UNIVERSIDAD AUTÓNOMA METROPOLITANA

Casa abierta al tiempo

UNIDAD IZTAPALAPA

División de Ciencias Básicas e Ingeniería

Posgrado en Ciencias (Ciencias y Tecnologías de la Información)

“Integración de un mecanismo de autenticación TOTP en un ambiente MQTT para dispositivos de gama baja”

TESIS

Que para obtener el grado de Maestro en Ciencias (Ciencias y Tecnologías de la Información)

Presenta:

Rubén Darío García González

Matrícula: 2222800215

ggrd.na00@gmail.com

Director

Dr. Luis Martín Rojas Cárdenas

Jurado:

Presidente:

Dr Rubén Vázquez Medina

Secretario

Dr. Leonardo Palacios Luengas

Vocal

Mtro. Omar Lucio Cabrera Jimenez

Iztapalapa, Ciudad de México a 13 de junio de 2025

Agradecimientos

En primer lugar, quiero expresar mi más profundo agradecimiento a mi asesor de tesis el Dr. Luis Martín Rojas por su invaluable guía, paciencia y apoyo a lo largo de este proyecto. Su experiencia y consejos fueron fundamentales para la culminación de esta investigación.

Agradezco también a la UAM Iztapalapa y a todos los profesores que a lo largo de mi formación me brindaron las herramientas necesarias para llevar a cabo este trabajo. También a mis compañeros de la generación 2022, quienes me acompañaron en cada paso de este camino, por su constante colaboración y amistad.

Un agradecimiento especial a Cyril Russo, autor del software esp-eMQTT5 ya que ha sido una herramienta crucial en la realización de esta investigación. Su trabajo ha facilitado de manera significativa la implementación de la solución y el desarrollo de los resultados obtenidos.

Agradezco profundamente a mis padres Carmen y Rubén, que, con su cariño y comprensión, siempre han sido mi pilar. Gracias por apoyarme en cada decisión, por sus consejos y por su infinita paciencia. A ellos les debo todo lo que soy y este logro también es suyo.

A mi hermana Ana Elena por estar siempre a mi lado, por cuidarme, por creer en mí y por darme fuerzas cuando más las necesitaba.

A mi mascota, Cosme, porque nunca sabes cuánto puedes querer a un perro hasta que lo haces.

A mis amigos Cristian, Enrique, Francisco y Roberto por las risas, la motivación, las noches de juego y por estar siempre ahí, incluso en los momentos más difíciles. A todos, mi más sincero agradecimiento.

Finalmente, pero no menos importante, agradezco al Conahcyt, por brindarme los recursos necesarios para llevar a cabo esta investigación.

Resumen

El Internet de las Cosas (IoT) es una de las tecnologías que más rápidamente está avanzando y que promete transformar diversos ámbitos de nuestra vida cotidiana, pues permite que diversos dispositivos físicos, aplicaciones domésticas, vehículos, electrodomésticos, etc., estén interconectados, pero también expuestos a los riesgos que conlleva el uso de Internet. En consecuencia, su uso implica la necesidad fundamental de proteger la red de ataques e intromisiones indeseadas. Los dispositivos IoT suelen utilizar protocolos de comunicación máquina a máquina, como MQTT (*Message Queueing Telemetry Transport*). Debido a su diseño basado en la estrategia de publicación-suscripción y a la naturaleza heterogénea de las redes IoT, MQTT se convierte en un objetivo atractivo para múltiples tipos de ataques. En este sentido, numerosas técnicas de ataque a redes se enfocan en este protocolo.

Esta tesis propone e implementa un esquema de autenticación que aprovecha la infraestructura *Enhanced Authentication* que nos ofrece el más reciente estándar MQTT v5.0. Esta propuesta consiste en la integración del esquema de autenticación TOTP (*Time-based One Time Password*) en el protocolo MQTT para dispositivos de bajo poder de cómputo reemplazando la función *hash* subyacente del algoritmo TOTP con el objetivo de reducir el consumo de recursos computacionales sin comprometer la seguridad. Se amplía la funcionalidad de la plataforma Mosquitto mediante el desarrollo de un plugin que implementa nuestro esquema y así evaluar su desempeño. Los resultados muestran que el esquema propuesto requiere apenas de un incremento marginal en el uso de recursos para proporcionar una autenticación mucho más robusta, comparada con la autenticación simple usuario-contraseña predeterminada de MQTT.

Índice general

Índice de tablas

Índice de figuras

Lista de acrónimos y abreviaturas

1.2 Abreviatura

1.3 Significado

API	Application Programming Interface
AugPAKE	Augmented Password-Authenticated Key Agreement
BCM	Business Continuity Management
CISA	Cybersecurity and Infrastructure Security Agency
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DDOS	Distributed Denial of Service
ECDHE	Elliptic Curve Diffie-Hellman Key Exchange
GE	Gate Equivalent
HMAC	hash-based Message Authentication Code
HOTP	hash-based One Time Password
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IIoT	Industrial Internet of Things
IP	Internet Protocol

LDAP	Lightweight Directory Access Protocol
M2M	Machine to machine
MFA	Multiple Factor Authentication
MQTT	Message Queueing Telemetry Transport
OASIS	The Organization for the Advancement of Structured Information Standards
PFS	Perfect Forward Secrecy
PSK	Pre-shared Key
PUBACK	Publish Acknowledge
PUBCOMP	Publish Complete
PUBREC	Publish Received
PUBREL	Publish Release
QoS	Quality of Service
RFID	Radio Frecuency Identification
SA	Servidor de Autorización
SCRAM	Salted Challenge Response Authentication Mechanism
SFA	Single Factor Authentication
SSH	Secure Shell
TI	Tecnologías de la Información
TCP	Transport Control Protocol

TOTP	Time-based One Time Password
UCON	Usage Control
UDP	User Datagram Protocol
WSN	Wireless Sensor Network
XMPP	Extensive Messaging and Presence Protocol

Capítulo 1

1. Introducción

El Internet de las Cosas (IoT) es una de las tecnologías más prometedoras de los últimos años, se prevé que en el transcurso de los siguientes 7 años la cantidad de dispositivos IoT conectados a Internet en el mundo llegue a 25,000 millones y se espera que para 2030 el 75% de todos los dispositivos conectados serán IoT [41].

Para que los dispositivos puedan comunicarse, se requiere un protocolo estandarizado. En la actualidad, existen numerosos candidatos que pueden ocupar ese lugar, pero uno en particular ha ganado terreno en la comunidad: el *protocolo Message Queuing Telemetry Transport* (MQTT).

Han habido grandes avances en la industria del IoT para diferentes propósitos y usos. Cada campo tiene un gran número de diferentes especificaciones, objetivos, desafíos y requerimientos de seguridad, estos últimos se han convertido en uno de los mayores desafíos para los desarrolladores debido a la exposición creciente de datos a más aplicaciones.

MQTT fue concebido originalmente para su uso en redes cerradas por lo que la seguridad no fue una consideración clave durante su diseño, de ahí que una de las principales preocupaciones al utilizar este protocolo en las redes IoT sea la seguridad.

Para esta investigación, la seguridad se define como la protección de la información contra el acceso o interferencia no autorizados al garantizar la confidencialidad, la integridad y la autenticidad de los datos. La confidencialidad se define como la protección de los datos contra su revelación a personas, organizaciones o sistemas no autorizados. La integridad se define como la prevención de la modificación de los datos de parte de personas o sistemas no autorizados. Y la autenticidad se refiere a la verificación apropiada de un dispositivo o un sistema siguiendo un proceso especial de identificación. En este trabajo se abordará en particular el problema de la autenticación, es decir, cómo tener la certeza de quien accede al dispositivo es quien dice ser.

Los sistemas embebidos con capacidades de comunicación que se integran al Internet son cada vez más numerosos, sin importar el tipo de aplicación para la cual estén destinados o el nivel de rango e importancia de estos, la tecnología de los dispositivos IoT debe contar con métodos de

contención destinados a anular todo tipo de ataque proveniente de la red. En efecto, por el simple hecho de estar conectados, estos sistemas están a merced de cualquier tipo de ente malicioso con intenciones diversas, pero en todo caso ilegítimas, poco éticas y con consecuencias que pueden ser irreparables.

La cantidad de investigaciones que ofrecen soluciones a los problemas de seguridad en el terreno de los dispositivos IoT, si bien son numerosas, aún no son definitivas. Un primer análisis del estado del conocimiento muestra que los trabajos que buscan robustecer la seguridad de los dispositivos IoT enfocan sus esfuerzos en los sistemas construidos alrededor de procesadores dotados con capacidad de cómputo suficiente para soportar mecanismos de seguridad estándar.

En efecto, una parte importante de los dispositivos IoT queda sin atención debido a sus escasos recursos computacionales y a los cuales nos referiremos como IoT de gama baja. Consideramos que estos dispositivos cuentan con procesadores de 8 a 32 bits con modestas capacidades de almacenamiento de sólo algunas decenas de kilobytes en RAM y ROM, y no están dotados de máquinas de cifrado vía *hardware*.

En la literatura, uno de los principales obstáculos que se mencionan para no integrar mecanismos robustos de autenticación en los sistemas IoT es su baja capacidad de cómputo. Los estudios se centran en dispositivos capaces de implementar complejos algoritmos de cifrado o en el uso de canales alternativos de comunicación para la autenticación, dejando de lado los dispositivos de gama baja, incapaces de realizar su función e implementar estas soluciones al mismo tiempo debido a los reducidos recursos con los que cuenta, entre otros, un bajo poder de cálculo y una memoria reducida.

Motivado por estos problemas, este trabajo propone implementar un modelo de autenticación segura para el protocolo MQTT adaptado a los recursos limitados de los dispositivos IoT conectados y resistente a ataques, con el fin de evitar que cualquier entidad maliciosa o no autorizada tenga acceso a un sistema IoT que utilice el protocolo MQTT.

Se propone un modelo de autenticación dinámico, seguro y ligero a través de contraseñas de un solo uso (OTP) para el protocolo MQTT versión 5 (MQTT v5.0). Este modelo aborda los problemas y desafíos mencionados anteriormente:

- Brinda un mecanismo de autenticación ligero para su implementación en dispositivos con recursos limitados.

- Permite una autenticación rápida mediante un único intercambio de información, minimizando las latencias.

- Ofrece una autenticación dinámica mediante contraseñas de un solo uso (OTP) que resisten ataques de fuerza bruta, ataques de repetición, criptoanálisis y ataques de intermediarios (*man-in-the-middle*).

1.1. Objetivos

1.1.1. Objetivo general

Integrar un sistema de autenticación al protocolo MQTT para dispositivos IoT que sea funcional en dispositivos con bajo poder de cómputo.

1.1.2. Objetivos específicos

- Evaluar las alternativas tecnológicas disponibles que permitan desarrollar el sistema de autenticación conforme a los requerimientos establecidos.

- Proponer e implementar la arquitectura de un sistema de autenticación TOTP que se integre conforme a las normas del protocolo MQTT v5.0.

- Realizar pruebas de funcionalidad.

- Evaluar y realizar la idónea comunicación de resultados.

1.2. Metodología

La metodología empleada para este trabajo de investigación se describe a continuación:

- Revisión y análisis de la literatura sobre trabajos relacionados.

- Estudio del protocolo MQTT v 5.0 e implementación en un microcontrolador ESP32.

- Estudio del método de autenticación TOTP e implementación en un microcontrolador ESP32.

- Estudio de las funciones *hash* criptográficas de bajo costo.
- Selección de una función *hash* de bajo costo que reemplace a las usualmente utilizadas.
- Implementación del método TOTP usando la función *hash* ligera seleccionada en un microcontrolador ESP32.
- Integración del TOTP en MQTT v5.0 usando la infraestructura "*Enhanced Authentication*" proporcionada por el protocolo.
- Validación de la solución propuesta usando tarjetas electrónicas de desarrollo S2 Mini.
- Elaboración de conclusiones.
- Comunicar los resultados.

1.3. Aportaciones del trabajo

Los principales aportes de este proyecto de investigación son los siguientes:

1. La adecuación del algoritmo TOTP de modo que use una función *hash* criptográfica ligera en lugar de las recomendadas en el RFC (SHA1), de modo que se reduzca la carga computacional en dispositivos de recursos limitados.
2. La integración del algoritmo antes citado en MQTT v5.0 usando la infraestructura "*Enhanced Authentication*" que forma parte del protocolo.
3. El desarrollo de un *plugin* para el *broker* Mosquitto 2.0 que integrará el método de autenticación TOTP al protocolo MQTT v5.0.
4. La puesta a punto del cliente MQTT esp-eMQTT5, trabajando conjuntamente con su creador, Cyrill Russo.

1.4. Organización del documento

Esta tesis se estructura de la siguiente manera:

Capítulo 2: Se profundiza en la esencia del IoT, destacando las tecnologías detrás de su surgimiento y exponiendo sus aplicaciones y componentes típicos. Se proporciona una revisión de

las soluciones de seguridad para IoT, así como los desafíos asociados. Se discuten los principios básicos de la seguridad informática relevantes en este ámbito.

Capítulo 3: Se centra en el Protocolo MQTT, sus objetivos, su arquitectura, sus mensajes incluyendo una explicación sobre la sintaxis y semántica de estos. También se proporciona un breve análisis sobre la seguridad de MQTT y los ataques asociados.

Capítulo 4: Presenta una revisión exhaustiva de la literatura. El capítulo también revisa investigaciones previas relacionadas con el tema, diferenciando nuestro* enfoque de los estudios existentes.

Capítulo 5: Presenta a fondo la propuesta, incluyendo la arquitectura y las ventajas y desventajas sobre las soluciones existentes.

Capítulo 6: Muestra el diseño del banco de pruebas y su implementación.

Capítulo 7: Concluimos nuestra tesis en este capítulo, seguido de trabajos futuros.

Capítulo 2

2. Marco Conceptual

Este capítulo en particular ofrece una exploración exhaustiva de los principios fundamentales del Internet de las Cosas (IoT). Se clarifica la definición de IoT, las posibles aplicaciones y su arquitectura, incluyendo los elementos clave y los protocolos utilizados dentro del mismo.

2.1. Introducción a IoT

El término IoT describe la amplia red de objetos tangibles en todo el mundo conectados a través de Internet colectivamente recolectando y compartiendo información. Fomenta una transformación en la cual los dispositivos con capacidad de Internet se convierten en un ecosistema interconectado en el que los datos digitales son accesibles de forma ubicua en todo momento [1]. Los dispositivos IoT abarcan desde objetos pequeños hasta maquinaria a gran escala y pueden comunicarse fácilmente entre sí a través de Internet sin necesidad de intervención humana [2].

2.1.1. Características del IoT

El IoT combina diferentes tipos de *hardware* y *software* en un sistema integrado. Estas soluciones aprovechan la tecnología de la información, que abarca dispositivos y programas capaces de almacenar, recuperar y procesar datos [2].

El Internet es la principal forma en que los dispositivos se comunican, teniendo como aplicación tecnologías como RFID y WSNs que emplean sensores para monitorear el entorno. Estos dispositivos suelen disponer de capacidades de potencia de procesamiento, memoria, almacenamiento y vida útil de la batería limitadas [3].

