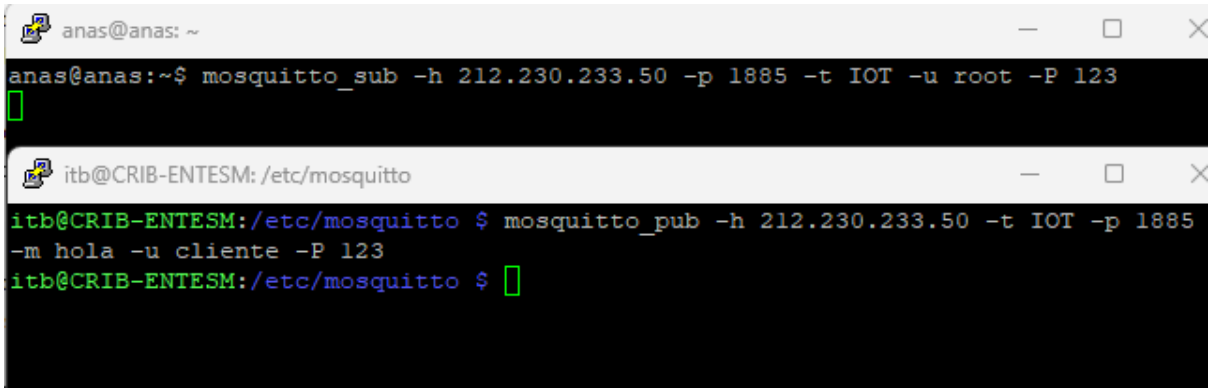


Figura 5.21: Prueba ACLs con tema IOT

La única diferencia entre la prueba 1 y 2 es el tema.

Prueba 2:

1. Suscribirnos desde el servidor al servidor al topic IOTDEVICES con el usuario root.
2. Publicar desde la Raspberry pi 4 al servidor al topic IOTDEVICES con el usuario cliente.

   En la figura 5.22, vemos que ahora el usuario root que estaba suscrito al tema IOTDEVICES, si que ha recibido la publicación del usuario cliente. Esto se debe ya que root tiene permisos permisos de lectura y escritura para cualquier tema, en cambio cliente solo tiene permisos al tema IOTDEVICES.

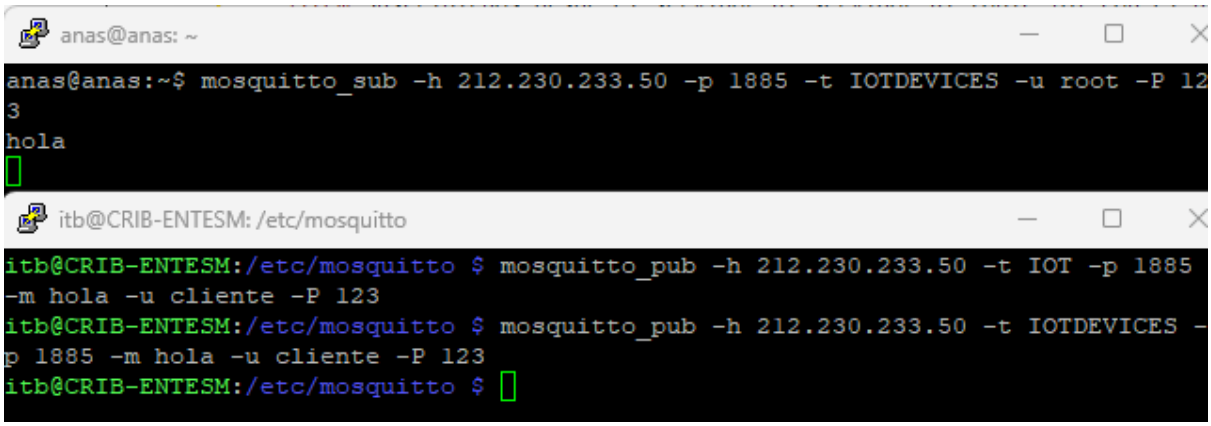


Figura 5.22: Prueba ACLs con tema IOTDEVICES

Antes de comenzar con la implementación de los certificados para TLS/SSL, vamos a mostrar un paquete. Para esto utilizaremos una herramienta sniffer llamada wireshark. Wireshark es

una popular herramienta de análisis de paquetes de código abierto. Filtraremos en la captura de paquetes, por el puerto 1885/TCP.

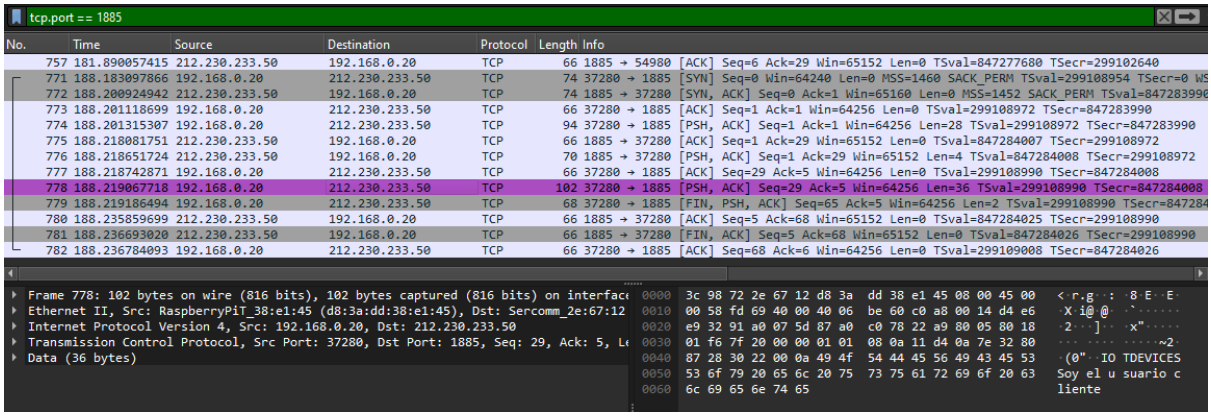


Figura 5.23: Paquete wireshark

Como observamos en la figura 5.23, si vamos a la línea 778 del wireshark y nos fijamos en el contenido del paquete, podremos leer con facilidad el contenido de estos paquetes. Este es el mensaje que mando la Raspberry pi 4 hacia el servidor MQTT. Por esto motivo, surge la necesidad de la securización del protocolo MQTT.

5.5. Implementación certificados para TLS/SSL

Ahora que hemos implemento el protocolo MQTT con las credenciales y las ACLs, procederemos a implementar TLS/SSL para el cifrado del contenido de los paquetes MQTT, ya que como vimos en la figura 5.23 podemos leer fácilmente estos paquetes.

Certificados en el servidor MQTT

Vamos a explicar cómo realizar TLS/SSL en MQTT, lo que sería el MQTTS.

Para realizar esto, lo primero que tendremos que hacer es generar los certificados en el broker y más adelante en el cliente.

Introduciremos una serie de comandos para generar estos.

Primero de todo instalaremos openssl:

```
sudo apt-get install openssl
```

Luego entramos al directorio donde deseemos crear los certificados, en nuestro caso, como estamos securizando el protocolo MQTT y estamos usando el servidor MQTT mosquitto, en-

traremos a `/etc/mosquitto`. La propia instalación de mosquitto nos crea 2 carpetas en dicha ruta, una llamada `ca_certificates` y otra `certs`.

En `ca_certificates` se suelen guardar los certificados de la Autoridad de Certificación (Certificate Authority, CA). Aquí podremos guardar el certificado llamado `ca.key` y `ca.crt`.

En `certs` guardaremos los certificados del broker.

No obstante, no importa la ubicación de los certificados, siempre que estén dentro de /etc/-mosquitto. Yo los pondré todos juntos en `/etc/mosquitto/certs` para que sea más cómodo a la hora de definir las rutas en el fichero de configuración `mosquitto.conf`.

La `ca.key` contiene la clave privada de la Autoridad de Certificación. La clave privada se utiliza para firmar los certificados de otras entidades, como el certificado del mosquitto MQTT `ca.crt` que enviaremos al cliente, en este caso, a la Raspberry Pi 4.

Comando para generar la clave privada:

```
openssl genrsa -des3 -out ca.key 2048
```

- `openssl genrsa`: Es el comando para generar una clave privada RSA.
- `-des3`: Especifica que se utilizará cifrado DES3 para proteger la clave privada. Esto significa que se solicitará una contraseña para proteger la clave privada.
- `-out ca.key`: Especifica el nombre del archivo de salida que contendrá la clave privada de la Autoridad de Certificación (CA).
- `2048`: Especifica el tamaño de la clave RSA, en este caso, 2048 bits. La longitud de la clave, en este caso 2048 bits, determina la seguridad de la clave privada RSA. Cuantos más bits tenga la clave, más difícil será para un atacante descifrarla mediante técnicas de fuerza bruta. Los 2048 bits se consideran actualmente seguros para la mayoría de las aplicaciones, ofreciendo un equilibrio adecuado entre seguridad y rendimiento.

Comando para generar la clave pública:

```
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

- `openssl req`: Es el comando para generar una solicitud de certificado y realizar operaciones relacionadas con las solicitudes de certificado.
- `-new`: Indica que se está generando una nueva solicitud de certificado.
- `-x509`: Especifica que se está creando un certificado autofirmado. (Un certificado autofirmado es un certificado digital donde la entidad que emite el certificado también es la que lo firma. En otras palabras, la entidad certificadora y el propietario del certificado son la misma entidad.)

- `-days 1826`: Especifica la validez del certificado en días. En este caso, 1826 días (aproximadamente 5 años). Este parámetro puede estar sujeto a los años de contratación de los servicios de la empresa.
- `-key ca.key`: Especifica la clave privada que se utilizará para firmar el certificado. En este caso, la clave privada de la CA.
- `-out ca.crt`: Especifica el nombre del archivo de salida que contendrá el certificado autofirmado de la CA.

El certificado `ca.crt` se usará para los clientes.

Generamos una clave privada RSA que el servidor MQTT utilizará en su propia configuración

```
openssl genrsa -out server.key 2048
```

No se utiliza cifrado DES3 en este caso, por lo que no se solicitará una contraseña al generar la clave privada.

Ahora crearemos una solicitud de certificado csr con la clave privada RSA.

```
openssl req -new -out server.csr -key server.key
```

Ahora utilizaremos el certificado y la contraseña para verificar y firmar el certificado del servidor MQTT.

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key
-CAcreateserial -out server.crt -days 360
```

- `openssl x509`: Es el comando utilizado para firmar y manejar certificados X.509.
- `-req`: Indica que se está procesando una solicitud de certificado.
- `-in server.csr`: Especifica el archivo que contiene la solicitud de certificado del broker.
- `-CA ca.crt`: Especifica el certificado de la Autoridad de Certificación (CA) que se utilizará para firmar el certificado del broker.
- `-CAkey ca.key`: Especifica la clave privada de la CA que se utilizará para firmar el certificado del broker.
- `-CAcreateserial`: Crea un archivo de serie para la CA si no existe. Este archivo se utiliza para rastrear el número de serie de los certificados firmados.
- `-out server.crt`: Especifica el nombre del archivo de salida que contendrá el certificado firmado del broker.
- `-days 360`: Especifica la validez del certificado firmado en días. En este caso, el certificado será válido durante 360 días.

Vamos a seguir los pasos y generar los certificados.

Generamos la clave privada, nos pedirá un contraseña y la introducimos se nos habrá creado el fichero ca.key como vemos en la siguiente figura 5.24.

```
anas@anas:/etc/mosquitto/certs/tls$ sudo openssl genrsa -des3 -out ca.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....
.....+++++
e is 65537 (0x010001)
Enter pass phrase for ca.key:
Verifying - Enter pass phrase for ca.key:
```

Figura 5.24: clave privada

Ahora vamos a generar una solicitud de certificado y realizar operaciones relacionadas con las solicitudes de certificado como observamos en la siguiente figura 5.25.

```
anas@anas:/etc/mosquitto/certs/tls$ sudo  openssl req -new -x509 -days 1825 -key
 ca.key -out ca.crt
Enter pass phrase for ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Castellon
Locality Name (eg, city) []:Castellon de la plana
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Empresa Juan
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:212.230.233.50
Email Address []:
```

Figura 5.25: Solicitud de certificado

Nos pedirá la contraseña que hemos introducido anteriormente. Rellenamos los campos que nos pide. Hay que estar atentos al Common Name, tenemos que poner una IP que se pueda resolver. Esto es muy importante ya que si introducimos una IP o el nombre del dominio mal, no nos funcionará. En este caso ponemos la IP publica del servidor.

Generamos una **clave privada RSA**. Además del cifrar las comunicaciones, la clave privada se utiliza para firmar digitalmente los mensajes enviados desde el servidor MQTT. Esto permite a los clientes verificar la autenticidad e integridad de los mensajes recibidos, lo que ayuda a prevenir la manipulación de los datos en tránsito. En la figura 5.26 se muestra el comando para realizar este paso.

```
anas@anas:/etc/mosquitto/certs/tls$ sudo  openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
..............+++++
.....................................................................
+++++
e is 65537 (0x010001)
```

Figura 5.26: Clave privada RSA

Crearemos una solicitud de certificado csr con la clave privada RSA. Igual que hemos hecho anteriormente, nos pedirá rellenar algunas campos, es importante fijarse bien que el campo de Common Name este bien puesto. En la siguiente figura 5.27 observamos los campos a rellenar.

```
anas@anas:/etc/mosquitto/certs/tls$ sudo  openssl req -new -out server.csr -key
server.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Castellon
Locality Name (eg, city) []:Castellon de la plana
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ITB
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:212.230.233.50
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123
string is too short, it needs to be at least 4 bytes long
A challenge password []:ITB%123
An optional company name []:
```

Figura 5.27: Solicitud de certificado csr

Ahora utilizaremos el certificado y la contraseña para verificar y firmar el certificado del servidor MQTT. En la figura 5.28 se muestra como se realiza este paso.

```
anas@anas:/etc/mosquitto/certs/tls$ sudo  openssl x509 -req -in server.csr -CA c
a.crt -CAkey ca.key -CAcreateserial -out server.crt -days 360
Signature ok
subject=C = ES, ST = Castellon, L = Castellon de la plana, O = ITB, CN = 212.230
.233.50
Getting CA Private Key
Enter pass phrase for ca.key:
```

Figura 5.28: Verificar y firmar el certificado del servidor MQTT

Estos son los ficheros que tendríamos que tener al final de ejecutar los comandos anteriores. En la siguiente figura 5.29 se muestran cuales son.

```
anas@anas:/etc/mosquitto/certs/tls$ ls
ca.crt  ca.key  ca.srl  server.crt  server.csr  server.key
```

Figura 5.29: Lista ficheros

Ahora necesitamos transferir el ca.crt a la Raspberry pi 4. Lo hacemos con el comando scp. Ya que cuando la Raspberry pi 4 publique, necesitará este certificado. En la siguiente figura 5.30 vemos que se ha transferido correctamente.

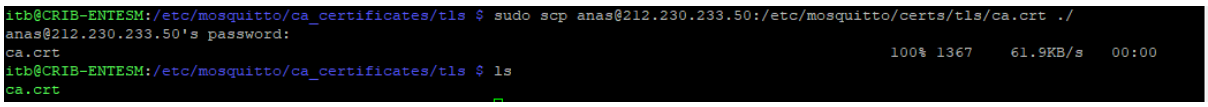


Figura 5.30: Transferir ca.crt a Raspberry pi 4

Con estos pasos hemos obtenido diferentes ficheros pero solo necesitaremos 3 de estos para implementar TLS/SSL en el protocolo MQTT y son los siguientes:

1. **ca.crt:** Este archivo es la certificación de la Autoridad de Certificación (CA). Debemos transferir este archivo a la Raspberry Pi 4 para permitirle publicar mensajes en el servidor MQTT de forma segura.

2. **server.crt y server.key:** Estos archivos son el certificado del servidor y su clave privada, respectivamente. Deben ser configurados en el servidor MQTT para permitir conexiones seguras desde clientes remotos, en este caso la Raspberry Pi 4.

Para ver el **contenido de un certificado** en texto, porque están codificados en base64, usar:

```
openssl x509 -in server.crt -text -noout
```

En la siguiente figura 5.31 se muestra dicho contenido.

```
anas@anas:/etc/mosquitto/certs/tls$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            62:70:9f:0b:d1:36:96:fb:b7:3b:b8:dc:7c:00:12:e2:86:8a:ba:85
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = ES, ST = Castellon, L = Castellon de la plana, O = Empresa J
uan, CN = 212.230.233.50
        Validity
            Not Before: May 14 01:18:35 2024 GMT
            Not After : May  9 01:18:35 2025 GMT
        Subject: C = ES, ST = Castellon, L = Castellon de la plana, O = ITB, CN
= 212.230.233.50
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d2:fe:1c:a5:ff:e4:86:83:98:dd:7d:c9:5a:31:
                    12:b4:65:4b:65:e9:29:d0:8e:ce:40:e1:58:56:00:
                    1e:b2:34:b6:44:55:80:6a:4f:08:58:52:66:48:87:
                    8b:66:ec:9c:02:92:e3:1f:b1:d9:24:fc:86:89:80:
                    44:d5:dc:66:4d:de:74:6a:91:3a:a6:33:3f:12:e3:
                    9d:b6:eb:56:da:bf:65:ee:38:71:c3:ef:ca:54:bf:
                    79:7f:9f:27:d3:e4:9e:6e:00:7c:c4:c0:06:ce:a6:
                    70:81:7d:61:f1:2a:03:29:74:8d:f3:7d:70:e5:4d:
                    44:8d:f6:26:73:3e:6b:3b:e0:7f:80:ae:dd:14:d4:
                    38:4b:6c:97:01:c5:c8:f1:7f:ac:14:17:f9:55:a9:
                    b4:49:fa:d1:9d:49:d3:6c:72:65:ff:33:d8:04:5b:
                    73:00:00:78:1b:cf:dc:19:ec:b0:a3:98:6a:f4:73:
                    1d:3a:1f:27:92:38:64:aa:10:c2:1a:6f:17:9f:13:
                    f8:32:23:04:8d:53:c0:ce:c0:fe:6d:7f:6e:d6:1b:
                    e1:76:10:45:ef:32:62:4c:ac:eb:24:ab:bf:dd:af:
                    bf:1f:78:22:94:da:7e:00:46:55:d6:f4:4d:fe:73:
                    0e:d7:40:4a:17:cb:ce:56:f1:d6:9c:dd:b8:6f:9b:
                    45:e7
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         a6:15:77:52:95:af:7d:f0:93:18:29:a5:12:8d:c1:31:4f:66:
         1d:6b:29:45:50:aa:42:ab:11:d1:f4:8d:e0:3e:37:0e:e5:09:
         b9:83:32:42:7b:a1:60:87:ee:ab:f8:c5:d6:66:d4:e2:b1:5a:
         9a:b0:0a:0e:2b:7d:0f:ea:3d:ac:a7:32:67:14:00:70:ea:e7:
         19:ad:47:cf:53:c3:af:7d:b7:09:8d:c9:fe:42:2f:92:13:26:
         90:4c:6a:6b:da:09:d6:79:54:30:0f:05:ec:18:fd:7a:56:0a:
         f2:17:33:24:38:c3:1b:a2:c7:e4:15:30:d4:38:6a:52:0a:26:
         7c:df:3b:8a:3a:ae:6a:cb:fc:33:17:a5:c7:af:1f:56:70:f9:
         b5:00:1a:bd:b7:22:c7:22:d4:be:e0:36:fd:ed:1d:03:f9:b8:
         81:9d:85:03:32:9f:b0:74:78:31:a9:ad:fe:d6:8c:b9:8a:f0:
         5f:93:59:05:a6:81:8d:81:77:d9:4f:fd:9e:a0:f4:0c:85:c4:
         94:bc:2f:1b:5d:68:1c:94:b6:e7:81:18:89:c5:7f:cb:79:d7:
         96:2c:2a:cf:97:74:95:c3:6b:86:2a:83:ac:e1:cf:ae:38:35:
         23:99:c5:b3:5c:89:4b:1f:26:f8:2a:e2:9e:6b:04:93:87:a1:
         c0:49:e3:7a
```

Figura 5.31: Contenido del certificado server.crt

Como observamos en la siguiente figura 5.32 este problema se nos puede presentar si no

damos los permisos suficientes al usuario mosquitto y a los ficheros que acabamos de generar.

```
anas@anas:~$ /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
1715650494: mosquitto version 2.0.11 starting
1715650494: Config loaded from /etc/mosquitto/mosquitto.conf.
1715650494: Warning: ACL pattern '#' does not contain '%c' or '%u'.
1715650494: Opening ipv4 listen socket on port 8883.
1715650494: Opening ipv6 listen socket on port 8883.
1715650494: Error: Unable to load server key file "/etc/mosquitto/certs/tls/serv
er.key". Check keyfile.
1715650494: OpenSSL Error[0]: error:0200100D:system library:fopen:Permission den
ied
1715650494: OpenSSL Error[1]: error:20074002:BIO routines:file_ctrl:system lib
1715650494: OpenSSL Error[2]: error:140B0002:SSL routines:SSL_CTX_use_PrivateKey
_file:system lib
```

Figura 5.32: Error certificados

Para realizar esto ejecutaremos el comando para darle permisos al usuario mosquitto:

```
sudo chown –R mosquitto: /etc/mosquitto
```

Ahora que ya tenemos todo listo, nos dirigimos hacia el fichero de configuración **/etc/mosquitto/mosquitto.conf**, y añadiremos algunas lineas para que use el puerto 8883 (este puerto se usa para la securización en MQTT), poner que versión de TLS estaremos usando y finalmente los certificados. En la figura 5.33, podemos ver dicha configuración.

```
  GNU nano 5.4                    mosquitto.conf *
per_listener_settings false
allow_anonymous false

listener 8883

password_file /etc/mosquitto/passwf
acl_file /etc/mosquitto/acl
persistence true
persistence_location /var/lib/mosquitto/
cafile /etc/mosquitto/certs/tls/ca.crt
certfile /etc/mosquitto/certs/tls/server.crt
keyfile /etc/mosquitto/certs/tls/server.key
tls_version tlsv1.3
```

Figura 5.33: Fichero de configuración mosquitto.conf con certificados

Ahora vamos a hacer las pruebas y capturaremos un paquete con el wireshark para ver si el contenido esta cifrado.

1. Nos suscribiremos desde el servidor MQTT al tema “IOTDEVICES”.

2. Publicaremos desde la Raspberry pi 4 al servidor en el tema “IOTDEVICES”.

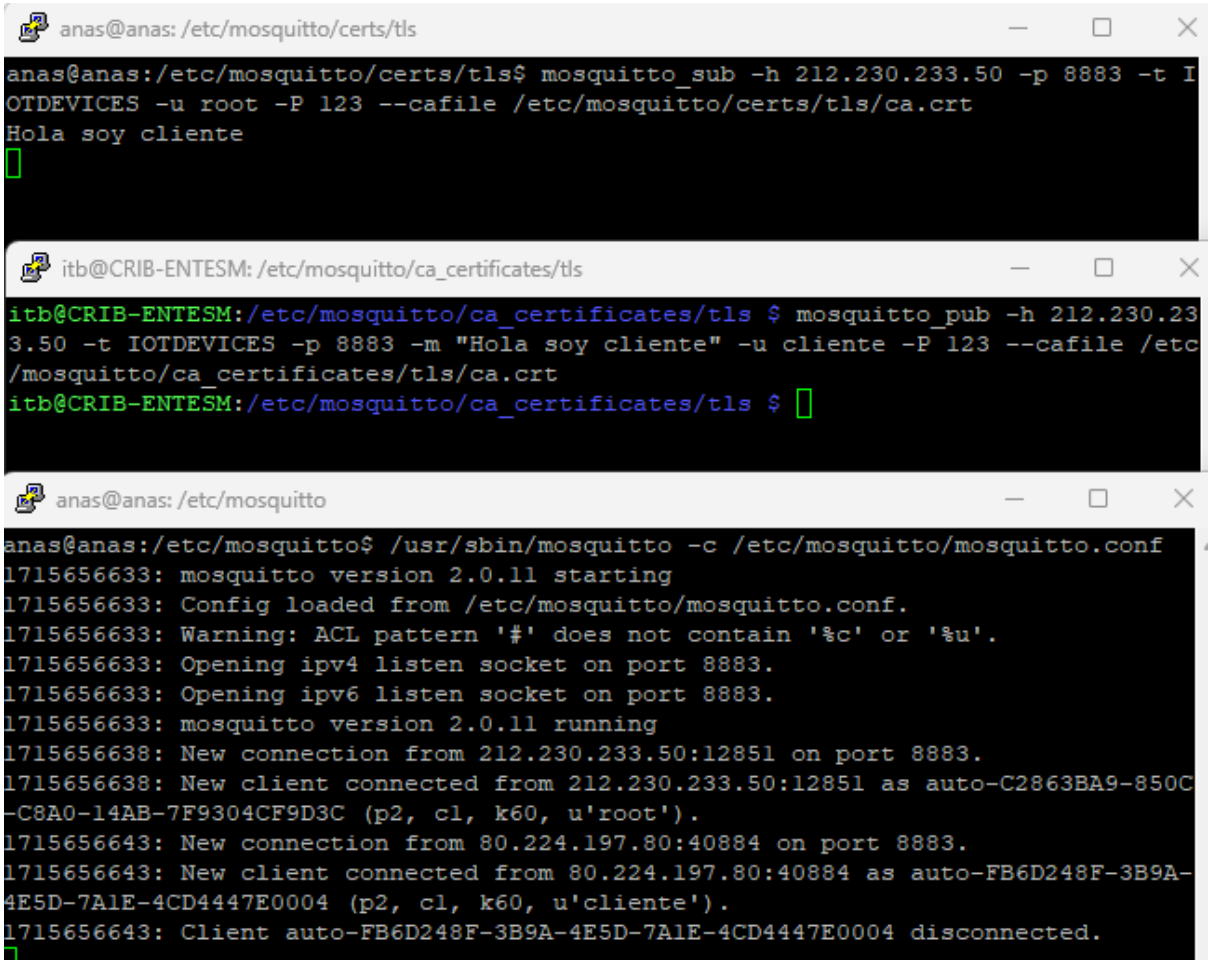


Figura 5.34: Comprobación correcto funcionamiento certificados.