Hay algunas características cruciales del IoT:

Conectividad: Un requisito imprescindible de la infraestructura del IoT es que los dispositivos puedan estar conectados a la infraestructura de Internet. Cualquiera, en cualquier lugar y en cualquier momento puede garantizar la conectividad. Sin conexión, nada tiene sentido.

Inteligencia e identidad: La extracción de conocimiento y el análisis a partir de los datos generados es clave. Por ejemplo, se puede tener un sensor que genere datos, pero esos datos solo serán útiles si se interpretan de forma correcta.

Cada dispositivo IoT viene con una identidad única. Esta identificación es útil para rastrear el equipo, y ocasionalmente, consultar su estado.

Heterogeneidad: Esto implica que aunque los dispositivos IoT pueden tener diferentes hardware y redes subyacentes siguen siendo capaces de comunicarse entre sí.

Cambios dinámicos: Los dispositivos en el ámbito del IoT pueden cambiar de estados con frecuencia, incluyendo patrones de actividad e inactividad, estado de conectividad y factores contextuales como ubicación y velocidad.

Escalabilidad: La cantidad de dispositivos conectados al IoT aumenta progresivamente a diario. Es por eso por lo que una configuración de IoT debe ser capaz de manejar la expansión masiva. Los datos generados como resultado pueden ser enormes, por lo que deben manejarse adecuadamente.

Seguridad: Existe cierto riesgo de que los datos personales y confidenciales de los usuarios sean revelados cuando varios de sus dispositivos están conectados a Internet, lo cual puede ser muy grave.Es por eso por lo que la seguridad de los datos es fundamental para IoT. Además, el equipo involucrado también es vulnerable, ya que las redes también podrían estar en riesgo,or tanto, la seguridad del equipo también es imprescindible.

2.1.2. Tecnologías del IoT

El IoT es una tecnología que permite la interconexión de distintos dispositivos y su comunicación con la nube, lo que posibilita la automatización y la eficiencia en distintas industrias. La red de dispositivos interconectados continúa creciendo, impulsada por el progreso tecnológico en sensores, teléfonos inteligentes, tecnología en la nube y capacidades de comunicación. Constituye una red que comprende diversas entidades físicas, incluyendo vehículos, maquinaria, electrodomésticos y más allá. Estas entidades aprovechan múltiples tecnologías para facilitar el intercambio de datos a través de Internet [2]. La tabla 2.1 explica las tecnologías que sustentan el concepto de IoT.

Tabla 2.1
Las tecnologías del IoT

Tecnología	Ejemplo
Sensores y actuadores	Sensores: Termómetros, acelerómetros, giroscopios Actuadores: válvulas, motores, compuertas
Protocolos de comunicación	MQTT, CoAP, HTTP, XMPP
Tecnologias de red	WiFi, Bluetooth, LoRA, Zigbee,
Cómputo en la nube	Software-as-a-service, Infraestructure-as-a-service, Gobierno de datos, ciberseguridad
Análisis de datos	Diagnóstico, descriptivo, predictivo

2.1.2.1 Sensores y actuadores

Un sensor es un dispositivo utilizado para la conversión de eventos o características físicas en señales eléctricas. Dicho de otra forma, es un dispositivo de *hardware* que toma la entrada del entorno y la proporciona al sistema al convertirla. Por ejemplo, un termómetro toma la temperatura como característica física y luego la convierte en señales eléctricas para el sistema.

Un actuador es un dispositivo que convierte las señales eléctricas en eventos o características físicas. Toma la entrada del sistema y proporciona la salida al entorno. Por ejemplo, los motores y calentadores son algunos de los actuadores más comúnmente utilizados.

2.1.2.2 Protocolos de comunicación

Los protocolos de comunicación son las reglas que gobiernan cómo los dispositivos se comunican entre sí. Existen muchos protocolos de comunicación diferentes utilizados en el IoT, cada uno con sus propias ventajas y desventajas. Algunos de los protocolos de comunicación más comunes para dispositivos IoT incluyen MQTT, CoAP (*Constrained Application Protocol*) y HTTP (*Hypertext Transfer Protocol*).

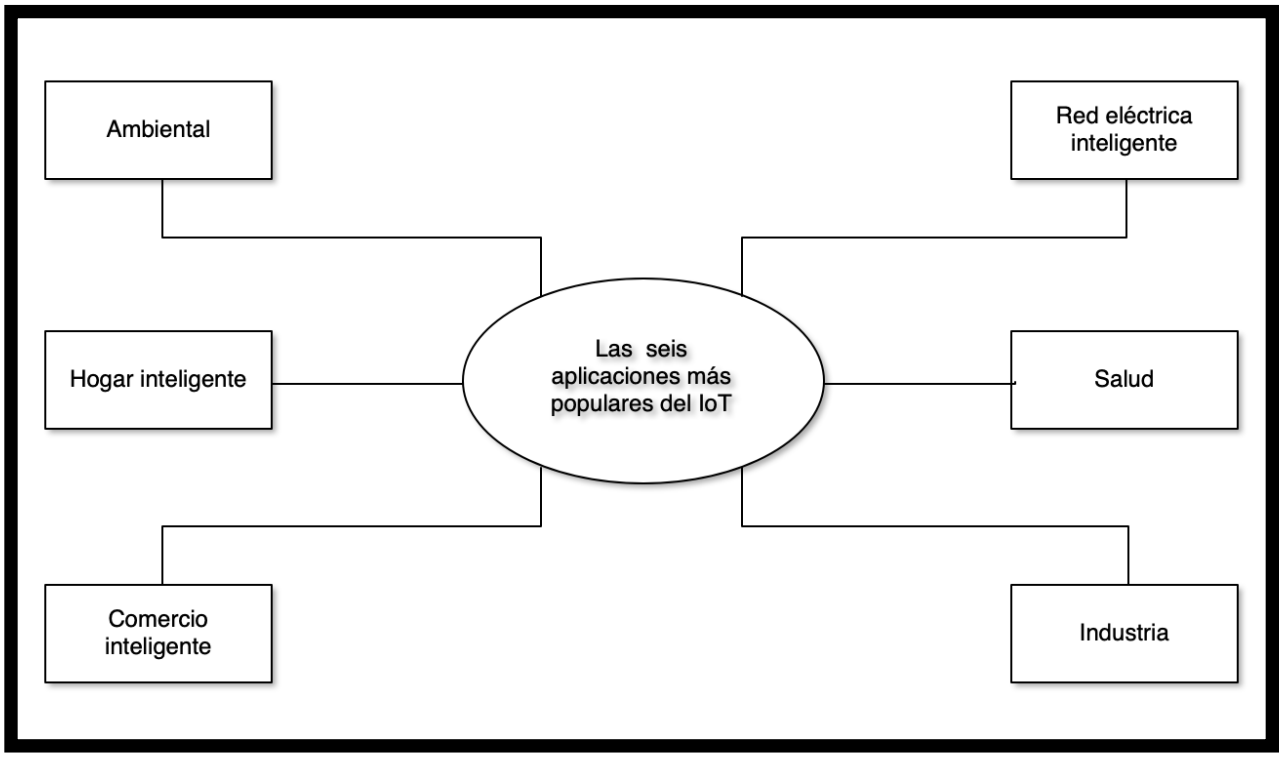

Figura 2.1. Aplicaciones del IoT

2.1.2.3 Tecnologías de red

Las tecnologías de red permiten que los dispositivos IoT se comuniquen entre sí con el objetivo de compartir datos y ofrecer servicios. Esta comunicación puede ser local o mediante Internet. Existe una gran variedad de tecnologías de red que se pueden usar para dispositivos IoT, incluyendo *Wi-Fi*, celular, Zigbee, LoRa y LPWAN (*Low Power Wide Area Network*).

2.1.2.4 Cómputo en la nube

El cómputo en la nube proporciona una plataforma para almacenar, recuperar y procesar datos provenientes de dispositivos IoT, lo que es esencial ya que permite a dichos dispositivos almacenar y procesar grandes cantidades de datos sin tener que disponer de su propio hardware dedicado.

2.1.2.5 Análisis de datos

El análisis de datos es el proceso al que se someten los datos para extraer información lo que es fundamental para el IoT, ya que permite a las empresas obtener información valiosa a partir de los datos recopilados por los dispositivos IoT.

2.1.3. Aplicaciones IoT

Las aplicaciones del IoT son múltiples porque es ajustable a casi cualquier tecnología que sea capaz de aportar información relevante sobre su propio funcionamiento, sobre el desempeño de una actividad e incluso sobre las condiciones ambientales que necesitemos monitorear y controlar a distancia.

Hoy en día, muchas empresas de diferentes rubros o sectores están adoptando esta tecnología para simplificar, mejorar, automatizar y controlar diferentes procesos. A continuación, mostramos algunas de las aplicaciones más populares (figura 2.1)

El concepto de hogar inteligente es una de esas aplicaciones populares. Incluye una variedad de dispositivos inteligentes como cerraduras, monitores de bebés y detectores de incendios que se comunican a través de canales inalámbricos. Estos electrodomésticos pueden ser accesibles de forma remota a través de un *gateway* local.

El IoT en medicina, también conocido como IoMT (*Internet of Medical Things*) se refiere a la utilización de aparatos médicos que pueden intercambiar datos informáticos entre sí o conectarse con plataformas en línea mediante una red *wifi*, con la finalidad de que dichos aparatos funcionen de manera sincronizada y le permitan al profesional de la salud manejar toda la información en tiempo real.

Dentro de la misma área, el IoT permite a los médicos registrar y guardar en la nube la historia clínica de cada paciente, con el fin de poder brindar la atención adecuada y hacer un seguimiento del caso a distancia.

Dentro del ámbito del transporte inteligente, una vasta red de vehículos inteligentes cuenta con sistemas y dispositivos IoT con la capacidad de comunicarse entre sí, con elementos de infraestructura y con personas a pie a través de canales inalámbricos. Estos sistemas pueden

evaluar situaciones de tráfico en tiempo real, ajustar la velocidad del vehículo y compartir información para garantizar experiencias de manejo eficientes y seguras.

Por otro lado los sistemas ambientales proporcionan gestión remota de factores como temperatura, humedad, niveles de agua, humedad del suelo y condiciones climáticas específicas para optimizar los estándares de producción. En la ganadería avanzada se pueden colocar sensores en animales, ofreciendo información sobre su comportamiento para garantizar su bienestar.

El sector industrial ha adoptado el IoT, dando lugar a la industria inteligente. El objetivo principal del IIoT (*Industrial Internet of Things*) es mejorar la supervisión del proceso de fabricación, los datos y los desafíos emergentes, garantizando la calidad de los productos finales.

En el campo de las ventas existe la "venta inteligente" que se refiere a una red de dispositivos conectados que recopilan y comparten datos para automatizar y optimizar procesos clave en las tiendas y cadenas de suministro como la gestión de inventarios, seguimiento de activos y análisis de clientes.

El verdadero valor del IoT en el comercio radica en su capacidad para proporcionar datos en tiempo real, lo que permite a los minoristas tomar decisiones rápidas y eficientes.

Por último, se encuentra la red eléctrica inteligente, un uso común del IoT, diseñado para medir y controlar la cantidad de electricidad utilizada. Ayuda a los usuarios a manejar su consumo de electricidad de manera confiable, ahorra energía y reduce la posibilidad de problemas con la red eléctrica.

2.1.4. Arquitectura del IoT

El IoT está construido sobre una variedad de capas tecnológicas que componen su estructura fundamental. Esta arquitectura delinea la interrelación de diversas tecnologías y la comunicabilidad, modularidad y configuración de las implementaciones de IoT en escenarios variados.

La arquitectura del IoT consta de cuatro capas: la capa de percepción, la capa de red, la capa de procesamiento y la capa de aplicación. Cada una cumple una función específica e incluye componentes particulares que contribuyen a la funcionalidad global del sistema IoT. Esta arquitectura proporciona un marco para el diseño e implementación de sistemas IoT, orientando la selección e integración de componentes en cada capa.[5].

2.1.4.1 Capa de percepción

La capa física es el fundamento del Internet de las Cosas. Esta capa abarca muchos dispositivos IoT con sensores y actuadores, cada uno equipado para recopilar, procesar y transmitir información a través de la red. Estos dispositivos pueden estar conectados mediante conexiones cableadas o inalámbricas y utilizar tecnologías inteligentes para recopilar datos.

2.1.4.2 Capa de red

Esta capa es un componente crítico ya que abarca diversas tecnologías de comunicación que permiten la conectividad de los dispositivos del IoT. La capa de red también incluye *Gateways* y *Sistemas de Adquisición de Datos* (DAS) que recopilan, agregan y transmiten los datos a través de la red

2.1.4.3 Capa de procesamiento

Los datos pasan por una serie de transformaciones y cálculos dentro de esta capa para extraer ideas significativas, es una etapa intermedia que sirve de puente entre la recolección de las capas de aplicación y dichos datos, lo que implica prepararlos para ser utilizados por aplicaciones de *software* que supervisan, controlan e implementan acciones posteriores guiadas por todo lo ya interpretado.

2.1.4.4 Capa de aplicación

Esta capa generalmente se encuentra en la nube y es donde los datos procesados se muestran al usuario final para que este pueda tomar decisiones basándose en dichos datos.

2.1.5. Desafíos del IoT

Aunque los dispositivos de IoT gestionan y producen una cantidad considerable de datos, suelen ser económicos debido a que tienden a tener una potencia de cálculo, capacidad de almacenamiento y recursos de memoria limitados. De manera similar, el *software*implementado en los dispositivos de IoT puede estar basado en soluciones de código abierto o ser obsoleto, y a veces incluso, ser defectuoso. En el peor de los casos, podríamos tener un dispositivo de IoT conectado a una red con *software* sin parches de seguridad ni actualizaciones, sin conocer qué puertos están abiertos y visibles desde Internet, y sin control de acceso ni credenciales. Los

dispositivos de IoT, al estar conectados a redes públicas, pueden volverse susceptibles a ataques en ausencia de medidas de seguridad adecuadas. Debido a esta vulnerabilidad, las redes privadas pueden experimentar interrupciones imprevistas que ponen en peligro la accesibilidad del servicio, la seguridad de los datos y la seguridad del usuario.

2.2. Fundamentos de seguridad computacional

Los fundamentos de la seguridad informática son los principios y prácticas fundamentales que tienen como objetivo proteger los diversos componentes de *hardware* y *software*, conocidos como activos, dentro de un entorno informático. Estos fundamentos se centran en garantizar la confidencialidad, integridad, disponibilidad, autenticidad y no repudio de la información para los usuarios autorizados [6].

Confidencialidad: La confidencialidad se centra en mantener en secreto y privacidad los datos, permitiendo únicamente a individuos autorizados acceder a ellos. Esto implica implementar medidas para controlar el acceso físico y técnico, clasificar datos, establecer acuerdos de confidencialidad, implementar políticas de contraseñas fuertes, definir pautas para el uso de Tecnologías de la Información (TI) por parte de los empleados y proporcionar capacitación para detectar y prevenir ataques de ingeniería social [6].