Como podemos observar en la figura 5.34, nos salta un Warning: ACL pattern '#' does not contain '%c' or '%u'. No debemos preocuparnos sobre este warning. Nos muestra este warning ya que los ACLs, cuando dimos permiso a todos los topic al usuario root, no pusimos su nombre '%u', pero al querer que root tenga acceso a todos los temas tal y como pedían, simplemente utilizamos el carácter comodín # para representar todos los temas. El mensaje de advertencia indica que el patrón de control de acceso (ACL) no contiene los marcadores de posición %c o %u, que se utilizan para especificar el nombre de cliente (%c) o el nombre de usuario (%u). Esto significa que el ACL no está configurado para incluir específicamente el nombre del cliente o usuario en el patrón.

Sin embargo, en este caso, **no es necesario preocuparse por el warning**, ya que el objetivo era otorgar acceso a todos los temas al usuario root, y el uso del carácter comodín # logra este propósito sin necesidad de especificar un nombre de cliente o usuario. El mensaje de advertencia simplemente indica que no se están utilizando las opciones de sustitución de usuario o cliente en el patrón del ACL, pero esto no afecta la funcionalidad ni la seguridad de la configuración en este caso particular.

3. Mirar en el wireshark si el contenido del paquete MQTT va cifrado.

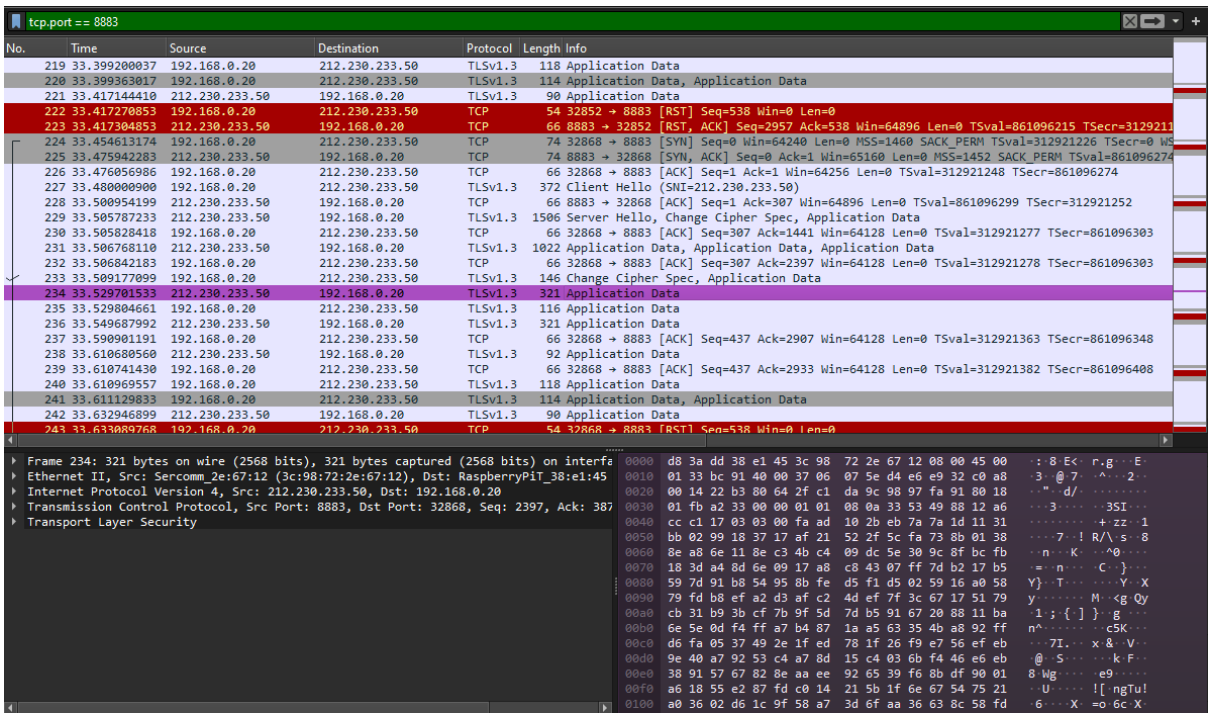


Figura 5.35: Contenido cifrado Wireshark

Si analizamos la figura anterior 5.35, observamos que el protocolo que se esta utilizando para la transferencia de datos en el puerto 8883, es TLS1.3, tal y como hemos especificado en el fichero de configuración mosquitto.conf. Si vemos el contenido del paquete de la línea 234 y de los demás también, vemos que esta completamente cifrado.

Con esto hemos cumplido con la implementación de la securización del protocolo MQTT.

5.6. Implementación VPNs

Antes de adentrarnos en **la implementación de las VPNs**, es importante destacar un aspecto crucial: la securización del protocolo MQTT. Esta medida se está planeando con miras a una expansión significativa en el alcance de la empresa. La securización del protocolo MQTT se considera un paso fundamental en este proceso de expansión.

Es importante tener en cuenta que esta expansión está sujeta a cambios constantes debido a la complejidad y la variedad de variables involucradas en su implementación. Durante mi período de prácticas, recibí información sobre estos cambios de forma regular, lo que me permitió tener una idea general del panorama en evolución.

En una de esas ocasiones, mi supervisor mencionó que se había reunido con uno de los administradores de sistemas, quien desempeñaba un papel crucial en el proyecto. Durante esta reunión, discutieron aspectos importantes relacionados con el proyecto, como los requisitos, las estrategias de implementación y entre ellas la implementación de las VPNs. En esta se tomó la

decisión de externalizar esta tarea, la implantación de las VPNs, a una empresa especializada que ya tenía una relación establecida con la organización.

Aunque inicialmente se había considerado realizar esta **implementación internamente**, la decisión de externalizar el proyecto no necesariamente invalida la idea de **tener un conocimiento interno sobre estas tecnologías**. La implementación de estas VPNs internamente podría haber sido una opción explorada para garantizar una comprensión profunda de las tecnologías involucradas y para tener la capacidad de gestionarlas en el futuro, incluso si la ejecución final del proyecto recae en una **empresa externa**.

Cuando se me informó de esta decisión, ya había avanzado en la configuración de las VPNs WireGuard y OpenVPN, pero aún no se había trabajado en la implementación de IPsec. Sin embargo, durante las discusiones sobre el proyecto, surgieron preocupaciones sobre las características y las posibles complicaciones asociadas con IPsec. Además, la empresa ya había tenido experiencias previas poco favorables con esta tecnología de VPN. Como resultado, se decidió descartar la implementación de IPsec y centrarse en las soluciones de WireGuard y OpenVPN, que parecían más adecuadas para las necesidades y los requisitos del proyecto, así como para evitar posibles obstáculos técnicos y de implementación.

Ahora que ya hemos comentado este aspecto crucial, podemos comenzar con la implementación de las VPNs.

Comenzaremos con WireGuard y mas adelante con OpenVPN.

5.6.1. Implementación VPN WireGuard

Implementación del lado del servidor

Nos descargamos la VPN WireGuard con el siguiente comando:

```
sudo apt-get install wireguard
```

Y implementamos el código de la siguiente figura 5.36.

[Interface]

- **Address:** Esta línea especifica la dirección IP y la máscara de subred que se asignará a la interfaz de red del servidor WireGuard. En este caso, el servidor tendrá la dirección IP 10.0.0.1 con una máscara de subred /24.

- **PostUp y PostDown:** Estos son comandos que se ejecutarán después de que la interfaz WireGuard se active (PostUp) o se desactive (PostDown). En este caso, se están configurando reglas de iptables para el reenvío de paquetes y el enmascaramiento de direcciones IP. Estos comandos específicos permiten el reenvío de paquetes y el enmascaramiento de direcciones IP en la interfaz ens192.

- **ListenPort:** Este es el puerto en el que el servidor WireGuard escuchará las conexiones entrantes. En este caso, el servidor escuchará en el puerto 51194.

- **FwMark:** Este parámetro define la marca de firewall (FwMark) que se aplicará a los paquetes enviados a través de la interfaz WireGuard. En este caso, se establece en 0xca6c.

- **PrivateKey:** Esta es la clave privada del servidor WireGuard. Se utiliza para establecer una conexión segura entre el servidor y los clientes. Es crucial para la autenticación y el cifrado de los datos transmitidos a través de la VPN. La clave privada debe permanecer segura y no debe ser compartida con nadie más.

[**Peer**]

- **PublicKey:** Esta es la clave pública del par (cliente) que se utilizará para establecer la conexión segura. Es necesaria para autenticar el par y cifrar los datos transmitidos.

- **AllowedIPs:** Esta especifica las direcciones IP a las que el servidor WireGuard permitirá que el par (cliente) acceda a través de la conexión VPN. En este caso, el servidor permitirá el acceso a las direcciones de la subred 10.0.0.0/24.

- **Endpoint:** Esta es la dirección IP y el puerto del servidor al que se conectará el par (servidor) para establecer la conexión VPN. En este caso, el cliente se conectará al servidor en la dirección [IP publica de nuestro router] en el puerto 51194.

```
  GNU nano 5.4                          /etc/wireguard/wg0.conf *
[Interface]
Address = 10.0.0.1/24
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o ens192 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o ens192 -j MASQUERADE
ListenPort = 51194
FwMark = 0xca6c
PrivateKey = wMojVlFomp9EegLfS55xXS3ELMQJfmAQOxLampL5Z2E=

[Peer]
PublicKey = c7nxl+K++tb4bqyVpiU6z9tYBV3Ml2Deikg+TFUZvFk=
AllowedIPs = 10.0.0.0/24, 192.168.0.0/16, 192.168.111.12/32
Endpoint = 89.141.172.166:51194
```

Figura 5.36: Fichero de configuración wg0.conf servidor

Cuando tengamos el fichero de configuración listo, activamos la interfaz de la VPN con el siguiente comando `sudo systemctl start wg-quick@wg0`.

Luego ejecutamos el siguiente comando `sudo systemctl status wg-quick@wg0` para ver que se activó la interfaz correctamente.

También podemos comprobarlo con el comando `sudo wg`. Como se ve en la figura 5.37.

```
interface: wg0
  public key: W7UQ2qJuBlbGJB+6FLa7XONeFL1ItvcYO4zZPA0PcCE=
  private key: (hidden)
  listening port: 51194
  fwmark: 0xca6c

peer: c7nxl+K++tb4bqyVpiU6z9tYBV3Ml2Deikg+TFUZvFk=
  endpoint: 89.141.172.166:51194
  allowed ips: 10.0.0.0/24
```

Figura 5.37: Información túnel WireGuard

Otra manera de comprobarlo es con el comando `IP a`. Podemos ver que se nos ha asignado la misma IP que introducimos en el fichero de configuración. En la figura 5.38 podemos comprobarlo.

```
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
    link/none
    inet 10.0.0.1/24 scope global wg0
       valid_lft forever preferred_lft forever
```

Figura 5.38: Interfaz wg0

Implementación del lado del cliente

Nos descargamos la VPN WireGuard.

Una vez lo tengamos instalado, accedemos a la ruta /etc/WireGuard y creamos un fichero llamado wg0.conf. Este fichero contendrá toda la configuración de nuestra VPN. En la figura 5.39 podemos ver dicha configuración.

```
  GNU nano 5.4                          /etc/wireguard/wg0.conf *
[Interface]
PrivateKey = oChsoNUQ4keNi/sDA8iuDaJguyoc4ispz+I2QVf4e0k=
Address = 10.0.0.2/24
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
ListenPort = 51194
DNS = 1.1.1.1

[Peer]
PublicKey = W7UQ2qJuBlbGJB+6FLa7XONeFL1ItvcYO4zZPA0PcCE=
AllowedIPs = 10.0.0.0/24
Endpoint = 212.230.233.50:51194
```

Figura 5.39: Fichero de configuración wg0.conf cliente

Cuando tengamos el fichero de configuración listo, activamos la interfaz de la VPN con el siguiente comando `sudo systemctl start wg-quick@wg0`.

Luego ejecutamos el siguiente comando `sudo systemctl status wg-quick@wg0` para ver que se activó la interfaz correctamente.

También podemos comprobarlo con el comando `sudo wg`. Podemos ver dicha comprobación en la figura 5.40.

```
itb@CRIB-ENTESM:~ $ sudo wg
interface: wg0
  public key: c7nxl+K++tb4bqyVpiU6z9tYBV3Ml2Deikg+TFUZvFk=
  private key: (hidden)
  listening port: 51194

peer: W7UQ2qJuBlbGJB+6FLa7XONeFL1ItvcYO4zZPA0PcCE=
  endpoint: 212.230.233.50:51194
  allowed ips: 10.0.0.0/24
```

Figura 5.40: Interfaz wg0 servidor

Otra manera de comprobarlo es con el comando `IP a`. Podemos ver que se nos ha asignado la misma IP que introducimos en el fichero de configuración. En la figura 5.41 vemos que se nos ha asignado dicha IP.

```
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
    link/none
    inet 10.0.0.2/24 scope global wg0
       valid_lft forever preferred_lft forever
```

Figura 5.41: Salida comando IP a

Con estas configuraciones ya tendríamos conectividad entre ambos extremos. Para comprobarlo haremos un ping desde cualquier extremo, en este caso haré un ping desde el servidor al cliente. En la figura 5.42 podemos ver esto.

```
anas@anas:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=48.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=22.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=21.0 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.9 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=21.6 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 20.903/26.875/48.567/10.856 ms
```

Figura 5.42: Ping

Después de lanzar el comando ping, si volvemos a ejecutar el comando sudo wg, observamos que ya esta recibiendo y mandando paquetes. En la figura 5.43 podemos observar esto.

```
anas@anas:~$ sudo wg
[sudo] password for anas:
interface: wg0
  public key: W7UQ2qJuBlbGJB+6FLa7XONeFL1ItvcYO4zZPA0PcCE=
  private key: (hidden)
  listening port: 51194
  fwmark: 0xca6c

peer: c7nxl+K++tb4bqyVpiU6z9tYBV3M12Deikg+TFUZvFk=
  endpoint: 95.62.173.154:51194
  allowed ips: 10.0.0.0/24
  latest handshake: 19 minutes, 53 seconds ago
  transfer: 1.02 KiB received, 1.14 KiB sent
```

Figura 5.43: Interfaz wg0 cliente

Una vez nos de ping, haremos una prueba rápida para ver el correcto funcionamiento del túnel.

1. Nos suscribimos desde el servidor MQTT al servidor MQTT.

2. Publicamos desde la Raspberry pi 4 hacia el servidor MQTT.

   En la figura 5.44 podemos ver dicha publicación y suscripción.

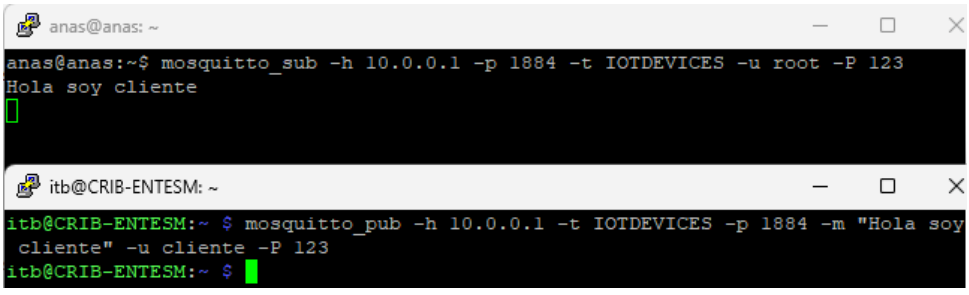


Figura 5.44: Publicación y suscripción MQTT

3. Mirar el log en la figura 5.45.

   Vemos que se nos conectaron las IPs 10.0.0.1 y 10.0.0.2, que son la del servidor y Raspberry pi 4 respectivamente.

```
1715725449: New connection from 10.0.0.1:36532 on port 1884.
1715725449: New client connected from 10.0.0.1:36532 as auto-A27E47B2-EB4D-A660-
1B47-607230C62F50 (p2, c1, k60, u'root').
1715725459: New connection from 10.0.0.2:56644 on port 1884.
1715725459: New client connected from 10.0.0.2:56644 as auto-3D259904-0AAB-BF60-
4674-F4C8E722F78E (p2, c1, k60, u'cliente').
1715725459: Client auto-3D259904-0AAB-BF60-4674-F4C8E722F78E disconnected.
```

Figura 5.45: Log mosquitto MQTT

Ahora procederemos a capturar los paquetes con el wireshark.

Mostraremos 2 interfaces. La interfaz de la tarjeta de red y la del túnel.

Interfaz tarjeta de red eth0

Podemos comprobar en la siguiente figura 5.46 que la información se encuentra encriptada por el túnel de WireGuard. Este protocolo utiliza un mecanismo de cifrado para proteger los datos mientras viajan a través de la red.

El uso de cifrado garantiza que la información transmitida no sea legible para terceros que intenten interceptarla.

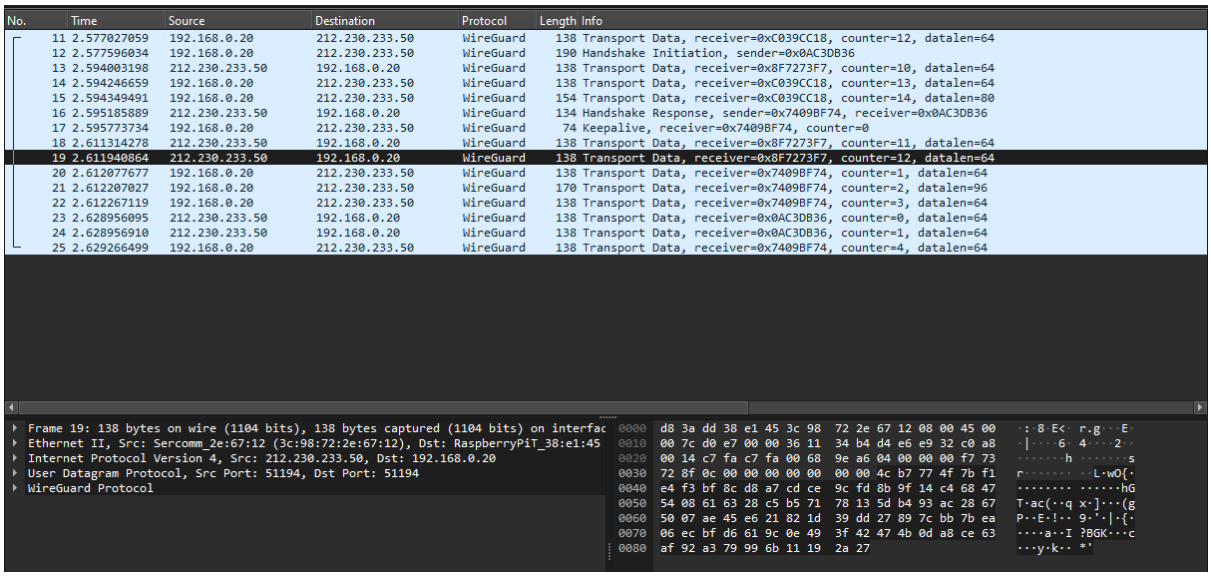


Figura 5.46: Captura de paquete eth0

Interfaz túnel wg0

Si entramos en la interfaz del túnel, podremos ver la información que viaja dentro del paquete en texto plano. En la figura 5.47 podemos ver dicho contenido.

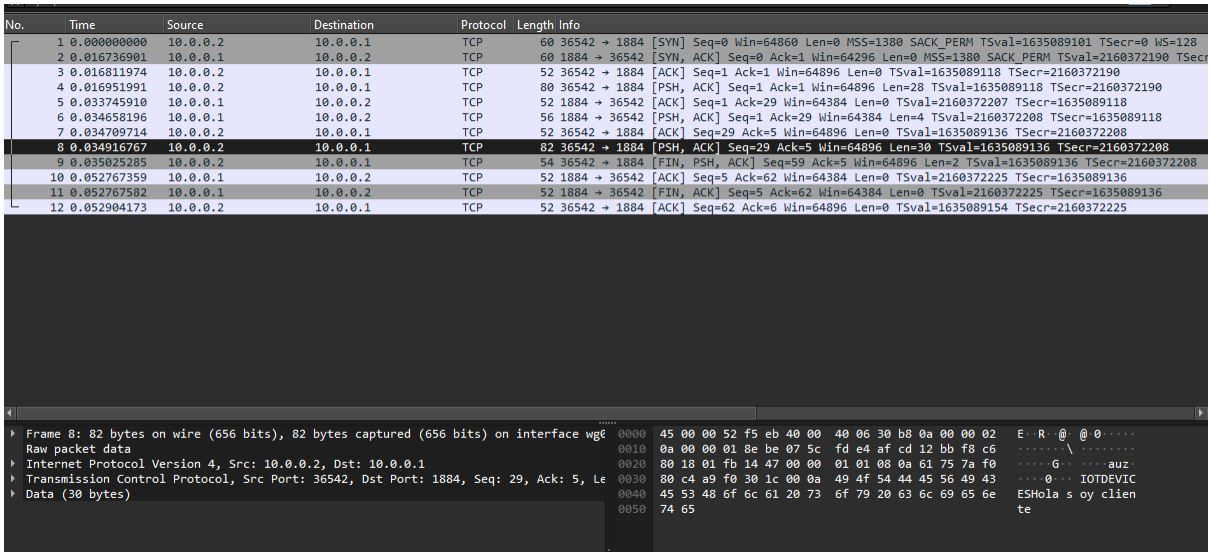


Figura 5.47: Captura de paquete wg0

5.6.2. Implementación VPN OpenVPN

Servidor OpenVPN

OpenVPN ofrece un script llamado .^easy-rsa"que facilita la gestión de certificados y claves necesarios para la configuración de la VPN. Este script automatiza tareas como la generación de certificados de servidor y cliente, la creación de autoridades de certificación (CA), la revocación de certificados, entre otros. Utilizaremos este script para gestionar los certificados y claves requeridos para nuestra implementación de OpenVPN, lo que nos permitirá configurarla de una manera más sencilla.

Nos descargamos el script desde el siguiente repositorio de github: `https://raw.githubusercontent.com/angristan/openvpn-install/master/openvpn-install.sh`.

Creamos un carpeta llamada OpenVPN y ejecutamos dicho comando.

Otorgamos permiso de ejecución y lo ejecutamos.

Cuando lo ejecutemos, nos ira pidiendo configuraciones de la VPN. En la figura 5.48 podemos ver dichos campos.

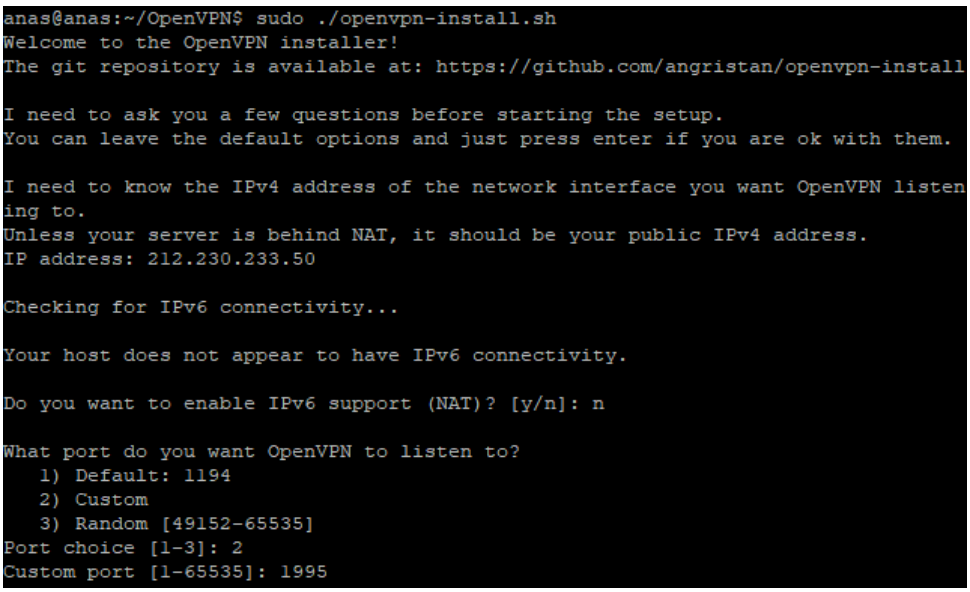


Figura 5.48: Campos a rellenar script OpenVPN

Cuando haya terminado todo el proceso y ejecutamos de nuevo el script nos saldrá el siguiente menú. En la siguiente figura 5.49 podemos ver el menú.