Integridad: La integridad asegura que los datos sólo puedan ser modificados por usuarios autorizados y tiene como objetivo preservar la precisión y el estado inalterado de la información. Las modificaciones no autorizadas, ya sean deliberadas o accidentales, pueden comprometer la integridad de los datos. Para mantener la integridad, las organizaciones emplean medidas preventivas para protegerse contra cambios fraudulentos, tanto en documentos físicos como digitales [7].

Disponibilidad: La disponibilidad se refiere a proporcionar a los usuarios autorizados acceso oportuno a la información y servicios según sea necesario [7]. Para garantizar la disponibilidad, las organizaciones implementan procedimientos de respaldo, adoptan prácticas de gestión de continuidad del negocio (*Business Continuity Management* o BCM) y establecen sistemas de recuperación ante desastres que permiten la duplicación de servicios y aplicaciones esenciales incluso frente a accidentes, desastres naturales o sabotaje intencional.

Autenticidad: La autenticidad consiste en asegurarse de que la información y la comunicación provengan de una fuente confiable. Esto incluye protegerse contra la suplantación de identidad, el *spoofing* y otros tipos de fraude de identidad. Las técnicas comunes utilizadas para establecer la autenticidad incluyen los métodos autenticación de uno o más factores, los certificados digitales y la identificación biométrica.

No repudio: El no repudio se refiere a garantizar que una parte no pueda negar haber enviado o recibido un mensaje o transacción. Esto incluye la protección contra la manipulación de mensajes y los ataques de repetición. Las técnicas comunes utilizadas para establecer el no repudio incluyen firmas digitales, códigos de autenticación de mensajes y marcas de tiempo.

La tabla 2.2 resume los principales objetivos o metas de la seguridad informática.

Esta tesis se centra en la autenticación, de la que hablaremos más a fondo en las siguientes secciones.

Tabla 2.2
Fundamentos de la seguridad informática

Objetivo	Descripción
Confidencialidad	Asegurar que la información se mantenga confidencial y accesible solo para individuos o sistemas autorizados.
Integridad	Mantener la precisión, coherencia y confiabilidad de la información a lo largo de su ciclo de vida.
Disponibilidad	Garantizar que los datos y servicios estén disponibles y sean utilizables por usuarios autorizados en el momento adecuado
Autenticacidad	Validar la identidad de una entidad-usuario, servidor o aplicación cliente antes de otorgarle acceso a un recurso protegido.
Irrenuciabilidad	Asegurar la procedencia de los datos, su autenticidad y que no han sido alterados. Confirmar quién envió la información y verificar la identidad del destinatario de modo que ninguna de las dos partes pueda negar que la comunicación ocurrió o fue manejada de otra manera.

2.3. Tendencias actuales en seguridad IoT

Con el crecimiento del mercado del IoT, las fallas de seguridad han tomado una relevancia cada vez mayor. En la investigación académica reciente se han publicado una gran cantidad de artículos relacionados con dichas fallas. Los desafíos que enfrenta la vulnerabilidad en el IoT varían enormemente, entre ellos se consideran *hardware*, heterogeneidad, recursos limitados, privacidad, despliegues a gran escala y falta de consideraciones de seguridad.

En la actualidad, la autenticación es la consideración de seguridad más popular para conceder acceso a un usuario a las redes de IoT [8]. Asimismo, la investigación sobre el problema de la autenticación se enfoca en la seguridad en IoT, protocolos de autenticación y modelos de ataque. No obstante aún se requieren más y mejores protocolos para proveer un mejor servicio a los usuarios finales.

2.3.1. Autenticación y autorización

La autenticación es el proceso de probar que un usuario, sistema o dispositivo tiene permiso de acceder a la información contenida en un sistema, [9] esto se logra al verificar que la persona se encuentre en la lista de entidades autorizadas. Tal procedimiento es requerido en sistemas que no sean públicos o que contengan información privada relativa a una persona o grupo de personas. Los mecanismos de autenticación pueden dividirse en dos grandes grupos: la autenticación de un sólo factor (SFA) y la autenticación de múltiples factores (MFA) .

Los mecanismos SFA son las más simples formas de autenticación [10], en estos el cliente se identifica en el sistema ingresando sólo un secreto, usualmente un nombre de usuario y una contraseña, aunque también se puede usar OTP o reconocimiento facial. Los sistemas SFA se prefieren porque son más simples de implementar [11].

Por su parte, los mecanismos MFA se usan generalmente en sistemas que requieren medidas de seguridad más avanzadas, ya que buscan incrementar la seguridad aumentando el número de factores que utiliza el usuario para ingresar al sistema. Además de la contraseña pueden solicitarse rasgos biométricos, OTP, preguntas de seguridad o el uso de *hardware* dedicado.

2.3.2. Autenticación en IoT

La autenticación en los ecosistemas IoT es problemática debido a varias razones importantes, como las limitaciones de recursos inherentes a estos dispositivos, lo que hace que los esquemas de

autenticación convencionales sean ineficientes o inaplicables. Además, la naturaleza autónoma de ciertos dispositivos IoT vuelve a los esquemas basados en contraseñas mucho menos útiles en estos sistemas. Otra preocupación es que algunos de los esquemas de autenticación existentes son relativamente fáciles de romper y, por lo tanto, no pueden garantizar la seguridad. Por ejemplo, si un atacante roba las credenciales de un usuario para obtener acceso a la cerradura de una puerta inteligente o al sistema de monitoreo de salud de algún paciente, podría representar una amenaza de vida o muerte. En este contexto, es sumamente importante mantener la seguridad durante el proceso de autenticación.

2.3.3. Mecanismos de autenticación de dos factores

Los mecanismos de autenticación de dos factores son una rama de los mecanismos MFA. Estos proveen mayor seguridad añadiendo una capa extra a un factor único como una contraseña o un token al momento de realizar la autenticación. La razón de esto es prevenir que el factor usado en un mecanismo SFA sea capturado por un atacante e ingrese al sistema. En los últimos diez años, millones de nombres de usuario y contraseñas han sido robados de organizaciones e individuos. La Agencia de Ciberseguridad e Infraestructura del Departamento de Seguridad Nacional (CISA) han anunciado que los mecanismos de autenticación de un solo factor son riesgosos [11].

Los mecanismos SFA se basan generalmente en tres categorías: conocimiento, posesión e inherencia. La autenticación de dos factores se forma combinando dos de estos tipos. Los factores de conocimiento generalmente los determina el usuario mismo, como las contraseñas y los pins. La confidencialidad de estos es crucial, pues pueden usarse en más de un sistema. De acuerdo con un estudio de Google, 13% de las personas usan la misma contraseña en todas sus cuentas [12]. Los factores de posesión se basan en un elemento físico que un usuario tiene y que se usa para verificar su identidad, como una tarjeta o una memoria USB. Por último, los factores de inherencia se basan en características físicas o de comportamiento únicas de una persona, por ejemplo, la forma de la cara, la huella digital o la voz. Los mecanismos MFA generalmente son estáticos y deben almacenarse tanto en el servidor como en el cliente, lo que supone un problema de seguridad en caso de ataque. En este aspecto, los OTP son más seguros debido a que sólo se utilizan una vez por sesión [13].

2.3.4. OTP (*One-Time Password*)

Las contraseñas de un solo uso son cadenas de bits que se generan automáticamente y se usan en una única sesión de autenticación, usualmente constan de 6 a 8 caracteres y se vuelven inválidas después que ha expirado un intervalo de tiempo específico, por lo que deben regenerarse cada vez que esto suceda. Estas contraseñas dinámicas han proporcionado una solución al robo o la pérdida de contraseñas estáticas.

Existen dos enfoques para los mecanismos OTP: basados en el tiempo y basados en eventos. En ambos la contraseña se genera con base en una semilla y un factor variable, este último, en el enfoque basado en eventos o HOTP (OTP basado en *hash*), es el valor de un contador que permite que se generen diferentes OTP's con la función HMAC usando un valor fijo como semilla. Cuando ocurre la autenticación, el contador se incrementa en 1 tanto en el servidor como en el cliente HOTP, de esta forma, se garantiza que ambos sistemas generen los mismos valores con la función *hash*. Situación bajo la cual se verifica la autenticidad del cliente HOTP.

En los sistemas TOTP (OTP basado en tiempo) el factor variable es el tiempo. Las contraseñas se generan basándose en el tiempo transcurrido desde la estampa de tiempo acordada por el cliente y el servidor.

Tanto el enfoque HOTP como el TOPT tienen distintas ventajas relativas de uno sobre el otro, la caducidad de la contraseña en el caso de los sistemas basados en tiempo le da ventaja en términos de seguridad, sin embargo, puede consumir más recursos de cómputo si las contraseñas se generan mediante algoritmos demasiado complejos. En el caso de los OTP basados en eventos, la contraseña debe almacenarse de forma segura porque mantiene su validez por mucho más tiempo[14], pero es más amigable para el usuario porque se puede usar en cualquier momento.

2.3.5. TOTP: protocolo de contraseña de un solo uso basado en el tiempo

El esquema TOTP [14] es un protocolo que permite el único uso de una contraseña que además sólo es válida por un muy corto periodo, como treinta segundos o un minuto, este esquema utiliza la llave secreta compartida almacenada tanto en el cliente como en el servidor. Las llaves se generan ejecutando la operación HMAC entre las llaves compartidas y la información de tiempo en el cliente y el servidor. El tiempo que se usa es el valor actual de tiempo UNIX.

El tiempo UNIX es el número de segundos transcurridos desde el 1 de enero de 1970. Ese número de segundos se divide en intervalos. Las contraseñas creadas en cada uno de ellos sólo son válidas mientras dure.

2.3.6. Implementación

2.3.6.1 Inicialización

Convertir la fecha y la hora a tiempo UNIX

Calcular $N = T/t_s$

T = Número total de periodos en tiempo UNIX

t_s= duración del periodo

Convertir *N* a hexadecimal

Escribir *N* en forma de arreglo de 8 bytes y asignar un valor al mensaje *m*

Escribir la llave secreta en formato de 20 bytes y asignar el valor a *k*

Calcular el valor HMAC usando *k* y *m*

Obtener el valor en decimal de los 4 últimos bits del valor HMAC para determinar el offset.

Obtener los primeros 4 bytes, comenzando desde el offset, del valor HMAC

Ejecutar la operación binaria para cada byte.

Convertir el valor binario al entero *i*

Seleccionar el tamaño del token *n*

Token $t = i \% 10n$

Si ($t < n$) Ent agregar el prefijo

2.3.6.2 Autenticación

El servidor y el cliente crean la contraseña utilizando el algoritmo descrito en 2.3.6 usando la llave secreta compartida en el mismo periodo. El cliente envía el OTP generado al servidor. Si el OTP que recibe el servidor desde el cliente es el mismo que generó, el servidor autoriza la solicitud.

2.3.6.3 Análisis del protocolo

En la estructura TOTP, el secreto se almacena en el servidor. Si los atacantes comprometen el servidor, se pueden obtener las llaves y generar *tokens* válidos. Un atacante que obtiene el secreto puede hacerse pasar por un cliente y autenticarse en el sistema.

Como ventaja de TOTP, puede mencionarse que no es necesario agregar *hardware* adicional pues genera contraseñas basadas en *software*. Esto proporciona beneficios en términos de costo y facilidad de uso. Aunque, si se desea, también puede integrarse en *hardware* especializado.

2.3.6.4 Resincronización del reloj

Los servidores de autenticación cuentan con una recomendación según el estándar TOTP RFC 6238 para acomodar el desfase tanto hacia adelante como hacia atrás en los pasos de tiempo, lo que permite reducir el impacto en los usuarios finales [44].

El *software* de sincronización de tiempo debe configurarse para verificar un número específico de intervalos de tiempo dentro de una ventana de tiempo determinada antes de que se rechacen los OTP y se niegue el acceso.

La sugerencia es permitir tres intentos con cada lapso de 30 segundos (generalmente una verificación en el tiempo actual y dos verificaciones en los dos pasos de tiempo anteriores). Por lo tanto, el desfase máximo de tiempo permitido suele ser de 89 segundos: 29 segundos para la marca de tiempo correcta y 60 segundos para los dos pasos anteriores. Luego, el servidor de autenticación puede registrar automáticamente el desfase de tiempo detectado y ajustar la marca de tiempo para la generación de posteriores TOTP.

2.3.7. Criptografía ligera

La criptografía ligera, también conocida como cifrado ligero [15], es una forma de encriptado diseñada para dispositivos con recursos limitados, esta tecnología utiliza menos memoria RAM, menos recursos informáticos y menor cantidad de energía para proporcionar soluciones seguras en la red. Aunque la combinación de AES (*Advanced Encryption Standard*) y SHA (*Secure hash Algorithm*) son muy buenos para la interfaz de computación, no pueden hacer frente a un entorno de IoT, ya que consumen un exceso de capacidad informática. Por ello, en los últimos años se han desarrollado y utilizado numerosos algoritmos de criptografía ligera para satisfacer la restricción de estos recursos limitados. Organizaciones como el NIST (*National*

Institute of Standards and Technology) han descrito varios métodos que son posibles y útiles para los dispositivos IoT/RFID. A pesar de tener una menor complejidad, mantienen un buen nivel de seguridad para proteger los dispositivos y los sensores de las redes inalámbricas. En definitiva, mayor simplicidad, más eficiencia.

Los beneficios de esto se traducen en que la criptografía ligera exige muchos menos recursos de los dispositivos y tomaría menos tiempo completar sus procesos esenciales. El uso de costosas soluciones pesadas para cada dispositivo pequeño en el IoT también haría que el costo de los dispositivos no sea práctico para las organizaciones que implementan soluciones. Por estas razones, la criptografía ligera funcionaría mejor para asegurar las transmisiones de datos confidenciales que ocurren cada segundo en el IoT.

Las soluciones de dispositivos simples generalmente se basan en criptografía simétrica: una versión de criptografía en la que los remitentes y destinatarios de mensajes tienen la misma clave digital para cifrar y descifrar mensajes. El NIST especifica que los algoritmos criptográficos ligeros deben usar cifrado autenticado con datos asociados o AEAD (*Authenticated Encryption with Associated Data*), lo que significa que el destinatario de un mensaje puede usar la autenticación para verificar la integridad de la información cifrada y no cifrada dentro de este. Esto asegura que los mensajes provienen de quienes dicen ser y que su contenido no ha sido alterado en tránsito.

La criptografía ligera se acerca en el horizonte de las soluciones criptográficas. A medida que el IoT se expande y proyectos como los vehículos autónomos o las ciudades inteligentes se desarrollan a su alrededor, la criptografía ligera probablemente se convertirá en una parte integral de la vida diaria.

En estos últimos años expertos criptográficos han propuesto diversas soluciones de criptografía ligera que tienen un gran compromiso con la seguridad, el rendimiento y el coste en comparación con la criptografía convencional, tales como:

- Cifrados en flujo
- Funciones *hash*
- Cifrados en bloque
- Códigos de autenticación de mensaje

Capítulo 3

3. MQTT

En este capítulo, exploramos MQTT, un protocolo de comunicación ligero y eficiente diseñado para la comunicación M2M (*Machine to machine*). Discutimos la simplicidad, apertura y alta eficiencia de ancho de banda de MQTT, que lo convierte en una opción ideal para entornos restringidos. Además, profundizamos en la arquitectura editor/suscriptor de MQTT, sus componentes (editores, suscriptores y *brokers*) y cómo se dirigen los mensajes según los temas. También abordamos la importancia de la seguridad de MQTT y las posibles vulnerabilidades a las que se enfrenta, junto con una visión general de los ataques comunes a MQTT en entornos de IoT. Comprender MQTT y sus consideraciones de seguridad es esencial para garantizar una comunicación robusta y segura en sistemas del IoT.

3.1. El protocolo MQTT

El protocolo MQTT, conocido por su naturaleza ligera, es sumamente confiable. El patrón de comunicación editor/suscriptor utilizado por este protocolo de estándar abierto, aprobado por OASIS (*The Organization for the Advancement of Structured Information Standards*) [16], es particularmente adecuado para la comunicación máquina a máquina (M2M). Gracias a que funciona sobre TCP, el protocolo MQTT muestra una confiabilidad ejemplar, garantizando un modo de comunicación organizado, bidireccional y sin pérdidas. Opera sobre temas, que se organizan jerárquicamente en categorías bajo las cuales los usuarios pueden publicar mensajes. Posteriormente, cualquier cliente adicional que tenga una suscripción al tema específico recibirá estos mensajes, estableciendo un marco de comunicación organizado y dirigido.

Los tres nodos que conforman la estructura física de MQTT son editores, suscriptores y *brokers*, como se muestra en la figura 3.1. Los editores son nodos que envían mensajes, los

suscriptores son nodos que reciben mensajes, y los *brokers* actúan como intermediarios para

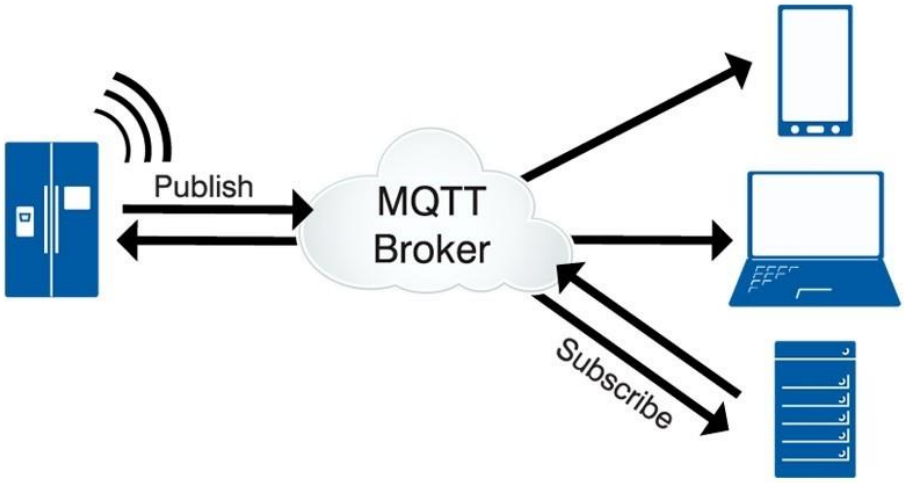

Figura 3.1 Arquitectura de MQTT

coordinar la entrega de mensajes desde los editores a los suscriptores [17].

La arquitectura del protocolo MQTT está diseñada intencionalmente para ser sencilla y eficiente en recursos. Este diseño facilita una comunicación fluida entre clientes y el *broker* central, que es responsable de distribuir mensajes a todos los suscriptores relevantes. La incorporación de temas y niveles de calidad de servicio (QoS) mejora la versatilidad y confiabilidad de la comunicación del cliente.**QoS nivel 0 (A lo sumo una vez):** A menudo llamado "enviar y olvidar", este nivel ofrece la menor garantía. Los mensajes se envían como máximo una vez, posiblemente ninguna, y no implica confirmación ni sobrecarga. Aunque es el modo de transmisión más rápido, carece de garantía de entrega.

QoS nivel 1 (Al menos una vez): El mensaje se transmite al menos una vez por medio de un único intercambio de mensajes PUBLISH. El reenvío del mensaje PUBLISH es una opción para el remitente si no se recibe un acuse de recibo (PUBACK). Sin embargo, este enfoque puede llevar a la duplicación de mensajes.

QoS nivel 2 (Exactamente una vez): Este nivel garantiza la entrega exacta y única del mensaje, representando el nivel más alto de QoS. Implica un proceso de cuatro pasos de negociación entre el remitente y el receptor, lo que lo hace muy adecuado para aplicaciones que requieren garantía de entrega de mensajes. Sin embargo, este nivel impone la mayor sobrecarga.

Los componentes de la negociación incluyen los mensajes PUBLISH, PUBREC, PUBREL y PUBCOMP.

La selección del nivel de QoS depende de las especificaciones particulares de la aplicación de IoT. Si la velocidad es más crucial que la confiabilidad, se podría utilizar un nivel más bajo. Si la garantía de entrega es el factor más crucial, entonces sería adecuado un nivel más alto. La figura 3.2 muestra los tres modos de QoS que se pueden definir en los mensajes PUBLISH intercambiados entre el cliente y el *broker* [17].

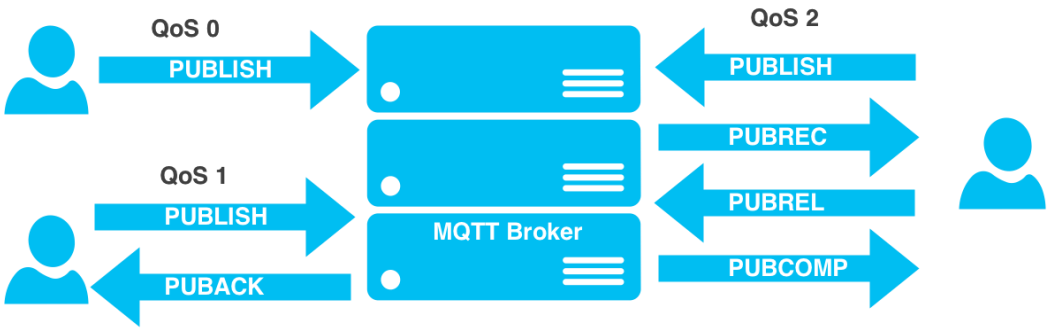

Figura 3.2 Niveles de QoS

3.2. Mensajes MQTT

Los mensajes de comunicación MQTT son el principal mecanismo para transmitir datos entre clientes y el *broker*. Existen varios tipos, cada uno con un propósito y formato específicos los cuales proporcionan la funcionalidad necesaria para una comunicación eficiente y confiable entre clientes y el *broker* [18].

A continuación, se describen algunos de los tipos de mensajes MQTT:

CONNECT: Este es el primer mensaje enviado.

CONNACK: Esta es la respuesta del *broker* al cliente reconociendo la solicitud de conexión. Contiene un código de retorno que indica si la conexión fue aceptada o rechazada.

PUBLISH: Este mensaje sirve para transmitir mensajes de aplicación ya sea del *broker* al cliente o en sentido contrario dentro del marco de comunicación MQTT.

PUBACK: Este mensaje es la confirmación del *broker* al cliente por un mensaje Publish recibido a nivel de QoS 1.

PUBREC: El remitente es el receptor de este mensaje para confirmar un mensaje Publish recibido a nivel de QoS 2.

PUBREL: Este mensaje es enviado del remitente al receptor para asegurar que el mensaje Publish a nivel de QoS 2 fue recibido.

PUBCOMP: El remitente es el receptor de este mensaje para confirmar el mensaje Pubrel.

SUBSCRIBE: Este mensaje es utilizado por el cliente para suscribirse a uno o más temas del *broker*.

SUBACK: Esta es la confirmación del *broker* al cliente indicando que la suscripción a un tema específico fue exitosa.

UNSUBSCRIBE: Este mensaje es enviado por el cliente con la intención de retirar su suscripción de uno o varios temas dentro del sistema MQTT.

PINGREQ: Este mensaje es utilizado por el cliente para verificar que la conexión de red está activa.

PINGRESP: Esta es la respuesta del *broker* al mensaje Pingreq.

DISCONNECT: Este mensaje es enviado por el cliente para señalar su deseo de desconectarse. Al transmitirse, el cliente debe finalizar la conexión de red.

3.3. Encabezado MQTT

El encabezado del mensaje MQTT [18] es su primer *byte* y lo que le permite al *broker* manejarlo y distribuirlo correctamente a los suscriptores. Es la parte fija de un mensaje MQTT, que contiene información sobre el tipo de mensaje que es, el tema, el nivel de calidad de servicio (QoS) y sus banderas. El encabezado MQTT está seguido por la carga útil de longitud variable, que contiene los datos que se están transmitiendo, e incluye los siguientes campos:

Tipo de mensaje: Este es un campo de 4 *bits* que especifica el tipo de mensaje que se está transmitiendo, como un mensaje de publicación, un mensaje de suscripción o un mensaje de acuse de recibo.

Bandera de entrega duplicada: Este es un campo de 1 *bit* que indica si el mensaje que se está transmitiendo es una duplicación de un mensaje previamente transmitido. Sólo se utiliza para mensajes de nivel de QoS 1 y 2.

Nivel de calidad de servicio (QoS): Este atributo, que ocupa un campo de 2 *bits*, denota el estándar de garantía para la entrega de un mensaje particular. Especifica el grado de garantía de entrega del mensaje, ofreciendo tres niveles distintos.

Bandera de retención: Esta es una bandera de 1 *bit* que indica si el *broker* debe retener el mensaje para su entrega posterior a nuevos suscriptores.

Longitud del tema: Este campo especifica la longitud del campo del tema en *bytes*.

Identificador de mensaje: Este campo se utiliza para identificar mensajes y rastrear su entrega. Solo está presente en mensajes con niveles de QoS superiores a 0.

3.4. Mecanismo Keep Alive en MQTT

MQTT funciona sobre el protocolo TCP, el cual es orientado a la conexión [70] y brinda un flujo estable y ordenado de *bytes* entre dos entidades conectadas.

Sin embargo, en algunos casos, TCP puede tener problemas de conexión a medias lo que significa que una conexión que ha sido desconectada o no establecida en un lado, mientras que en el otro lado la conexión sigue activa. En este caso, la parte conectada puede enviar datos continuamente, los cuales, como es obvio, nunca llegarán al otro lado. Para evitar los vacíos de comunicación causados por las conexiones a medias, el protocolo MQTT proporciona un mecanismo de *Keep Alive* [18] que permite al cliente y al servidor MQTT determinar si hay un problema de conexión a medias y cerrar el enlace correspondiente.

Cuando un cliente MQTT establece una conexión con el *broker* MQTT, el mecanismo de *Keep Alive* puede activarse entre las partes en comunicación configurando el campo *Keep Alive* en el paquete CONNECT con un valor distinto de cero. *Keep Alive* es un número entero entre 0 y 65,535 que representa el tiempo máximo, en segundos, permitido entre los paquetes MQTT enviados por el cliente.

Cuando el *broker* recibe una solicitud de conexión de un cliente, verifica el valor del campo *Keep Alive* en la cabecera variablesi hay un valor, el *broker* activará el mecanismo de *Keep Alive*.

Después de establecer la conexión, el cliente debe asegurarse de que el intervalo entre dos paquetes MQTT que envíe no supere el valor de *Keep Alive*. Si el cliente está inactivo y no tiene paquetes para enviar, puede enviar paquetes PINGREQ en su lugar.

Cuando el cliente envía un paquete PINGREQ, el *broker* debe responder con un paquete PINGRESP, si este no se recibe dentro de un intervalo razonable, significa que hay una conexión a medias, que el *broker* está fuera de línea o que hay una falla en la red, por lo que el cliente debe cerrar la conexión.

En el caso del *broker*, si este no recibe ningún paquete del cliente dentro de 1.5 veces el tiempo de *Keep Alive*, asumirá que hay un problema con la conexión del cliente y procederá a desconectarlo.

Si el *broker* recibe un paquete de protocolo PINGREQ del cliente, debe responder con un paquete de protocolo PINGRESP para confirmar la conexión.

3.5. Seguridad en MQTT

MQTT está diseñado para ser eficiente y ligero, es un protocolo de comunicación IoT popular utilizado en hogares inteligentes y sistemas IoT industriales. Sin embargo, al igual que cualquier protocolo de comunicación, puede ser vulnerable a varios tipos de ataques [19]. Asegurar las comunicaciones MQTT y aplicar diversas medidas de seguridad son esenciales para prevenir amenazas y ataques de seguridad. Algunos ataques comunes a MQTT en IoT incluyen:

Ataque de Espionaje: Significa la interceptación ilícita de datos que se transfieren entre un dispositivo cliente (como un sensor o un dispositivo IoT) y un *broker* (la entidad que gestiona las comunicaciones MQTT). El agresor se mete en el canal de comunicación y descifra los mensajes intercambiados entre el cliente y el *broker*. Esta acción podría llevar a la adquisición de información confidencial, incluyendo credenciales de usuario o datos del sensor.

Ataque de Hombre en el Medio (MitM): Se dirige a situaciones en las que un malhechor intercepta y modifica la transmisión que tiene lugar entre un dispositivo cliente (como un sensor) y un *broker*.

Suplantación de identidad: Se refiere a un escenario en el que un agresor se hace pasar por un dispositivo cliente legítimo o un *broker* para manipular la comunicación entre ambos. El

atacante envía mensajes al *broker* o al cliente que parecen provenir de una fuente confiable, pero en realidad no lo son.

Ataque de Denegación de Servicio (DoS): Se refiere a la situación en la que un agresor interrumpe el funcionamiento normal de un *broker* abrumándolo con un gran volumen de solicitudes, lo que resulta en una interrupción completa o parcial del servicio, pudiendo inducir a una suspensión generalizada en la funcionalidad de los sistemas IoT.

Ataque de Inyección: Se refiere a un ataque en el que se manipulan los datos que se transmiten entre un dispositivo cliente y un *broker* al inyectar cargas útiles maliciosas en el canal de comunicación, provocando que los dispositivos funcionen de manera incorrecta o inesperada. En un entorno IoT, el atacante puede alterar las lecturas de los sensores para causar daños o interrupciones en el funcionamiento del sistema IoT, o modificar los contenidos de los paquetes de autenticación para comprometer la seguridad del sistema.

Sniffing: Se refiere a un ataque en el que un agresor intercepta el tráfico MQTT y escucha la comunicación entre un dispositivo cliente y un *broker*, con el fin de obtener información confidencial, como contraseñas, nombres de usuario u otros datos sensibles.

Ataque de fuerza bruta en MQTT: Se refiere al intento de obtener acceso ilícito a un *broker* MQTT probando repetidamente diversas claves o contraseñas. [20].

3.6. MQTT versión 5.0

MQTT actualmente se ha convertido en un protocolo popular para conectar dispositivos de Internet de las cosas (IoT) a la nube, no obstante, se desarrolló originalmente en 1999 para monitorear oleoductos y gasoductos a través de redes satelitales, en ese momento, se requería de un protocolo que usara la mínima cantidad de energía posible, que tuviera un uso eficiente del ancho de banda y que pudiera operar a través de conexiones de red poco confiables. Cuando se desarrolló MQTT, el término IoT no se había acuñado, la computación en la nube no existía y el conjunto diverso de aplicaciones para IoT no había surgido.