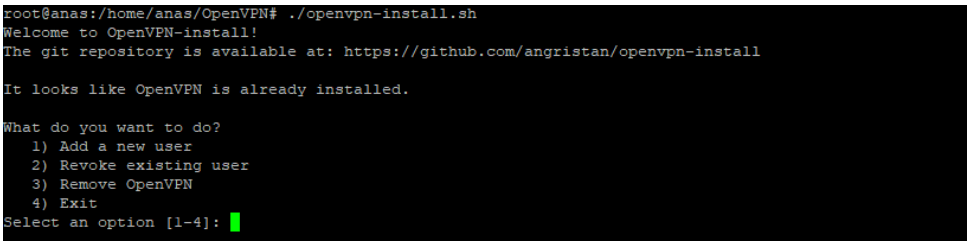


Figura 5.49: Menú OpenVPN

Marcamos el numero 1 para generar un certificado cliente para la Raspberry pi 4. En la siguiente figura 5.50 vemos como crear un cliente.

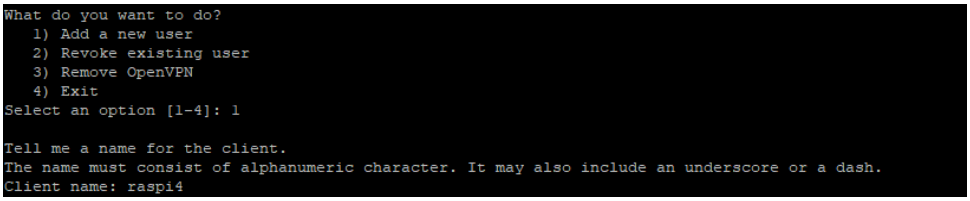


Figura 5.50: Creación cliente OpenVPN

Y se guardara en la ruta /home/anas/raspi4.ovpn. Este certificado tendremos que pasarlo a la Raspberry pi 4.

El certificado que hemos creado anteriormente se encuentra en la ruta /home/anas. Ahora

debemos de transferir este fichero de configuración de OpenVPN.

Una vez lo tengamos en la Raspberry pi 4, lo movemos a /etc/openvpn/client/raspi4.ovpn. Una vez lo tengamos en esa ruta, lo ejecutamos con el comando sudo openvpn raspi4.ovpn, y si queremos que se ejecute en 2 plano agregamos una &. Seria sudo openvpn raspi4.ovpn &.

Cuando lo ejecutemos nos saldrá, entre muchas cosas, la IP que nos ha asignado en el túnel y la interfaz, en este caso, 10.8.0.2 y tun0 respectivamente. En la siguiente imagen 5.51 podemos ver estas asignaciones.

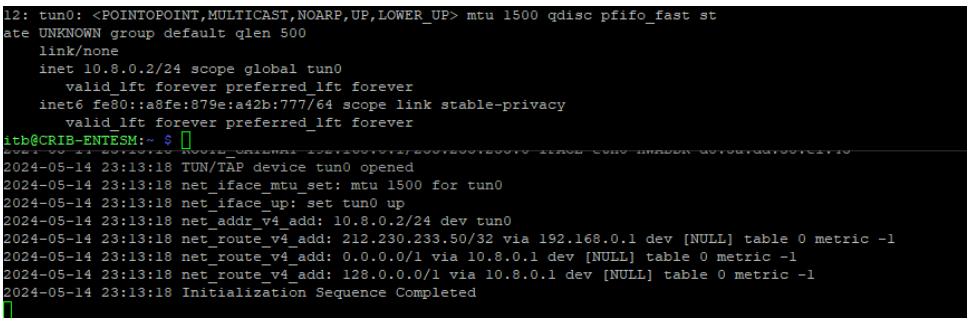


Figura 5.51: Asignación IP

Una vez nos de ping, haremos una prueba rápida para ver el correcto funcionamiento del túnel.

1. Nos suscribimos desde el servidor MQTT al servidor MQTT.

2. Publicamos desde la Raspberry pi 4 hacia el servidor MQTT. En la siguiente imagen 5.52 podemos ver la prueba de suscripción.

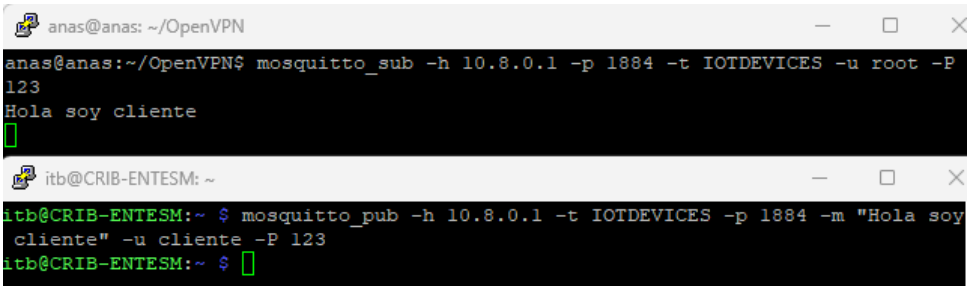


Figura 5.52: Prueba publicación suscripción

3. Mirar el log.

   Vemos que se nos conectaron las IPs 10.8.0.1 y 10.8.0.2, que son la del servidor y Raspberry pi 4 respectivamente. En la imagen 5.53 podemos ver el log.

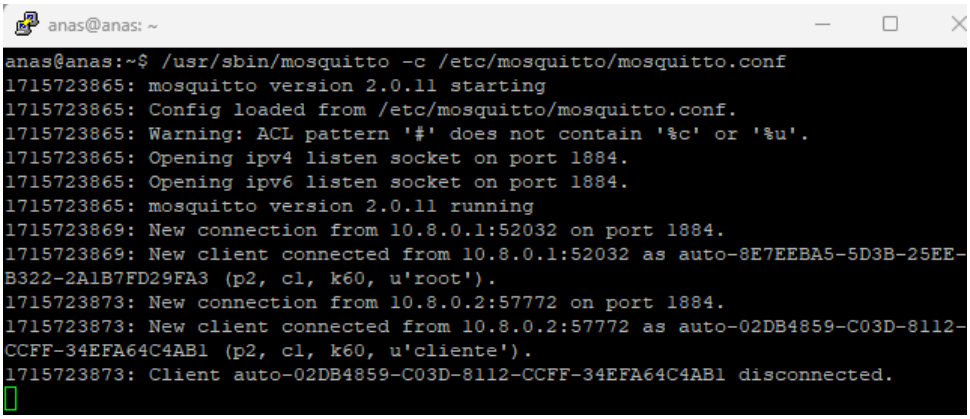


Figura 5.53: Log servidor MQTT

Ahora procederemos a capturar los paquetes con el wireshark.

Mostraremos 2 interfaces. La interfaz de la tarjeta de red y la del túnel.

Interfaz tarjeta de red eth0

Podemos comprobar en la siguiente figura 5.54 que la información se encuentra encriptada por el túnel de OpenVPN.

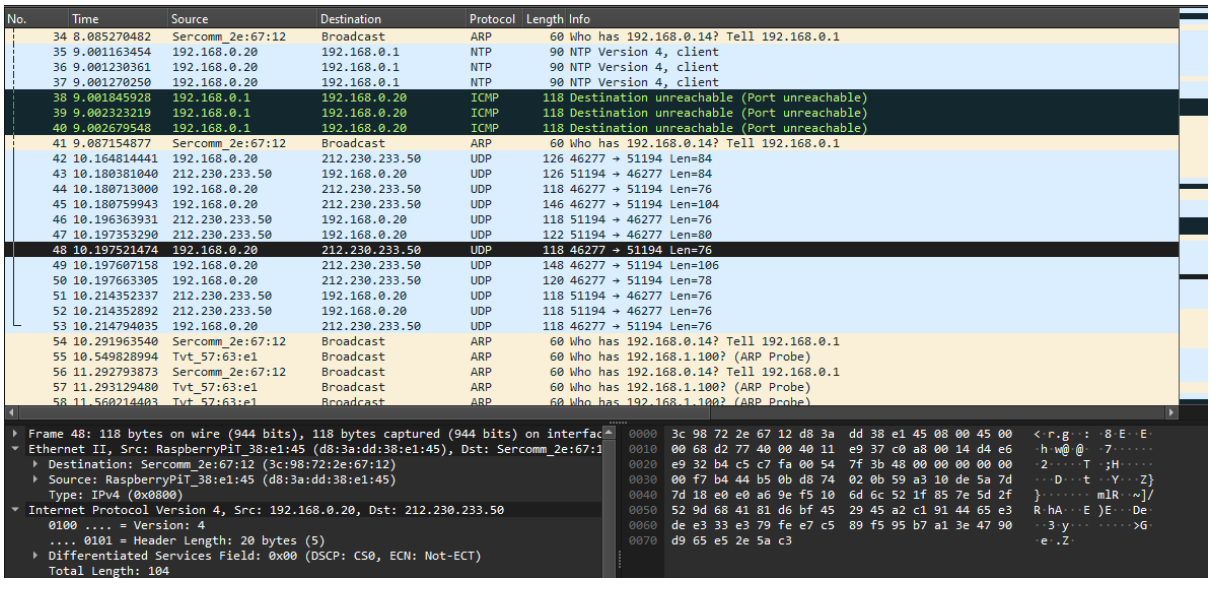


Figura 5.54: Paquete wireshark

Interfaz túnel tun0

Si entramos en la interfaz del túnel, podremos ver la información que viaja dentro del paquete en texto plano. En la imagen 5.55 vemos dicha información.

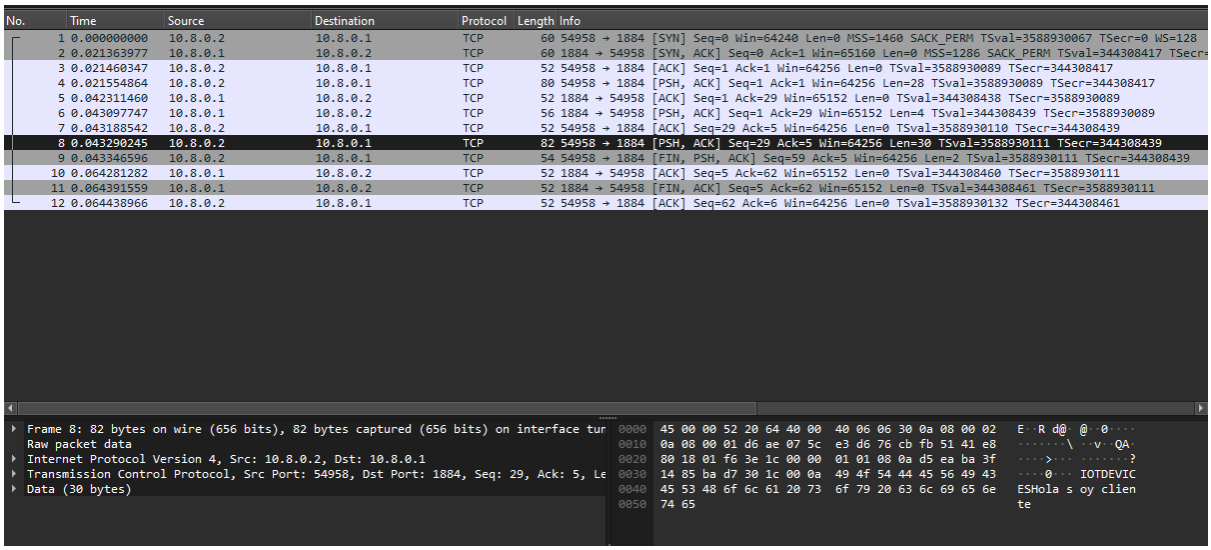


Figura 5.55: Captura de paquete tun0

5.6.3. Pruebas VPN WireGuard y OpenVPN

En esta sección llevaremos a cabo una serie de pruebas con el objetivo de comparar el rendimiento y la eficiencia de las VPNs WireGuard y OpenVPN. Estas pruebas nos permitirán determinar cuál de las dos opciones ofrece un mejor desempeño en términos de velocidad, latencia y estabilidad de la conexión, entre otros aspectos relevantes.