Por estos motivos, era necesario actualizar el protocolo MQTT para abordar algunas de las características faltantes necesarias para alojarlo en plataformas en la nube a gran escala y para manejar aplicaciones IoT adicionales. En 2015/2016, se comenzó a trabajar dentro de OASIS en

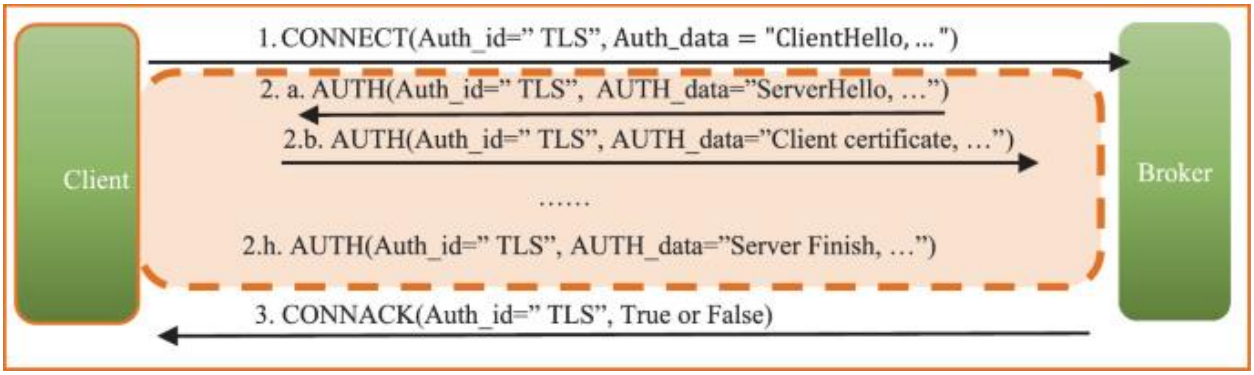

Figura 3.3 Autenticación Mejorada en MQTT v5

una nueva versión de la especificación, denominada MQTT v5.0. En marzo de 2019, MQTT v5.0 fue ratificado como estándar oficial de OASIS.

MQTT v5.0 introdujo numerosas mejoras al protocolo, aunque de todas ellas, la única relevante para nuestra propuesta es la Autenticación Mejorada.

3.6.1.Autenticación Mejorada en MQTT v5

La autenticación mejorada es un nuevo marco de autenticación introducido en MQTT v5.0. Ofrece una variedad de métodos alternativos que son más seguros que la autenticación tradicional mediante contraseña. Sin embargo, una mayor seguridad conlleva una mayor complejidad. Ciertos métodos de autenticación, como SCRAM (*Salted Challenge Response Authentication Mechanism*), requieren múltiples intercambios de datos de autenticación. Esto hace que el marco de autenticación de un solo intercambio de los paquetes CONNECT y CONNACK quede obsoleto. Para abordar esta limitación, MQTT 5.0 introduce el paquete AUTH, que admite múltiples intercambios de datos de autenticación. Esto permite el uso de mecanismos SASL (*Simple Authentication and Security Layer*) con un estilo de desafío-respuesta en MQTT. La figura 3.3 muestra la pila de protocolos del marco de la autenticación mejorada.

Antes de adentrarse en la autenticación mejorada, es esencial comprender las deficiencias de la autenticación mediante contraseña en términos de seguridad.

De hecho, a pesar de emplear técnicas como *salt* y *hash* para almacenar contraseñas de forma segura, el cliente debe transmitir la contraseña en texto plano a través de la red, lo que la hace vulnerable al robo. Incluso al emplear cifrado TLS (*Transport Layer Security*) para la comunicación, sigue existiendo el riesgo de que los atacantes obtengan datos sensibles como

contraseñas debido a versiones obsoletas de SSL (*Secure Sockets Layer*), suites de cifrado débiles o la presencia de certificados de CA falsos.

Además, la autenticación simple mediante contraseña sólo permite que el servidor verifique la identidad del cliente, pero no al revés, lo que permite al atacante hacerse pasar por el servidor y obtener datos sensibles del cliente. Esto es lo que a menudo llamamos un ataque de intermediario.

La autenticación mejorada permite a los usuarios verificar su identidad usando métodos altamente seguros dentro del marco SASL. Estos métodos ofrecen varias ventajas, como eliminar la transmisión de contraseñas a través de la red y facilitar la verificación mutua de identidades entre el cliente y el servidor. Al presentar estas opciones, los usuarios pueden seleccionar el método de autenticación que se alinee con sus necesidades específicas y preferencias de seguridad.

Capítulo 4

4. Trabajo relacionado

En este capítulo, realizamos un examen exhaustivo de la literatura existente relacionada con nuestro tema de investigación. Nos adentramos en varios estudios, artículos de investigación y fuentes académicas que discuten diversos mecanismos de autenticación en redes MQTT-IoT. Además, exploramos las fortalezas y limitaciones de diferentes enfoques, incluidas las técnicas de conjunto, para abordar los desafíos de ciberseguridad en este contexto.

MQTT v3.1 cuenta con características de seguridad predeterminadas [21], Para la autenticación proporciona un método simple mediante nombre de usuario y contraseña para conectar un cliente al *broker*. Los datos de autenticación se envían en texto plano sin ningún tipo de cifrado, por lo tanto, los desarrolladores pueden implementar sus propios mecanismos de seguridad en la capa de red y transporte e incluso se puede proporcionar en las capas inferiores.

En los últimos años, mucha investigación se ha enfocado en estos problemas de seguridad del protocolo MQTT, sin embargo, muchos de ellos tienen dificultades para ser efectivos en contextos donde las restricciones de recursos computacionales o de almacenamiento son altas. Las comunicaciones inalámbricas con objetos conectados están altamente expuestas a *sniffers*. Para que los procesos de control de acceso y autenticación sean eficientes para estos objetos proponemos las técnicas de OTP y autorización.

Verma [22], propone un modelo de seguridad OAuth para conexiones con el *broker,* en el que el proceso de inicio de sesión del cliente comienza enviando una solicitud de *token* de acceso al servidor de autorización, una vez que un cliente ha obtenido el *token,* puede enviarlo al *broker* a través del mensaje CONNECT. Cuando el *broker* recibe el *token,* puede verificar su fecha de vencimiento, la validez de su firma y revocarlo a través del servidor de autorización. El núcleo de este sistema de autenticación es el servidor de autorización centralizado. Por lo tanto, en caso de desincronización y corrupción de este núcleo, todo el sistema se vería comprometido porque los clientes legítimos serán bloqueados en la etapa de autenticación.

Sahmi *et al.* [23] proponen otro enfoque para fortalecer el mecanismo de autenticación de MQTT v3.1 utilizando el protocolo AugPAKE (*Augmented Password-Authenticated Key Agreement*) [50] que se basa en la modificación del protocolo de intercambio de claves de Diffie-Hellman. El cliente calcula una clave a partir de la contraseña para autenticarse ante el servidor, el cual no necesita almacenar las contraseñas de los clientes. Después de una autenticación exitosa, se establece una sesión segura y se realiza un acuerdo de clave entre el cliente y el servidor. Los pasos de intercambio para la gestión de autenticación no son compatibles con el enfoque ligero de MQTT.

Badis Hammi *et al.* [24] propusieron un algoritmo de intercambio de claves Elíptico de Diffie-Hellman Efímero (ECDHE) basado en curvas elípticas. Esta es una adaptación del protocolo de intercambio de claves de Diffie-Hellman (DH) [25] que permite que dos entidades establezcan un secreto compartido para minimizar la longitud de la clave y mejorar el rendimiento. Los autores proponen utilizar el algoritmo de autenticación de Clave Precompartida (PSK) [26] junto con el algoritmo de intercambio de claves ECDHE, cuya ventaja es que evita el cómputo costoso de la clave pública para la autenticación. El único problema es que, si el atacante puede obtener la clave secreta compartida, las sesiones anteriores y futuras se verán comprometidas. Cuando el mecanismo propuesto utiliza PSK con el algoritmo de intercambio de claves ECDHE, proporciona la característica de Secreto Adelantado Perfecto o PFS (*Perfect Forward Secrecy*) que protege las sesiones pasadas de futuros riesgos al proporcionar una clave separada para cada sesión. Amnalou et al. [27] y Yusoff *et al.* [28] también han confiado en las ventajas de las curvas elípticas para el fortalecimiento del protocolo MQTT. Sin embargo, los pasos y recursos requeridos por este mecanismo son un obstáculo para un sistema IoT limitado en recursos de cómputo y almacenamiento.

Mohamed Tahar Hammi *et al.* [29] [30] implementan el principio de OTP en un mecanismo personalizado de gestión de claves para la autenticación mutua de nodos en redes de sensores inalámbricos. Primero, un nodo hace una solicitud de asociación, luego recibe una solicitud de autenticación, genera y envía un OTP, y finalmente es autenticado por el Coordinador de Red de Área Personal (CPAN) si es legítimo. El trabajo de Esfahani *et al.* [31] y el del Laboratorio de Ciencias de la Ingeniería de la Universidad Ibn Tofail *et al.* [32] siguen prácticamente la misma lógica que Hammi y aportan bloques de seguridad en redes de sensores inalámbricos. Sin embargo, la técnica de gestión de claves utilizada en este algoritmo de autenticación requiere bastante

potencia de cálculo y, por lo tanto, es un obstáculo para nuestros dispositivos porque completa el proceso de autenticación sólo después de varios intercambios en el canal.

Bhawiyuga *et al.* [33] implementan una solución para dispositivos de gama baja, pero su evaluación se limita a la usabilidad y el tiempo de respuesta del *Servidor de Autorización* (SA). Calabretta *et al.* [34] proponen proteger los temas MQTT basados en el protocolo AugPAKE, sin embargo, no se proporciona un análisis de rendimiento. Collina *et al.* [35] proponen QEST, un *broker* MQTT RESTful y exploran cómo incorporar soporte para OAuth en el sistema MQTT, pero no se proporciona una implementación y evaluación. Niruntasukrat *et al.* [36] introducen un sistema basado en OAuth, donde tanto el *broker* como los dispositivos clientes tienen credenciales OAuth integradas. Dado que los canales de comunicación se consideran inseguros, los clientes generan una nueva firma en cada solicitud, lo que complica el sistema y requiere un viaje adicional al SA, lo que es costoso en recursos. Además, el uso de canales inseguros significa que el sistema es propenso a escuchas y ataques de intermediarios. Todas estas propuestas consideran una tercera entidad para realizar la autenticación, mientras que la nuestra sólo requiere del cliente y el *broker*.

La Marra *et al.* [37] proponen el Control de Uso o UCON (*Usage Control*) [49] para mejorar la seguridad de MQTT, aunque asumen que las conexiones ya están autenticadas. Meena *et al.* [38] sugieren el cifrado basado en curvas elípticas de los mensajes MQTT, pero asumen que los temas y los suscriptores son conocidos por los *brokers*, lo cual es impracticable. Chiwariro y Rajendran [39] ofrecen un esquema de cifrado ligero para garantizar la confidencialidad de los datos de carga útil de los paquetes MQTT, pero pasa por alto la autenticación del cliente. De manera similar, Mathews y Gondkar [40] proponen el cifrado de la carga útil con paso directo al *broker* sin abordar la autenticación del cliente.

Chien [46], [47] basándose en funciones y composiciones *hash*, propone un sistema de autenticación cliente-servidor aprovechando el marco de Autenticación Mejorada de MQTT v5.0. Sin embargo, las funciones *hash* utilizadas requieren una potencia de cómputo considerable y no son compatibles con dispositivos de bajo poder computacional.

Elisée Toé et al. [51] plantean un modelo de autenticación basado en HOTP para MQTT v3.1, no obstante, su propuesta introduce un nuevo paquete CONNACT fuera del estándar para intercambiar información necesaria para la siguiente autenticación.

Finalmente, la plataforma EMQX [48] ofrece una solución de autenticación que utiliza el marco de Autenticación Mejorada de MQTT v5.0. Sin embargo, esta solución requiere del uso de su *broker* y sólo acepta el método de autenticación SCRAM (*Salted Challenge Response Authentication Mechanism*).

De acuerdo con nuestra investigación, hasta donde sabemos no existe ninguna implementación o evaluación en dispositivos de bajo poder de cómputo que aproveche las últimas características de la versión 5 de MQTT.

En la tabla 4.1 los pros y contras de los trabajos revisados comparados con nuestra propuesta:

Tabla 4.1
Resumen de trabajo relacionado

Trabajo	Año	Versión de MQTT	Esquema de seguridad	Pros	Contras
[22]	2022	3.1	OAuth	Estándar ampliamente probado	Requiere una tercera entidad (SA)
[23]	2021	3.1	AugPAKE / Present	Autenticación mutua, autorización, protección contra los ataques más comunes	Enfoque no compatible con el esquema ligero de MQTT. No presenta implementación.
[24]	2020	3.1	PSK/ECDHE	Evita el cómputo costoso para la llave pública.	Si un atacante obtiene la llave secreta compartida, todo el sistema se compromete
[27], [28]	2020, 2022	3.1	ECDHE	Agrega seguridad basada en ECC sin cambios en el protocolo MQTT	No compatible con sistemas de escasos recursos de memoria y almacenamiento
[29], [30]	2017, 2017	-	HOTP	Autenticación al nivel de la capa MAC	El esquema no se integra a MQTT.
[31], [32]	2017, 2021	-	HOTP	Buena relación seguridad/rendimiento	Esquema enfocado en WSN's y no MQTT. Costoso en recursos

Tabla 4.1
Resumen de trabajo relacionado

Trabajo	Año	Versión de MQTT	Esquema de seguridad	Pros	Contras
[33]	2017	3.1	Tokens	Solución creada para dispositivos de gama baja	Sólo evalúa el tiempo de respuesta del SA
[34]	2018	3.1	AugPAKE	Protege el acceso a los temas.	No proporciona análisis de rendimiento
[35]	2012	3.1	OAuth	Propone integrar OAuth al broker MQTT eliminando la necesidad de un SA externo	No proporciona implementación ni evaluación
[36]	2016		OAuth	Integra las credenciales OAuth a los clientes. Considera el caso de comunicación sobre canales inseguros	Esquema complicado. Costoso en recursos.
[37]	2017	3.1	Control de uso	Mejora la seguridad de MQTT mediante la adaptación del modelo UCON	Asume que la conexión está autenticada
[38]	2015	3.1	Cifrado de los mensajes	Garantiza integridad y confidencialidad de los mensajes	Asume que el broker conoce de antemano los temas y sus suscriptores.
[39], [40]	2018, 2019	3.1	Cifrado de la carga útil	Garantiza integridad y confidencialidad de los mensajes	No se aborda la autenticación de los clientes.

Tabla 4.1
Resumen de trabajo relacionado

Trabajo	Año	Versión de MQTT	Esquema de seguridad	Pros	Contras
[51]	2023	3.1	HOTP	Se integra a MQTT sin requerir una tercera entidad	Implementa un paquete fuera del estándar para el intercambio de la información de autenticación
[46], [47]	2021	5.0	Intercambio de llaves personalizado	Aprovecha completamente el marco “Enhaced Authentication” de MQTT v5.0	Función hash requiere de elevado poder de cómputo
[48]	2024	5.0	SCRAM	Solución comercial. Cuenta con soporte y documentación	Requiere licencia. Sin versión para microcontroladores
Nuestra propuesta		5.0	TOTP	Aprovecha completamente el marco “Enhaced Authentication” de MQTT v5.0. Protección contra los ataques más comunes. Se enfoca en dispositivos de gama baja.	La distribución del secreto compartido puede ser complicada. La desincronización de los relojes puede presentar un problema a la larga. El esquema TOTP sólo garantiza la autenticación. La integridad y la confidencialidad deben garantizarse con otros esquemas.

Capítulo 5

5. Arquitectura propuesta de TOTP para MQTT v5.0

En este capítulo, proponemos la arquitectura de un sistema de autenticación TOTP compatible con MQTT v5. Nuestra propuesta se divide en 2 partes principales:

1. La adaptación del algoritmo TOTP de modo que use una función *hash* criptográfica ligera en lugar de las recomendadas en el RFC (SHA1), con el objetivo de reducir la carga computacional en dispositivos de recursos limitados.
2. La integración del algoritmo antes citado en MQTT v5.0 usando la infraestructura "Enhanced Authentication" que forma parte del protocolo.

Antes de describir el protocolo, en la sección 5.1 presentamos nuestro razonamiento para el uso de TOTP en nuestra propuesta. Posteriormente, hacemos una revisión breve sobre el algoritmo HMAC y construcciones esponja en la sección 5.2 y sobre la autenticación mejorada de MQTT v5.0 en la sección 5.3

5.1. ¿Por qué TOTP?

El IoT depende de la cooperación de múltiples dispositivos y numerosos escenarios requieren que sólo los usuarios de confianza puedan utilizar los servicios ofrecidos, por lo tanto, los requisitos de seguridad convencionales, como la autenticación, la integridad de los datos y, en algunos casos, la confidencialidad, son fundamentales para el funcionamiento de los dispositivos, redes y aplicaciones del IoT. Sin embargo, debido a las limitaciones y la heterogeneidad de los recursos de los dispositivos, las soluciones de seguridad existentes no están completamente adaptadas al ecosistema del IoT. Además, para garantizar estos requisitos de seguridad, se necesita la combinación de múltiples tecnologías y soluciones de seguridad, lo que conlleva altos costos de computación (en general, se requiere una solución para la autenticación e integridad de los datos donde se aplique al menos un algoritmo asimétrico para el intercambio de claves de sesión, y otra solución para la confidencialidad donde se aplique un algoritmo de cifrado para el intercambio de

información). En consecuencia, es necesario proponer nuevas soluciones de seguridad ligeras o adaptar las ya existentes para coincidir con las capacidades del IoT.

Gran parte de los esquemas de autenticación disponibles dependen de la arquitectura específica del sistema IoT. Además, la mayoría de ellos requiere una gestión local de claves y necesitan una infraestructura para almacenarlas, lo que los hace vulnerables al robo de contraseñas. Por último, la mayoría de ellos se basa en un esquema de autenticación de un solo factor, lo que puede ser un riesgo de seguridad en dicho entorno.

El OTP es un esquema de autenticación de múltiple factor en el que se genera una nueva contraseña para cada sesión y no es posible reutilizar la contraseña. El OTP es una de las soluciones más prometedoras para la autenticación en IoT pues es segura y fácil de automatizar. Sin embargo, existen muy pocos trabajos que hayan adaptado esquemas de OTP a MQTT, debido a que MQTT v3.x no proporciona la infraestructura para realizar el intercambio de información requerido por el esquema sin modificar el protocolo. Este problema queda resuelto con MQTT v5.0 y su marco "*Enhanced Authentication*".

Dentro de las implementaciones de OTP, las que usan HOTP tienen el problema de que la contraseña se mantiene hasta la siguiente autenticación (lo que podría tomar un largo tiempo); esto lo vuelve vulnerable a ciertos ataques como el *phishing.* El esquema TOTP mitiga este problema asignándole a cada periodo una contraseña diferente, limitando así la ventana de uso de la contraseña y protegiendo a los usuarios de dichos ataques. Es por esto por lo que en este trabajo optamos por usar TOTP como nuestra propuesta de solución.

5.2. HMAC y funciones esponja

El código HMAC (*keyed-hash message authentication code*), también conocido como código de autenticación de mensajes basado en *hash*, es un componente fundamental utilizado para garantizar la integridad de los datos en la seguridad de redes modernas. Protocolos de comunicación como SSL, TLS, IPSec (*Internet Protocol Security*), y muchos protocolos de transferencia de archivos como FTPS (*FTP Secure*), SFTP (*SSH File Transfer Protocol*) y HTTPS (*Hypertext Transfer Protocol Secure*) utilizan HMAC durante la transmisión y recepción de datos como un medio de verificación [68]. Aunque las funciones *hash* tradicionales como MD5, SHA-0, SHA-1, y SHA-2 se utilizan para verificar la integridad de los datos, no pueden establecer su autenticidad por sí solas. Por lo tanto, surge la necesidad de los HMAC, frecuentemente empleados

para establecer una medida de autenticidad. Un diagrama simple que describe HMAC se muestra en la figura 5.1, donde dos partes están transmitiendo y recibiendo un mensaje a través de un canal de comunicación posiblemente inseguro. La autenticidad de un mensaje se establece mediante un resumen *hash* que combina una clave secreta con su contenido, en lugar de utilizar únicamente el *hash* del propio contenido. Al emplear protocolos de comunicación con HMAC, el receptor puede detectar si la información ha sido alterada y no autenticada por el remitente, lo que advierte sobre una posible manipulación de datos. Este mecanismo garantiza la autenticidad entre dos partes a través de una clave secreta compartida ("*key*" en la figura), generada durante el intercambio de

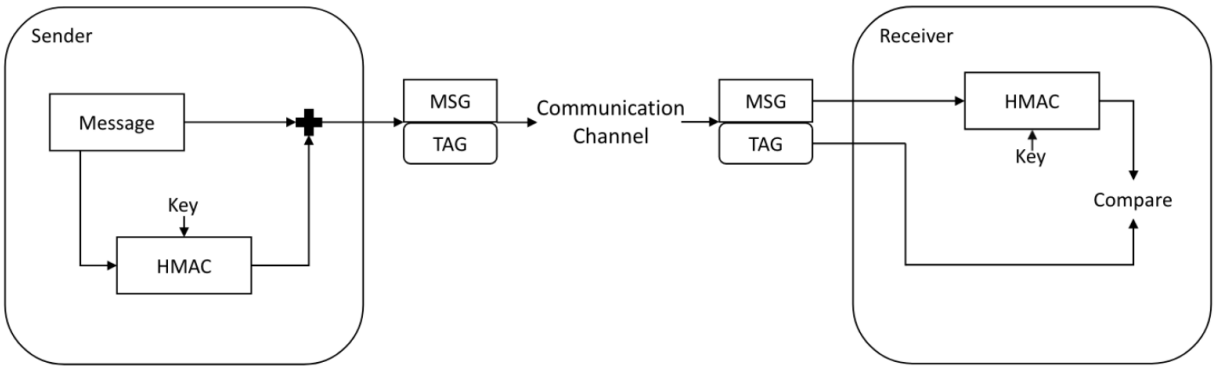

Figura 5.1. Descripción general de HMAC

claves y utilizada junto con los datos transmitidos.

Las funciones criptográficas de *hash* usuales como MD-5, SHA-1 y SHA-2 utilizan una construcción de tipo Merkle-Damgård [69]. Por su parte, las funciones de *hash* de última generación como SHA-3, Photon y QUARK utilizan una "construcción de esponja" que consta de tres fases distintas: relleno, absorción y exprimido. Inicialmente, se "rellena" el mensaje de entrada de modo que su longitud sea un múltiplo del tamaño del bloque de absorción, luego el algoritmo "absorbe" *bits* del mensaje en el estado del *hash*, luego se "exprime" una salida de longitud igual. El estado del *hash* tiene una longitud de 1600 *bits* y se representa como un cubo de 5 × 5 × 2^l, donde l = 6. Las construcciones de esponja utilizan una única permutación en lugar de una función de compresión, como se usa en la construcción Merkle-Damgård. Usan además operaciones XOR *bit* a *bit* (⊕) para mezclar los *bits* del mensaje en lugar de cifrados en bloque, como se utiliza en las funciones de *hash* que emplean la construcción Merkle-Damgård.

La figura 5.2 muestra la construcción esponja general. Inicialmente, la función de esponja realiza una operación XOR a nivel de *bits* del primer bloque de mensaje, m_0, con el vector de inicialización predefinido (una cadena de ceros). Luego, la función de permutación *P*, transforma el valor del estado interno a otro valor del mismo tamaño. La función aplica XOR al segundo bloque de mensaje, m_1, con la salida de la primera función de permutación y luego aplica al resultado la función de permutación *P*. Esto continúa hasta que todos los bloques de mensaje se absorben en la esponja. En la fase de exprimido, la función de esponja aplica *P* para extraer un bloque z_i de bits y formar el *hash*. El número de *bits* de salida se elige a voluntad por el usuario.

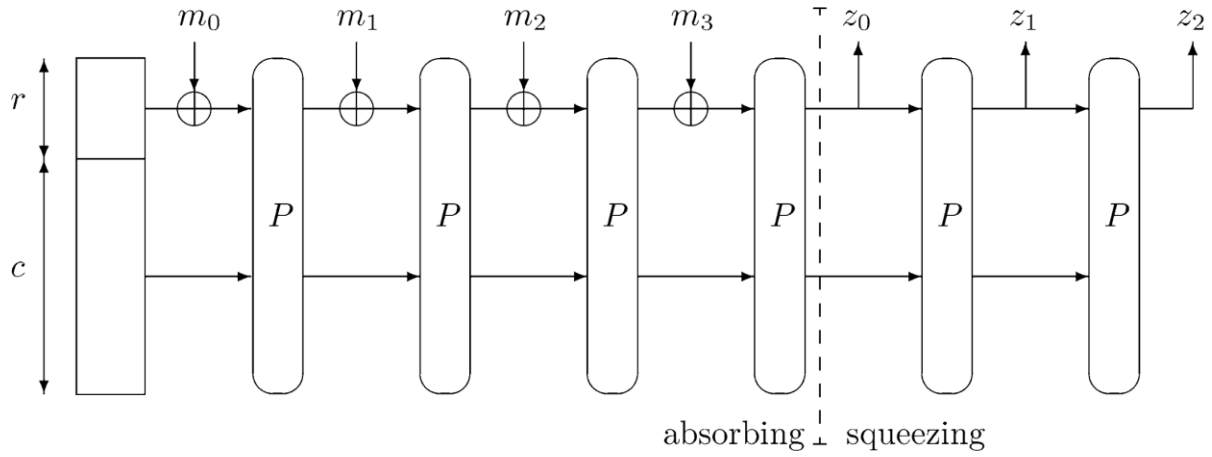

Figura 5.2. La construcción esponja utilizada por QUARK

La seguridad de las funciones de esponja depende de las longitudes del bloque de mensaje y del estado interno. Si cada bloque de mensaje tiene una longitud de *r bits* y la longitud del estado interno es de *w bits*, entonces los $c = w - r$ *bits* no se modifican al aplicar XOR con los bloques de mensaje. El valor *c* se llama la *capacidad* de la esponja. Por lo tanto, el nivel de seguridad garantizado por la función de esponja dentro de la computación clásica es *c/2*. La función de permutación *P* dentro de las funciones de esponja se comporta como una permutación aleatoria sin ninguna estructura matemática o sesgo estadístico que permita a un adversario predecir la salida. [42]

El algoritmo HMAC diseñado toma dos entradas juntas para producir los códigos de autenticación de mensajes: un secreto compartido y un mensaje. Este algoritmo reemplaza al

algoritmo clásico que usa SHA1 [44] en nuestra implementación de TOTP. Aunque no se realiza un análisis de la seguridad del algoritmo resultante por ir más allá de los alcances de este trabajo, el estudio de Naito *et al.* demuestra que reemplazar la función *hash* subyacente de una función HMAC por una de tipo esponja mejora la seguridad genérica del algoritmo. [45]

5.3. Autenticación Mejorada en MQTT v5.0

El marco de Autenticación Mejorada de MQTT v5.0 proporciona nuevas APIs (y mensajes) AUTH y nuevos campos relacionados con la autenticación llamados *Authentication Method y Authentication Data* (se referirán como *Auth_method* y *Auth_data* en el resto de este documento para abreviar). Así se puede diseñar e implementar cualquier esquema de autenticación que requiera intercambio de claves dentro del contexto de MQTT v5.0.

Un cliente (un editor o un suscriptor) inicia su conexión enviando una solicitud CONNECT en la cual se especifican *Auth_method* y *Auth_data* para notificar al *broker* qué método de autenticación se ha elegido. Durante el proceso de autenticación, pueden intercambiarse varios mensajes AUTH que transmiten *Auth_data* para el *Auth_method* especificado. Finalmente, se envía un CONNACK para notificar el resultado del proceso. Durante el cual, *Auth_method* debe incluirse en cada mensaje para asegurar que se refiere al método correcto.

5.4. El esquema propuesto en el contexto de MQTT v5.0

Basándonos en el marco de la Autenticación Mejorada de MQTT v5.0 presentamos nuestra propuesta de un esquema de autenticación basado en TOTP usando una función HMAC basada en U-QUARK.

En la inicialización, el secreto compartido se registra en el cliente y en el *broker,* asimismo, se sincronizan los relojes. El esquema para lidiar con una posible desincronización se explica en la sección 2.3.6

El flujo de nuestro esquema de autenticación en el contexto de MQTT v5.0 se muestra en la figura 5.3. En el primer paso, el cliente envía al *broker* un mensaje CONNECT con *auth_method*

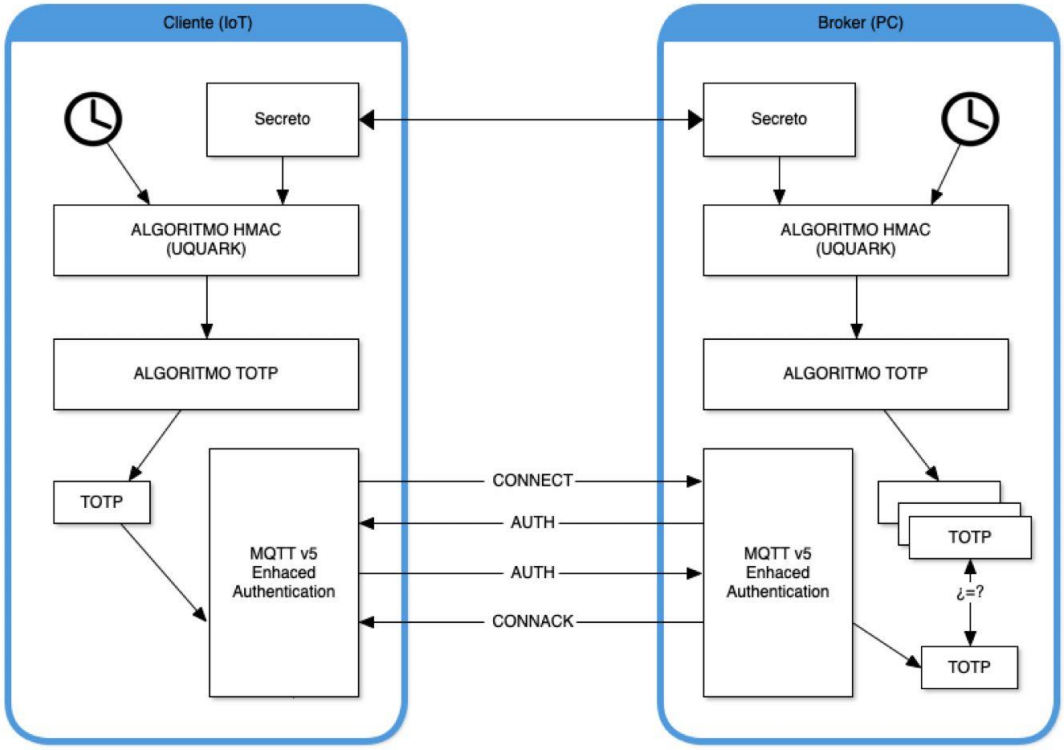

Figura 5.3. La arquitectura propuesta

= *"TOTP"* denotando el método de autenticación que desarrollamos.