Se realizarán las 2 siguientes pruebas:

- **Prueba 1:** El objetivo de esta prueba es evaluar la latencia y la estabilidad de la conexión entre el cliente y el servidor de la VPN. El comando ping envía 10 paquetes ICMP de tamaño 1000 bytes cada uno al servidor con la dirección IP 10.0.0.2 y registra el tiempo que tarda en recibir una respuesta para cada paquete. Esto nos proporciona información sobre la calidad de la conexión y la consistencia en los tiempos de respuesta.
  Comando: ping -c 100 -s 1000 durante 10 segundos

- **Prueba 2:** El objetivo de esta prueba es medir la velocidad de transferencia de datos entre el cliente y el servidor de la VPN bajo condiciones de carga intensiva. Utilizando el comando iperf3, se establece una conexión con el servidor (dirección_servidor) y se realizarán 10 múltiples instancias de la herramienta iperf, cada una simulando una conexión simultánea. Esto permite evaluar la capacidad de la VPN para manejar un alto volumen de tráfico y determinar la velocidad de transferencia de datos en condiciones de carga intensiva. El comando de la prueba se ejecutará en la Raspberry pi 4 y el servidor estará

escuchando.
Comando: iperf3 -c dirección_servidor -P 10

Para comparar el rendimiento entre OpenVPN y WireGuard, usaremos un tamaño de paquete más pequeño, como 1000 bytes o incluso menos (generalmente el ping se establece en 1500 bytes), para simular condiciones más realistas de la red.

Prueba 1 WireGuard

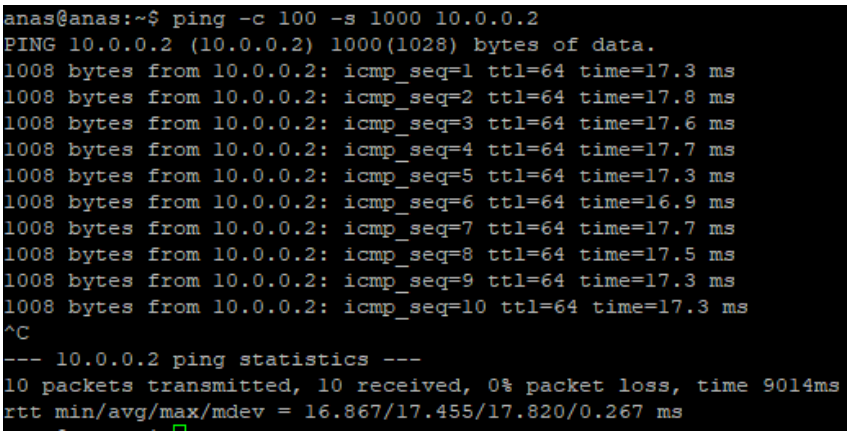


Figura 5.56: Prueba ping WireGuard

Prueba 1 OpenVPN

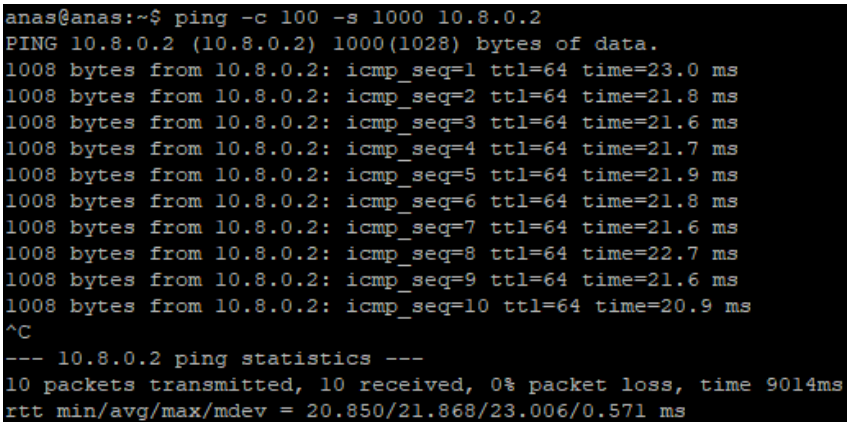


Figura 5.57: Prueba ping OpenVPN

5.6.4. Resultados de Prueba 1 con ping

En las imágenes 5.56 y 5.57 vemos los resultados de la prueba 1.

- Con WireGuard, la latencia mínima fue de 16.8 ms, la latencia promedio fue de 17.455 ms y la latencia máxima fue de 17.820 ms, con una desviación estándar de 0.267 ms.

- Con OpenVPN, la latencia mínima fue de 20.850 ms, la latencia promedio fue de 21.868 ms y la latencia máxima fue de 23.006 ms, con una desviación estándar de 0.571 ms.

Dado que la latencia es el tiempo que tarda un paquete de datos en viajar de un punto a otro en la red, podemos concluir que, en promedio, WireGuard proporcionó una latencia ligeramente mejor que OpenVPN en esta prueba específica. WireGuard mostró valores más bajos en todas las métricas de latencia: mínima, promedio y máxima. Además, la desviación estándar de WireGuard también fue menor, lo que sugiere una mayor consistencia en los tiempos de respuesta.

Por lo tanto, basándonos en esta prueba de latencia, podríamos concluir que WireGuard podría ofrecer un mejor rendimiento en términos de latencia en comparación con OpenVPN en este escenario específico.

Prueba 2 WireGuard

Servidor

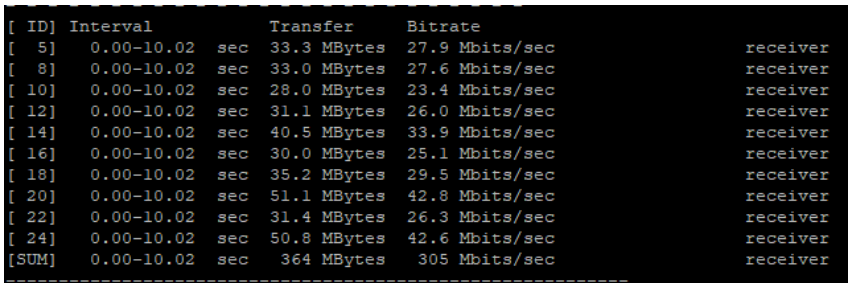
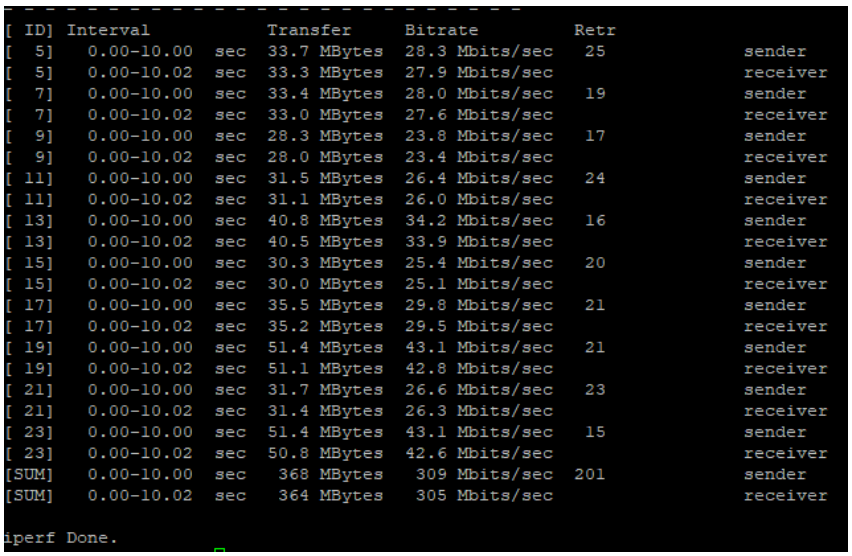


Figura 5.58: Prueba iperf WireGuard servidor

Raspberry pi 4



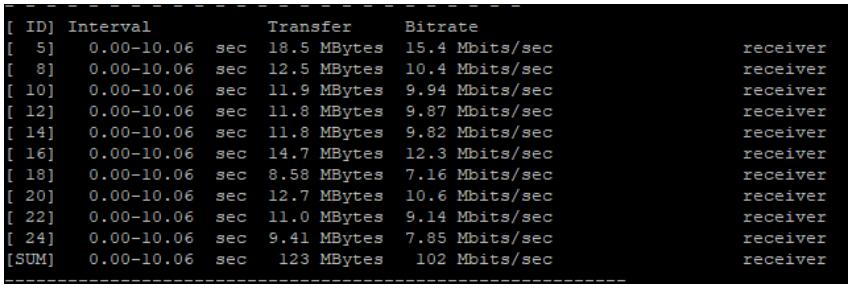
[ID]	Interval		Transfer	Bitrate	Retr	
[5]	0.00-10.00	sec	33.7 MBytes	28.3 Mbits/sec	25	sender
[5]	0.00-10.02	sec	33.3 MBytes	27.9 Mbits/sec		receiver
[7]	0.00-10.00	sec	33.4 MBytes	28.0 Mbits/sec	19	sender
[7]	0.00-10.02	sec	33.0 MBytes	27.6 Mbits/sec		receiver
[9]	0.00-10.00	sec	28.3 MBytes	23.8 Mbits/sec	17	sender
[9]	0.00-10.02	sec	28.0 MBytes	23.4 Mbits/sec		receiver
[11]	0.00-10.00	sec	31.5 MBytes	26.4 Mbits/sec	24	sender
[11]	0.00-10.02	sec	31.1 MBytes	26.0 Mbits/sec		receiver
[13]	0.00-10.00	sec	40.8 MBytes	34.2 Mbits/sec	16	sender
[13]	0.00-10.02	sec	40.5 MBytes	33.9 Mbits/sec		receiver
[15]	0.00-10.00	sec	30.3 MBytes	25.4 Mbits/sec	20	sender
[15]	0.00-10.02	sec	30.0 MBytes	25.1 Mbits/sec		receiver
[17]	0.00-10.00	sec	35.5 MBytes	29.8 Mbits/sec	21	sender
[17]	0.00-10.02	sec	35.2 MBytes	29.5 Mbits/sec		receiver
[19]	0.00-10.00	sec	51.4 MBytes	43.1 Mbits/sec	21	sender
[19]	0.00-10.02	sec	51.1 MBytes	42.8 Mbits/sec		receiver
[21]	0.00-10.00	sec	31.7 MBytes	26.6 Mbits/sec	23	sender
[21]	0.00-10.02	sec	31.4 MBytes	26.3 Mbits/sec		receiver
[23]	0.00-10.00	sec	51.4 MBytes	43.1 Mbits/sec	15	sender
[23]	0.00-10.02	sec	50.8 MBytes	42.6 Mbits/sec		receiver
[SUM]	0.00-10.00	sec	368 MBytes	309 Mbits/sec	201	sender
[SUM]	0.00-10.02	sec	364 MBytes	305 Mbits/sec		receiver

iperf Done.