En el mensaje CONNECT, el cliente también envía su identificador único. Cuando el *broker* recibe la solicitud, genera un número aleatorio que usaremos como *nonce* [55] y en el segundo paso responde con un mensaje AUTH que contiene a su vez el *auth_method = "TOTP", auth_data = nonce* y el código de respuesta *reason code* = "*Continue authentication"* (18).

En el tercer paso, si la verificación del *auth_method* tuvo éxito, el cliente calcula el TOTP correspondiente usando el secreto compartido, el tiempo y el *nonce* y responde con otro mensaje AUTH que contiene a*uth_method* = "TOTP", *auth_data* = "XXXXXXXXX" (el TOTP de 9 dígitos calculado) y el código de respuesta *reason code* = "*Continue authentication*" (18).

Finalmente, cuando el *broker* recibe el mensaje anterior, calcula los TOTP´s correspondientes a los tres más recientes pasos de tiempo y los compara con el TOTP recibido. Si alguna de las comparaciones es exitosa, el *broker* responde con un mensaje CONNACK con *auth_method = "TOTP"* y el código de respuesta *reason code* = "*Success*" (18), en caso contrario, el código de respuesta será "*Not authorized"* (87). La figura 5.4 muestra con detalle el intercambio

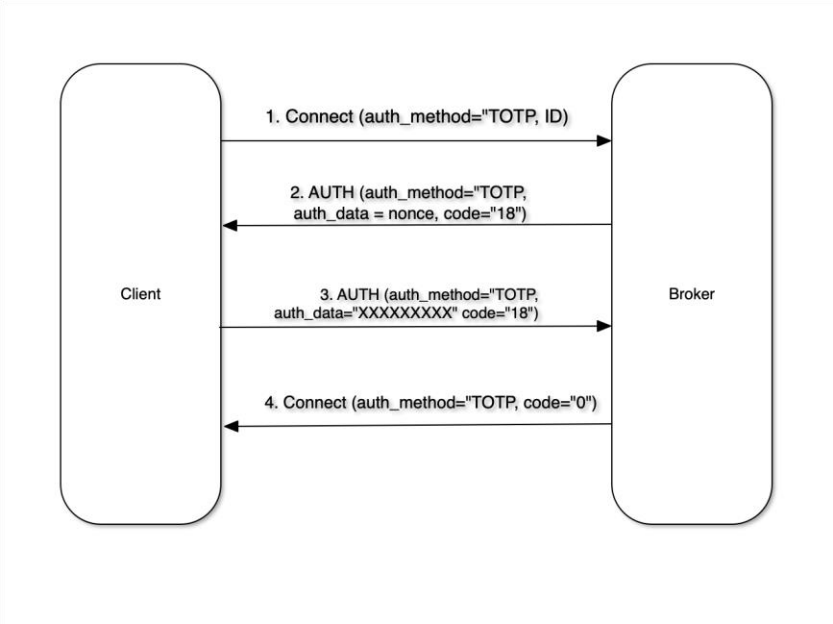

Figura 5.4. Intercambio de mensajes en el contexto MQTT v5.0

de mensajes en el marco de la Autenticación Mejorada de MQTT v5.0

Una vez que el intercambio de mensajes se completó con éxito, la sesión MQTT se considera iniciada y el cliente podrá entonces suscribirse a los temas y recibir mensajes mientras la sesión permanezca activa.

Capítulo 6

6. Implementación y resultados

En este capítulo, presentamos las herramientas de *hardware* y *software* usadas en la implementación de nuestra propuesta y el diseño de cada una de sus partes. Posteriormente se presentan los resultados obtenidos en las pruebas realizadas.

6.1. Implementación

6.1.1. Marco Experimental

En esta sección, presentamos el *hardware* y el *software* utilizados en la implementación de nuestra propuesta.

6.1.1.1 ESP32

ESP32 [52] es una familia de microcontroladores potentes y económicos para el desarrollo de aplicaciones de IoT; fueron desarrollados por la empresa *Espressif Systems* (Shanghái, China) y ofrece una poderosa combinación de funciones y capacidades para aplicaciones de IoT.

Los microcontroladores ESP32 tienen las siguientes características: un procesador de doble núcleo, conectividad integrada *Wi-Fi* y *Bluetooth*, un gran número de pines de entrada/salida de propósito general (GPIO) y bajo consumo de energía. Además, están equipados con un microprocesador dual-core Xtensa LX7, que proporciona mayor potencia de procesamiento y facilita la multitarea y la ejecución eficiente de tareas complejas. Cuentan con interfaces *Wi-Fi* y *Bluetooth* integradas que simplifican la conexión y la comunicación con otros dispositivos o redes. Soporta varios protocolos *Wi-Fi*, como 802.11 b/g/n, y proporciona opciones de conectividad *Bluetooth Classic* y *Bluetooth Low Energy* (BLE). Ofrece modos de suspensión y características de gestión de energía que ayudan a reducir el consumo de energía, haciéndolo adecuado para proyectos alimentados por baterías o con restricciones energéticas.

Los microcontroladores ESP32 se pueden conectar a pantallas convencionales, táctiles o indicadores LED para proporcionar a los operadores o al personal una interfaz fácil de usar.

Aunque se pueden programar utilizando varios *frameworks* de desarrollo, el IDE de Arduino no es recomendable para aplicaciones que quieran llevarse a un estado de producción debido a sus limitaciones tales como que no admite programación modular u orientada a objetos, lo que puede dificultar la organización o reutilización del código. Tampoco permite multitarea o concurrencia, lo que puede limitar su capacidad para manejar múltiples procesos o eventos. Además, Arduino IDE no proporciona ninguna herramienta o método para garantizar la calidad, fiabilidad o seguridad de su código, como análisis de este, verificación o validación. Por otra parte, el lenguaje de programación más utilizado es C++.De tal modo que las herramientas utilizadas por la industria para el desarrollo son *PlatformIO* y *Visual Studio* basados en las librerías provistas por el ESP-IDF (*Espressif IoT Deve1opment Framework*).

En cuanto a la disponibilidad de tarjetas para el desarrollo de aplicaciones, en el mercado existen diferentes módulos basados en los chips ESP32. Algunos módulos integran sensores, otros cámaras de video e incluso se pueden encontrar tarjetas con *hardware* de comunicación *Ethernet*, Lora y otros. Esta amplia variedad simplifica de manera importante el desarrollo de novedosas aplicaciones relacionadas con los IoT.

En la industria, los microcontroladores ESP32 son utilizados esencialmente por empresas emergentes para muy diversas aplicaciones. La empresa NORVI [66] ofrece cámaras para aplicaciones de inteligencia artificial, así como otros novedosos productos que incluyen los PLC y sistemas de sensores. Por otra parte, la empresa Sonoff [67] ofrece todo un catálogo de productos para la domótica a través de enlaces *Wi-Fi*. A pesar de que los ESP32 son dispositivos potentes y llaman la atención de diversos industriales, su uso en productos comerciales aún es bajo si la comparamos con procesadores de compañías como TI (*Texas Instruments*), *Silicon Labs* y Microchip, debido a que es un producto relativamente nuevo, a su poca disponibilidad en el mercado y que existen procesadores de otras compañías que presentan mejor documentación y mejor soporte técnico. En cambio, los ESP32 tienen como ventaja que sus sistemas de desarrollo son gratuitos, algo que no todas las otras compañías ofrecen.

Para este trabajo, decidimos utilizar el módulo S2 mini (figura 6.1), pues es una opción económica pero con la suficiente capacidad de cómputo para ejecutar nuestro algoritmo. El módulo incluye una conexión *micro-USB* que se puede utilizar para programación y/o alimentación. Además, el módulo incluye las siguientes características:

- Chip: ESP32-S2FN4R2
- Frecuencia: 240MHz
- Núcleos: 1
- ROM: 128kB
- SRAM: 320kB
- PSRAM: 2MB
- Memoria Flash: 4MB
- *Wi-Fi* integrado
- Número de pines GPIO: 27
- Número de pines PWM: 27 (máximo 8 simultáneos)
- Número de pines de entrada análoga (ADC): 18
- Número de pines de salida análoga (DAC): 2

El uso de MQTT en ESP32 ofrece varias ventajas:

Primero, MQTT es un protocolo de comunicación ligero optimizado para dispositivos y redes con recursos limitados, como los ESP32 y *Wi-Fi*, por lo que tiene un impacto mínimo en el consumo de energía y el ancho de banda.

Segundo, MQTT admite diferentes niveles de fiabilidad y calidad de servicio para adaptarse a las capacidades de los ESP32. Esta flexibilidad lo hace adecuado incluso cuando las redes son inestables.

Tercero, los ESP32 y MQTT son ampliamente utilizados en aplicaciones IoT, lo que permite que se integren bien en diversas soluciones tecnológicas. El protocolo MQTT también está

diseñado para simplificar la integración con plataformas en la nube, lo que permite el control de

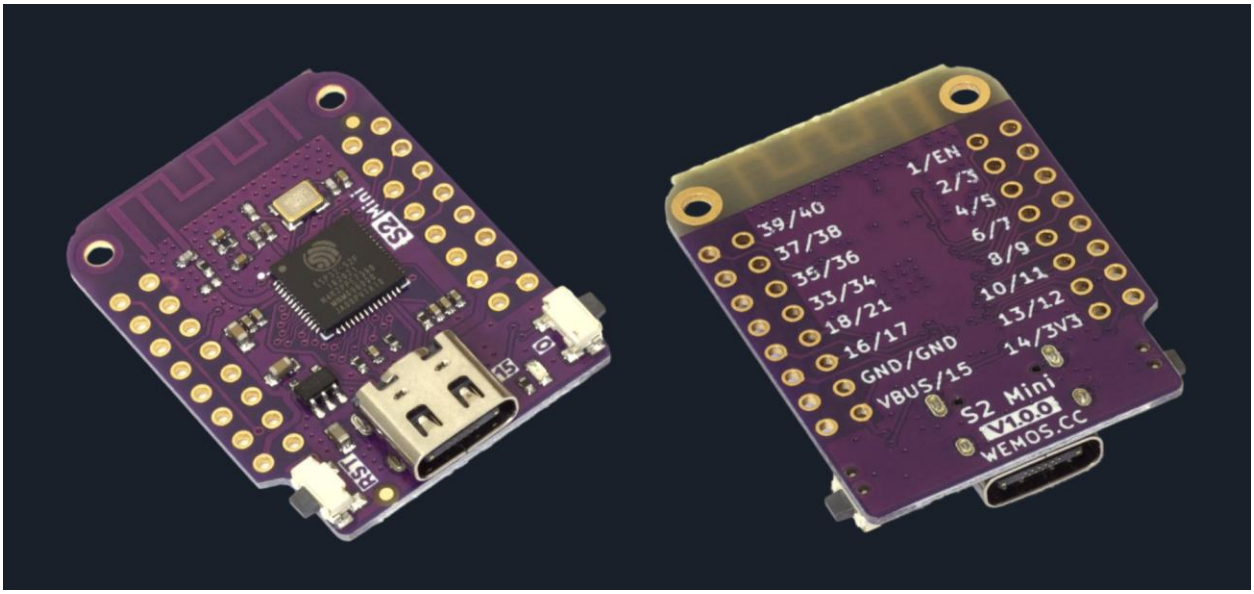

Figura 6.1. Módulo S2 Mini

dispositivos y el monitoreo de datos a través de redes.

En general, la combinación de microcontroladores ESP32 y MQTT es ideal para aplicaciones de IoT que requieren conectividad inalámbrica y comunicación eficiente entre múltiples dispositivos.

6.1.1.2 *Visual Studio Code* y ESP-IDF

Visual Studio Code [53] es un editor de código fuente gratuito, ligero, pero poderoso, que puede ejecutarse localmente y en la web, y está disponible para *Windows*, *macOS*, *Linux* y *Raspberry Pi OS*. Incluye soporte incorporado para *JavaScript*, *TypeScript* y *Node.js*, y cuenta con un enorme y variado ecosistema de extensiones para otros lenguajes de programación (como *C++*, *C#*, *Java*, *Python*, *PHP* y *Go*), entornos de ejecución (como *.NET* y *Unity*), entornos de desarrollo (como *Docker* y *Kubernetes*) y servicios en la nube (como *Amazon Web Services*, *Microsoft Azure* y *Google Cloud Platform*).

Además de ser liviano y arrancar rápidamente, *Visual Studio Code* tiene *IntelliSense* para el autocompletado de código en variables, métodos y módulos importados; depuración gráfica; *linting*, edición con múltiples cursores, sugerencias de parámetros y otras potentes funciones de

edición; elegante navegación y refactorización de código, y control de versiones integrado, incluido el soporte para *Git*. Gran parte de esto se adaptó de la tecnología de *Visual Studio*.

Visual Studio Code está construido utilizando el *shell* de *Electron*, *Node.js*, *TypeScript* y el Protocolo de Servidor de Lenguaje, y se actualiza mensualmente. Las numerosas extensiones se actualizan según sea necesario. La riqueza del soporte varía entre los diferentes lenguajes de programación y sus extensiones, abarcando desde un simple resaltado de sintaxis y coincidencia de corchetes, hasta depuración y refactorización. Si no hay un servidor de lenguaje disponible, se puede agregar soporte básico para nuestro lenguaje favorito a través de colorizadores de *TextMate*.

Una búsqueda rápida de las extensiones disponibles en el *Visual Studio Code Marketplace* arroja aproximadamente 38,000 resultados, que ofrecen soporte para cientos de lenguajes de programación. Estas extensiones se pueden gestionar desde el *Marketplace*, desde la barra lateral de Extensiones en *VS Code* y desde la Paleta de Comandos de *VS Code*. De todas estas opciones, la que nos interesa, por supuesto, es la extensión ESP-IDF.