Figura 5.59: Prueba iperf WireGuard Raspberry pi 4

Prueba 2 OpenVPN

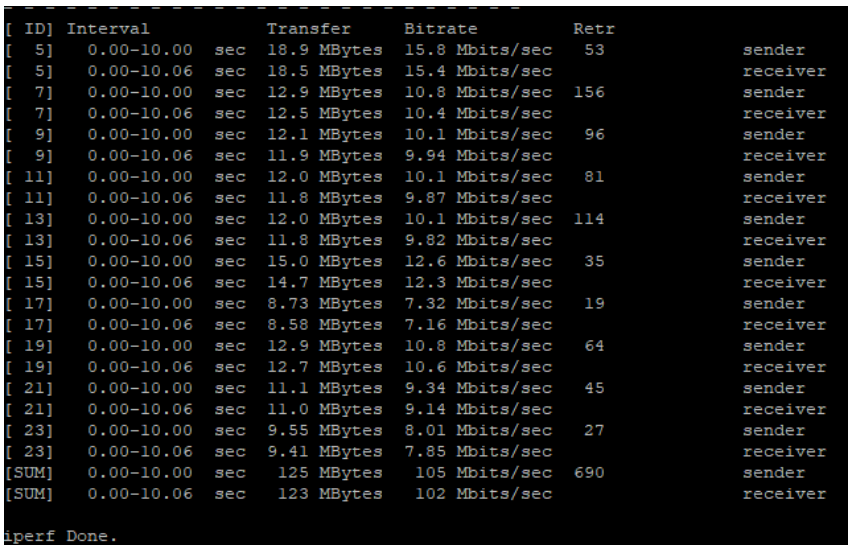
Servidor



[ID]	Interval		Transfer	Bitrate	
[5]	0.00-10.06	sec	18.5 MBytes	15.4 Mbits/sec	receiver
[8]	0.00-10.06	sec	12.5 MBytes	10.4 Mbits/sec	receiver
[10]	0.00-10.06	sec	11.9 MBytes	9.94 Mbits/sec	receiver
[12]	0.00-10.06	sec	11.8 MBytes	9.87 Mbits/sec	receiver
[14]	0.00-10.06	sec	11.8 MBytes	9.82 Mbits/sec	receiver
[16]	0.00-10.06	sec	14.7 MBytes	12.3 Mbits/sec	receiver
[18]	0.00-10.06	sec	8.58 MBytes	7.16 Mbits/sec	receiver
[20]	0.00-10.06	sec	12.7 MBytes	10.6 Mbits/sec	receiver
[22]	0.00-10.06	sec	11.0 MBytes	9.14 Mbits/sec	receiver
[24]	0.00-10.06	sec	9.41 MBytes	7.85 Mbits/sec	receiver
[SUM]	0.00-10.06	sec	123 MBytes	102 Mbits/sec	receiver

Figura 5.60: Prueba iperf OpenVPN servidor

Raspberry pi 4



```
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  18.9 MBytes  15.8 Mbits/sec   53             sender
[  5]   0.00-10.06  sec  18.5 MBytes  15.4 Mbits/sec                  receiver
[  7]   0.00-10.00  sec  12.9 MBytes  10.8 Mbits/sec  156             sender
[  7]   0.00-10.06  sec  12.5 MBytes  10.4 Mbits/sec                  receiver
[  9]   0.00-10.00  sec  12.1 MBytes  10.1 Mbits/sec   96             sender
[  9]   0.00-10.06  sec  11.9 MBytes  9.94 Mbits/sec                  receiver
[ 11]   0.00-10.00  sec  12.0 MBytes  10.1 Mbits/sec   81             sender
[ 11]   0.00-10.06  sec  11.8 MBytes  9.87 Mbits/sec                  receiver
[ 13]   0.00-10.00  sec  12.0 MBytes  10.1 Mbits/sec  114             sender
[ 13]   0.00-10.06  sec  11.8 MBytes  9.82 Mbits/sec                  receiver
[ 15]   0.00-10.00  sec  15.0 MBytes  12.6 Mbits/sec   35             sender
[ 15]   0.00-10.06  sec  14.7 MBytes  12.3 Mbits/sec                  receiver
[ 17]   0.00-10.00  sec  8.73 MBytes  7.32 Mbits/sec   19             sender
[ 17]   0.00-10.06  sec  8.58 MBytes  7.16 Mbits/sec                  receiver
[ 19]   0.00-10.00  sec  12.9 MBytes  10.8 Mbits/sec   64             sender
[ 19]   0.00-10.06  sec  12.7 MBytes  10.6 Mbits/sec                  receiver
[ 21]   0.00-10.00  sec  11.1 MBytes  9.34 Mbits/sec   45             sender
[ 21]   0.00-10.06  sec  11.0 MBytes  9.14 Mbits/sec                  receiver
[ 23]   0.00-10.00  sec  9.55 MBytes  8.01 Mbits/sec   27             sender
[ 23]   0.00-10.06  sec  9.41 MBytes  7.85 Mbits/sec                  receiver
[SUM]   0.00-10.00  sec   125 MBytes   105 Mbits/sec  690             sender
[SUM]   0.00-10.06  sec   123 MBytes   102 Mbits/sec                  receiver

iperf Done.
```

Figura 5.61: Prueba iperf OpenVPN Raspberry pi 4

5.6.5. Resultados de Prueba 2 con iperf3

En las imágenes 5.58, 5.59, 5.60 y 5.61 vemos los resultados de la prueba 2.

WireGuard:

- Con una sola instancia de iperf3 (`-s`), la tasa de transferencia promedio fue de 305 Mbits/-sec.
- Con 10 instancias de iperf3 (`-P 10`), la tasa de transferencia promedio fue de 309 Mbits/-sec.

OpenVPN:

- Con una sola instancia de iperf3 (`-s`), la tasa de transferencia promedio fue de 102 Mbits/-sec.
- Con 10 instancias de iperf3 (`-p 10`), la tasa de transferencia promedio fue de 105 Mbits/-sec.

Resumen:

- WireGuard mostró una tasa de transferencia promedio más alta en ambas configuraciones, tanto con una sola instancia como con 10 instancias de iperf3. La diferencia entre una sola instancia y 10 instancias no fue significativa, lo que sugiere una consistencia en el rendimiento.

- OpenVPN tuvo una tasa de transferencia promedio más baja en comparación con WireGuard en ambas configuraciones. Además, la diferencia entre una sola instancia y 10 instancias de iperf3 fue mínima, lo que también sugiere una consistencia en el rendimiento, pero a un nivel de rendimiento general más bajo que el de WireGuard.

5.6.6. Conclusión resultados VPNs

Las pruebas de ping destacaron la ventaja de WireGuard en términos de latencia y estabilidad de la conexión, mientras que las pruebas de iperf3 resaltaron su superioridad en la transferencia de datos y la capacidad de manejo de cargas de trabajo intensivas. Estos resultados sugieren que WireGuard puede ser la opción preferida para aplicaciones que requieren baja latencia y alta capacidad de transferencia de datos.

5.7. Elección de TLS de solución a implementar

Tras evaluar exhaustivamente las opciones de VPN disponibles y considerando los resultados obtenidos en las pruebas de rendimiento, se ha determinado que tanto WireGuard como OpenVPN ofrecen soluciones viables para las necesidades del proyecto. Sin embargo, teniendo en cuenta que la gestión de las VPNs será llevada a cabo por una empresa externa y considerando la estabilidad, rendimiento y versatilidad de TLS, **se ha tomado la decisión de implementar TLS sobre el código final ya en producción en múltiples empresas**. Esta elección no solo garantiza la seguridad de las comunicaciones, sino que también simplifica la gestión y mantenimiento a largo plazo, asegurando la integridad y confidencialidad de los datos transmitidos.

5.8. Implementación TLS código en producción

5.8.1. Configuración mosquitto servidor

En la siguiente imagen 5.62 observamos la configuración del mosquitto servidor.

```
  GNU nano 5.4              /etc/mosquitto/mosquitto.conf *
per_listener_settings false
allow_anonymous false

listener 8883

password_file /etc/mosquitto/passwf
acl_file /etc/mosquitto/acl
persistence true
persistence_location /var/lib/mosquitto/
cafile /etc/mosquitto/certs/tls/ca.crt
certfile /etc/mosquitto/certs/tls/server.crt
keyfile /etc/mosquitto/certs/tls/server.key
tls_version tlsv1.3
```

Figura 5.62: Configuración mosquitto servidor

5.8.2. Configuración mosquitto cliente

Hemos implementado el bridge, ya que este es necesario para el correcto funcionamiento del código que tienen implementado. Podemos ver dicha implementación en la figura 5.63.

```
  GNU nano 5.4              /etc/mosquitto/mosquitto.conf *
per_listener_settings false
allow_anonymous false
password_file /etc/mosquitto/passwf
acl_file /etc/mosquitto/acl.conf

tls_version tlsv1.3
persistence true
persistence_location /var/lib/mosquitto/


listener 8883

connection bridge-01
address 212.230.233.50:8883
topic IOTDEVICES both 2
cleansession true
remote_username cliente
remote_password 123
bridge_cafile /etc/mosquitto/ca_certificates/ca.crt
cafile /etc/mosquitto/certs/tls/ca.crt
certfile /etc/mosquitto/certs/tls/server.crt
keyfile /etc/mosquitto/certs/tls/server.key
```

Figura 5.63: Configuración mosquitto cliente

La información se va transmitir de la siguiente manera, el cliente va a publicar en su servidor MQTT (este servidor MQTT es del cliente, no del servidor), entonces gracias al bridge, la información que se publique en el cliente, también se publicara en el servidor. Entonces cuando la tenga el servidor, esta se almacenará en una base de datos. Esta será consultado mas tarde para tanto aplicaciones web como móvil.

Ahora editaremos el código que esta actualmente en producción. La parte de los MQTTs esta implementada con Paho MQTT. Paho MQTT es una biblioteca de código abierto que implementa el protocolo MQTT.

Primero editaremos un fichero de parámetros de configuración. En la siguiente figura 5.64 mostramos algunos de estos parámetros.

```
    "topic": "IOTDEVICES",
        "mqtt": {
                "remote_broker": "212.230.230.50",
                "port": 8883,
                "remote_topic": ["IOTDEVICES", "TOPIC-RESET_STOPTIME"]
        }

}
```

Figura 5.64: Fichero de parámetros de configuración

Solo hemos mostrado parte del fichero ya que contiene datos sensibles.

Cuando tengamos el **config.json** bien configurado, procederemos a cambiar el script mqttAgent.py. Aquí es donde agregaremos las credenciales y el TLS/SSL.

Parte del servidor MQTT local (En nuestro caso es la Raspberry pi 4)

En la siguiente imagen 5.65 vemos dicha configuración.

```
self.client = MQTT.Client(client_id=self.device_ID, clean_session=False)
self.client.username_pw_set(username="cliente", password="123")
self.client.tls_set(ca_certs="/etc/mosquitto/certs/tls/ca.crt")
self.local_Broker = '192.168.0.20'
self.local_Port = 8883
```

Figura 5.65: Parte del servidor MQTT local

Parte del servidor MQTT remoto (En nuestro caso es el servidor que esta alojado en la empresa):

En la siguiente imagen 5.66 vemos dicha configuración.

```
self.remote_client.username_pw_set(username="cliente", password="123")
self.remote_client.tls_set(ca_certs="/etc/mosquitto/ca_certificates/tls/ca.crt")
self.remote_client.connect(self.remote_Broker , self.remote_Port)
self.remote_client.on_connect = self.on_connect
self.remote_client.on_message = self.on_message
```

Figura 5.66: Parte del servidor MQTT remoto

Luego procederemos finalmente a ejecutar el script main.py. En la figura 5.67 vemos la ejecución del script.

Figura 5.67: Proceso de publicación cliente

Para que los mensajes MQTT se están publicando correctamente a través del script, nos suscribimos al tema "IOTDEVICES". Y como vemos en la figura 5.68, se están recibiendo correctamente.

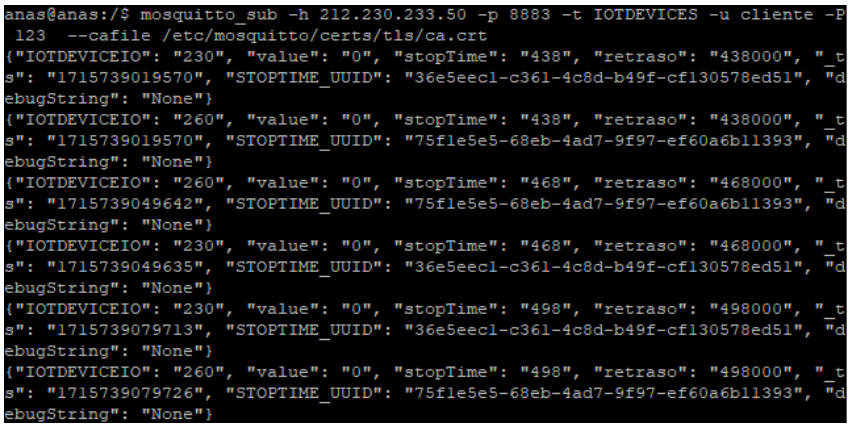


Figura 5.68: Información del script main.py

Para acabar, nos aseguramos en el wireshark de que los paquetes estén bien encriptados y de que se esté usando el protocolo TLS. En la figura 5.69 observamos que los datos están bien encriptados.

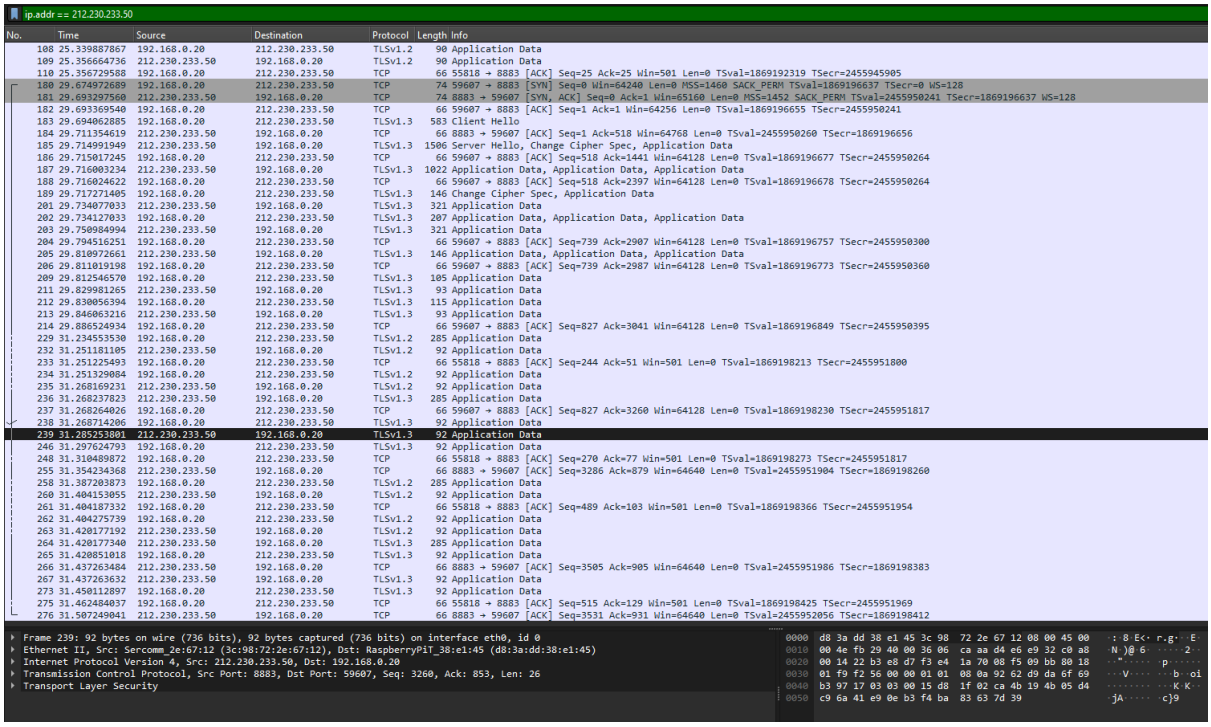


Figura 5.69: Captura paquete wireshark

Capítulo 6

Mantenimiento

6.1. Gestión de certificados

La gestión de certificados implica **la renovación periódica** de los certificados TLS utilizados para asegurar las comunicaciones. Esto es necesario debido a que los certificados tienen una fecha de vencimiento y deben ser actualizados para garantizar la seguridad de las conexiones.

El proceso de renovación de certificados generalmente implica **la generación de nuevos certificados** y la actualización de los certificados existentes en los sistemas y dispositivos pertinentes. Además, es importante verificar que los nuevos certificados se implementen correctamente y que todas las conexiones continúen siendo seguras después de la actualización.

6.2. Verificación VPN

La verificación de la VPN **implica monitorear** continuamente el estado de las conexiones VPN para detectar cualquier problema o interrupción en el servicio. Esto es esencial para garantizar la disponibilidad y la confiabilidad de las conexiones VPN, especialmente en entornos donde se depende de ellas para la comunicación segura entre **diferentes ubicaciones o dispositivos**.

El mantenimiento de la VPN puede incluir la configuración de alertas automáticas para notificar al personal de TI sobre cualquier caída de la conexión VPN. Además, se deben establecer procedimientos para diagnosticar y resolver rápidamente cualquier problema que surja, lo que puede implicar la intervención manual para **restablecer la conexión** o identificar y solucionar problemas de red subyacentes.

6.3. Ajustes ACL

Los ajustes ACL (Listas de Control de Acceso) son configuraciones que controlan **el acceso a recursos de red** específicos en función de ciertos criterios, como direcciones IP, puertos o protocolos. El mantenimiento de ACL implica revisar regularmente y actualizar estas configuraciones para garantizar que sigan siendo adecuadas y efectivas para proteger la red contra accesos no autorizados y ataques.

Los ajustes ACL pueden requerir modificaciones periódicas debido a cambios en los requisitos de seguridad, actualizaciones de software o infraestructura de red, o cambios en las políticas de acceso. Es importante realizar estas actualizaciones de manera cuidadosa y planificada para evitar interrupciones no deseadas en el acceso a los recursos de red y garantizar que la seguridad de la red se mantenga al día con las últimas **amenazas y vulnerabilidades**.

Capítulo 7

Conclusiones

En el transcurso de este proyecto, hemos explorado **diversas tecnologías** y conceptos relacionados con **la infraestructura** de tecnología de la información. A través de la implementación de soluciones de seguridad como TLS, la configuración de VPNs y la gestión de ACLs, hemos fortalecido la seguridad y la integridad de nuestras comunicaciones y recursos de red.

Uno de los hallazgos más destacados ha sido la **eficacia de la implementación** de TLS sobre el protocolo MQTT para garantizar la seguridad de nuestras comunicaciones. La adopción de certificados SSL/TLS ha proporcionado una capa de protección, asegurando la confidencialidad y la autenticidad de los datos transmitidos.

Para futuras mejoras, sería importante considerar el mantenimiento continuo de los certificados de seguridad, incluyendo la renovación oportuna y la gestión eficaz de los mismos. Además, la realización periódica de auditorías de seguridad y pruebas de penetración ayudaría a identificar posibles vulnerabilidades y garantizar la robustez del sistema frente a amenazas potenciales. **Integrar sistemas de monitoreo en tiempo real** para detectar y **responder rápidamente a cualquier anomalía** también sería una medida valiosa para mantener la seguridad del entorno industrial.

Bibliografía

[1] autonomosyemprendedor.es. Modelo de negocio b2b. https://www.autonomosyemprendedor.es/articulo/guias-de-emprendimiento/que-es-b2b-beneficios-inconvenientes/20231228235910033853.html. [Consulta: 26 de Mayo de 2024].

[2] bbc.com. Caso stuxnet en la plata nuclear de irán. https://www.bbc.com/mundo/noticias/2015/10/151007_iwonder_finde_tecnologia_virus_stuxnet. [Consulta: 26 de Mayo de 2024].

[3] bytebeam.io. Guia seguridad para principiantes mqtt. https://bytebeam.io/blog/beginners-guide-to-mqtt-security/#:~:text=User%20Credentials%20and%20Access%20Control&text=Set%20up%20unique%20user%20credentials,password%20changes%20for%20enhanced%20security. [Consulta: 26 de Mayo de 2024].

[4] dle.rae.es. Significado de pyme. https://dle.rae.es/pyme. [Consulta: 26 de Mayo de 2024].

[5] dongee.com. Distribución de puertos en redes. https://www.dongee.com/tutoriales/que-es-un-puerto-de-red/. [Consulta: 26 de Mayo de 2024].

[6] itbacking.com. Pagina principal web it.backing. https://www.itbacking.com/. [Consulta: 26 de Mayo de 2024].

[7] luisllamas.es. ¿qué es mqtt? su importancia como protocolo iot. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/. [Consulta: 26 de Mayo de 2024].

[8] mosquitto.org. Configuración acls. https://mosquitto.org/man/mosquitto-conf-5.html. [Consulta: 26 de Mayo de 2024].

[9] ssl.com. ¿qué es la criptografía de curva elíptica (ecc)? https://www.ssl.com/es/preguntas-frecuentes/%C2%BFQu%C3%A9-es-la-criptograf%C3%ADa-de-curva-el%C3%ADptica%3F/. [Consulta: 26 de Mayo de 2024].

Anexo A

Tabla de acrónimos

Acrónimo	Significado
MQTT	Message Queuing Telemetry Transport
TLS/SSL	Transport Layer Security/Secure Sockets Layer
VPN	Virtual Private Network
IoT	Internet of Things
EDT	Estructura de Desglose de Trabajo
ACLs	Access Control Lists
GMAO	Gestión de Mantenimiento Asistido por Computadora
PLC	Controlador Lógico Programable
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ECC	Elliptic curve cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
IANA	Internet Assigned Numbers Authority
VoIP	Voice over Internet Protocol
SSH	Secure Shell
Mes	Manufacturing Execution System
PYME	Pequeña y mediana empresa

Cuadro A.1: Acrónimos y sus significados.

Anexo B

Tabla de puertos

Puerto	Protocolo/Propósito
1883	MQTT (sin cifrado estándar)
1885	MQTT (sin cifrado usado en la implementación)
8883	MQTT (con cifrado TLS/SSL)
51194	WireGuard VPN
22	SSH (Secure Shell)
4500	IPSec NAT-T (Internet Protocol Security Network Address Translation - Traversal)
500	IPSec IKE (Internet Protocol Security Internet Key Exchange)
1195	OpenVPN

Cuadro B.1: Puertos y sus protocolos/asignaciones correspondientes.

Anexo C

Abrir puertos casa

En este anexo explicaremos detalladamente como abrir los puertos de un router Vodafone. La finalidad de esto es para que la Raspberry pi 4 (situada en mi casa) se pueda conectar al servidor alojado en la empresa.

Ahora mostrare el procedimiento para abrir dichos puertos en mi router (en mi caso tenia un router de Vodafone).

Buscamos en nuestro motor de búsqueda en línea, en nuestro caso google,la dirección IP de nuestro router. Normalmente es la 192.168.0.1, en nuestro caso si que lo era. En la figura C.1 observamos la IP privada del router.

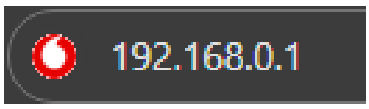


Figura C.1: Dirección IP del router

Nos saldrá una pagina web para poner las credenciales, las introducimos. En la barra de navegación pulsamos sobre Internet y luego en Redirección de Puertos. En la figura C.2 vemos la sección.

Figura C.2: Lugar de Sección Redirección de Puertos del router

Cuando ya estemos en redirección de Puertos, pincharemos en el botón + para agregar una redirección de Puerto. En la figura C.3 vemos el botón para agregar una redirección de Puerto.

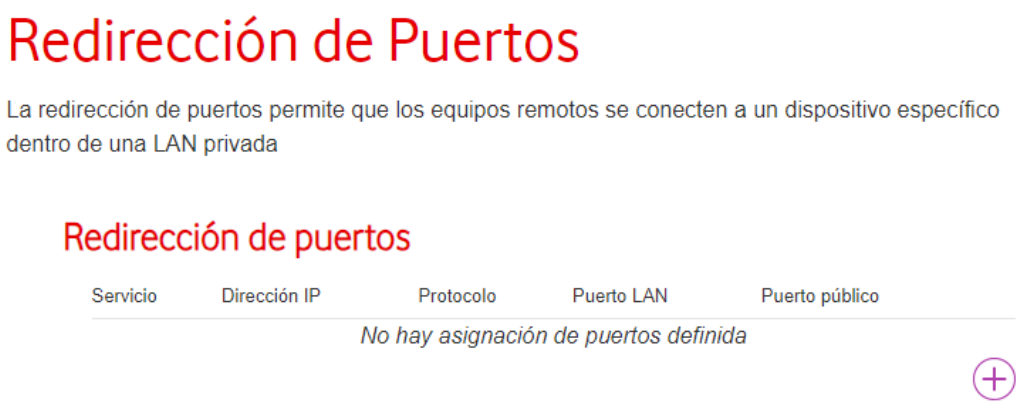


Figura C.3: Sección Redirección de Puertos del router

Cuando le demos al botón +, nos saldrá un formulario para agregar el servicio: UDP O TCP (en nuestro caso TCP), el dispositivo al que queremos hacer la redirección de puerto o en vez de coger el dispositivo, si nos sabemos su IP privada, podemos introducirla. Luego escogemos abrir solo 1 puerto y introducimos el puerto deseado a abrir. En nuestro caso serán los puertos 51194, 1885, 8883, 4500 y 500. En la figura C.4 observamos el formulario para proceder a esto.

Figura C.4: Formulario para agregar redirección de puerto

Pinchamos en guardar y se nos habrá añadido una entrada con los campos de hemos introducido en el formulario. En la siguiente figura C.5 vemos la nueva línea.

Figura C.5: Redirección de puerto agregada

Repetiremos este procedimiento con todos los puertos. Y ya tendríamos todo para empezar con la Implementación servidor MQTT.