ESP-IDF [54] (*Espressif IoT Development Framework*) es el *framework* oficial para las series de microcontroladores ESP32 y ESP8266, creado y mantenido por *Espressif Systems*. Es una plataforma de desarrollo completa y versátil que proporciona todas las herramientas, bibliotecas y APIs necesarias para desarrollar aplicaciones robustas y eficientes para los microcontroladores de *Espressif*. ESP-IDF está diseñado para trabajar de manera fluida con el ESP32 y el ESP8266, permitiendo a los desarrolladores aprovechar al máximo las capacidades de estos potentes microcontroladores.

Decidimos utilizar este entorno de desarrollo en vez de otro más sencillo como pudo ser Arduino IDE debido a que, al momento de implementar nuestra propuesta, no existía ninguna librería de Arduino que soportara MQTT v5.

La figura 6.2 muestra *VS Code* con la extensión ESP-IDF instalada.

6.1.1.3 Familia de funciones criptográficas QUARK

La familia de funciones criptográficas QUARK es un conjunto de funciones *hash* diseñadas para ser eficientes en términos de recursos, particularmente en entornos de *hardware* con limitaciones, como dispositivos embebidos o sistemas de Internet de las Cosas (IoT). Las principales características de QUARK incluyen:

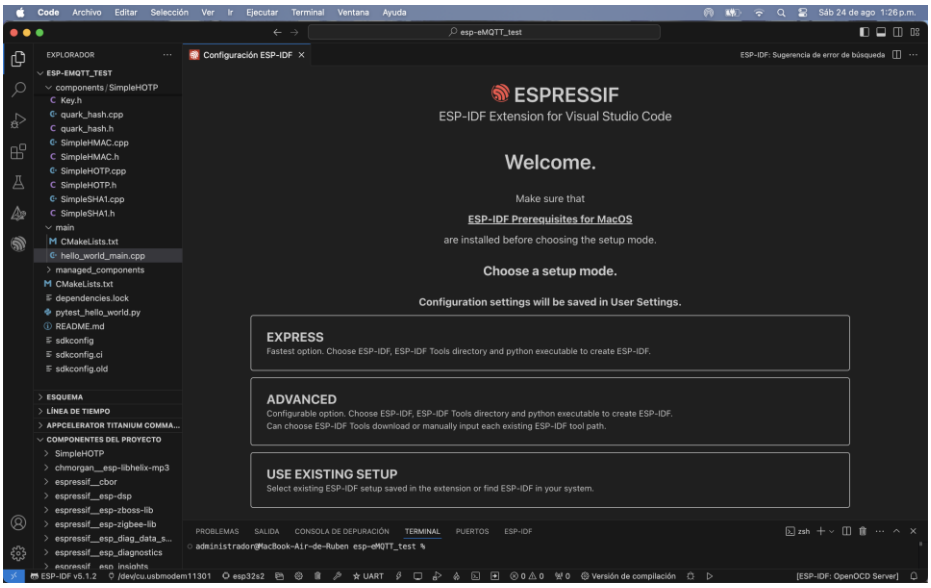

Figura 6.2. VS Code y la extensión ESP-IDF

Eficiencia: QUARK fue diseñado para ser muy ligero en términos de uso de almacenamiento y energía, por lo que es ideal para dispositivos con capacidades limitadas de memoria o procesamiento.

Tamaño de la salida: Las funciones *hash* de QUARK producen salidas de diferentes longitudes (rango comúnmente entre 128 y 256 bits), lo que permite flexibilidad en función de las necesidades de seguridad y eficiencia.

Tipos de QUARK: La familia está compuesta por tres variantes:

u-QUARK: Ofrece un tamaño de salida de 136 bits, la versión más ligera.

d-QUARK: Con un tamaño de salida de 176 bits.

t-QUARK: Ofrece la mayor seguridad, con un tamaño de salida de 256 bits.

Seguridad: Aunque QUARK fue diseñado con la eficiencia en mente, sigue manteniendo una robusta resistencia frente a ataques criptográficos comunes, como lo son los de colisión o preimagen [70]. Sin embargo, al ser una función ligera, puede tener menos margen de seguridad en comparación con otras funciones *hash* más pesadas como SHA-256.

Aplicaciones: Se usa principalmente en dispositivos IoT, donde la combinación de seguridad y eficiencia es crucial para proteger datos con un consumo mínimo de recursos.

En este trabajo, seleccionamos u-QUARK como base para la implementación de nuestra función HMAC ligera debido a la facilidad de implementación en *software* y a sus ventajas en cuanto ahorro de memoria y recursos reportados por sus autores. u-QUARK se describe en detalle en [43].

En la tabla 6.1 se presenta una comparación entre las diversas versiones de QUARK. Una comparación más detallada frente a otras funciones criptográficas ligeras, incluyendo otras de tipo esponja, se puede ver en [58].

Tabla 6.1
Comparación de las funciones de la familia QUARK

Funcion hash	Seguridad (bits)	Area (GE)	Velocidad (kbps)	Potencia (µW)
Arquitectura Compacta @100 kHz				
u-QUARK	64, 128	1379	1.47	2.96
d-QUARK	80, 160	1702	2.27	3.95
s-QUARK	112, 224	2296	3.13	5.53
Arquitectura de alta velocidad @714 kHz				
u-QUARK	64, 128	3032	84	37.01
d-QUARK	80, 160	3561	130	43.35
s-QUARK	112, 224	6220	357	75.27

6.1.1.4 Eclipse Mosquitto

Eclipse Mosquitto es un *broker* MQTT de código abierto que cumple con las licencias EPL/EDL y soporta las versiones 5.0, 3.1.1 y 3.1 de MQTT.

Mosquitto fue desarrollado inicialmente por Roger Light en 2009 y luego donado a la *Eclipse Foundation*. Probablemente fue el primer proyecto MQTT de código abierto.

El proyecto Mosquitto también proporciona una biblioteca en C para implementar clientes MQTT, así como los muy populares clientes MQTT de línea de comandos *mosquitto_pub* y *mosquitto_sub*.

Decidimos usar el *broker* Mosquitto por simplicidad y por el soporte ya existente para el manejo genérico de sistemas de seguridad. Al inicio de nuestra investigación, estudiamos diversas implementaciones en código abierto que pudieran usarse en este proyecto. Encontramos que tanto HiveMQ como Mosquitto eran apropiadas y cubrían con el requerimiento de soportar MQTT v5.0 y, en particular, contar con todas las características del marco *Enhanced Authentication*.

Comparamos las especificaciones de cada una de estas implementaciones y ambas tenían pros y contras aunque finalmente el factor decisivo fue el lenguaje en el que están escritas. Tanto HiveMQ como Mosquitto cuentan con un SDK (*Software Development Kit*) que permite agregar funcionalidades adicionales al *broker* sin modificar el código fuente mediante el uso de *plugins*. Sin embargo, los *plugins* para HiveMQ se escriben en Java mientras que los de Mosquitto se escriben en C. Dado que el resto de los componentes de nuestro proyecto también están programados en C, se optó por usar el *broker* Mosquitto para mantener la coherencia de lenguajes.

La tabla 6.2 muestra una comparación de los *brokers* considerados.

Tabla 6.2
Brokers considerados

Broker MQTT	Mosquitto	Beviwyse	ActiveMQ	HiveMQ	VerneMQ	EMQX
Código abierto	Sí	No	Sí	Sí	Sí	Sí
Escrito en	C	C, Python	Java	Java	Erlang	Erlang
Versión MQTT	3.1.1, 5.0	3.x, 5.0	3.1	3.x, 5.0	3.x, 5.0	3.1.1
Soporte QoS	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2
Sistema operativo	Linux, Mac, Windows	Windows, Windows server, Linux, Mac and Raspberry Pi	Windows, Unix/ Linux/Cygwin	Linux, Mac, Windows	Linux, Mac OS X	Linux, Mac, Windows, BSD

6.1.1.5 Plugins de autenticación de Mosquitto

Un *plugin* (también llamado complemento o módulo) es un *software* que se integra a otro programa más grande para ampliar su funcionalidad. Los *plugins* permiten agregar características o personalizar el comportamiento de una aplicación sin modificar su código fuente principal. Son especialmente útiles porque permiten que los desarrolladores externos mejoren un programa base de manera modular. Un *plugin* generalmente cumple con las siguientes características:

Extensibilidad: Los *plugins* permiten añadir nuevas capacidades a un programa sin alterar su funcionamiento central.

Modularidad: Un *plugin* es independiente, lo que significa que se puede agregar, actualizar o eliminar sin afectar al *software* principal.

Integración fácil: Los *plugins* se integran con el *software* principal a través de una interfaz específica (API) que proporciona puntos de entrada para interactuar con el programa base.

En el caso particular de Mosquitto, se pueden ampliar los mecanismos de autenticación mediante *plugins* para reforzar la seguridad, los que permiten al *broker* autenticar a los clientes que intentan conectarse, verificando sus credenciales (nombre de usuario y contraseña) o certificados en configuraciones más avanzadas.

Mosquitto cuenta con un mecanismo básico de autenticación mediante la utilidad *mosquitto_passwd*, donde las credenciales de los usuarios se almacenan en un archivo de contraseñas, y para protegerlas utiliza una función *hash*, pero no ofrece métodos de autenticación más avanzados de manera predeterminada; en caso de necesitarlos, se debe recurrir a los *plugins* de autenticación de Mosquitto.

Estos *plugins* permiten que Mosquitto admita varios sistemas de autenticación y brinde flexibilidad para diferentes casos de uso. Es posible escribir *plugins* personalizados para conectar Mosquitto con diferentes proveedores de autenticación, como bases de datos, directorios LDAP (*Lightweight Directory Access Protocol*), OAuth o incluso APIs personalizadas.

Los *plugins* de Mosquitto suelen estar escritos en lenguaje C, aunque algunas implementaciones personalizadas pueden usar otros lenguajes de programación mediante técnicas de *bridging*. Los *plugins* se cargan a través del archivo de configuración de Mosquitto (mosquitto.conf), donde se especifica la ubicación del *plugin* y otras opciones necesarias (por ejemplo, detalles de conexión a la base de datos). Por ejemplo:

auth_plugin /ruta/al/plugin.so

auth_opt_<opciones-específicas-del-plugin> <valor>

Actualmente, Mosquitto cuenta con algunos *plugins* de autenticación bien documentados, con un soporte activo de la comunidad. Pero ninguno aprovecha el marco *Enhanced Authentication* de MQTT v5.0 sino que sólo extienden la autenticación básica del protocolo. Uno de los objetivos de este trabajo es, de ser posible, presentar el *plugin* desarrollado a la comunidad de Mosquitto para su consideración y uso. [63]

6.1.1.6 esp-eMQTT5

eMQTT5 es un cliente MQTT escrito en C++ construido para maximizar el rendimiento al mismo tiempo que mantiene el tamaño de código tan pequeño como sea posible de modo que pueda usarse en sistemas embebidos con poca capacidad de memoria. Creado por Cyril Russo y disponible bajo la licencia MIT. esp-eMQTT5 es a su vez, un cliente MQTT optimizado para su uso en ESP32 basado en eMQTT5 por el mismo Cyril Russo.

Este es, hasta donde nuestra investigación determinó, el único cliente MQTT optimizado para su uso en ESP32 que implementa MQTT v5.0. Aunque se revisaron otras opciones, eMQTT5 siempre pareció la más adecuada.

La tabla 6.3 muestra la comparación de diversos clientes MQTT aptos para su uso en microcontroladores:

Tabla 6.3
Clientes MQTT para microcontrolador

Cliente	Versión MQTT	Licencia	Tamaño de código compilado (con dependencias)	Multiplataforma
256dpi esp-mqtt	3.1	MIT	11kB (113kB)	No (ESP32)
Espressif esp-mqtt	3.1	Apache 2.0	12kB (115kB)	No (ESP32)
wolfMQTT	5.0	GPL 2.0	No se probó debido a la licencia	Sí (Win32, Arduino)
Mosquitto	5.0	EPL	Grande	Sí

Tabla 6.3
Clientes MQTT para microcontrolador

Cliente	Versión MQTT	Licencia	Tamaño de código compilado (con dependencias)	Multiplataforma
eMQTT5	5.0	MIT	<17kB (sin dependencias)	Sí (Win32, Arduino, ESP32)

6.1.2. Diseño experimental

En esta sección presentamos el diseño de las extensiones del cliente y el *broker* desarrolladas en este trabajo. También describimos cómo se modificó la biblioteca SimpleHOTP de Arduino para convertirla en un generador de contraseñas TOTP. La arquitectura de cada componente se muestra en la figura 6.3 y se describe con detalle en las siguientes secciones.

6.1.2.1 El generador TOTP

Nuestro generador de TOTP es una versión de la biblioteca de Arduino SimpleHOTP cuyo código fuente está disponible públicamente en GitHub [59] la cual modificamos conforme a los siguientes puntos:

- Cambio de la función *hash* subyacente del tipo Merkle-Damgard SHA1 por la función *hash* de tipo esponja u-QUARK. La implementación de u-QUARK que utilizamos es la que proporcionan los autores en su página web [60].

- Uso del tiempo del sistema en lugar de un contador para generar el HMAC como indica el protocolo TOTP.

- Modificación del algoritmo de truncado dinámico de HOTP para adaptarlo a la menor longitud del HMAC resultante de 17 bytes contra los 20 bytes de SHA1.

El código fuente de nuestro generador se encuentra disponible en GitHub [71].

6.1.2.2 El cliente MQTT

Nuestro cliente es una instancia de la biblioteca esp-eMQTT5 cuyo código fuente se encuentra disponible públicamente en GitHub [61]. Nuestra extensión hace uso de las capacidades de la biblioteca para integrar las siguientes funcionalidades:

- API simple que permite instanciar un cliente MQTT fácilmente en un dispositivo ESP32.
- Integración de nuestro generador TOTP en la misma instancia del cliente, sin requerir una tercera entidad.
- Autenticación automática con el *broker* a través del método TOTP usando el marco *Enhanced Authentication* de MQTT.
- Reautenticación para renovar una sesión MQTT sin terminarla primero.

El código fuente de nuestro cliente está disponible públicamente en GitHub [72].

6.1.2.3 Contribuciones a esp-eMQTT5

Durante el desarrollo de nuestro trabajo, identificamos tres *bugs* en el cliente esp-eMQTT5 por lo que nos pusimos en contacto con su creador, Cyril Russo para solucionarlos. El primero era un problema de compatibilidad de versiones que impedía la compilación del cliente usando la versión 13 o superior de GCC. Le informamos al autor y corrigió el error un día después.

El segundo era un error donde el cliente, después de enviar un mensaje AUTH y recibir como respuesta un mensaje CONNACK, cerraba la comunicación devolviendo como razón un error de protocolo. Se levantó un reporte y se propuso una solución que fue aceptada por el autor.

Por último, se encontró un *bug* que cortaba la comunicación abruptamente durante el intercambio de mensajes AUTH. El autor nos solicitó entonces acceso a nuestro *broker* para poder realizar el *debugging* correspondiente, se le concedió el acceso y solucionó el problema un par de días después.

El intercambio de mensajes público con el autor puede verse en [56] y [57].

6.1.2.4 El *broker* MQTT

Nuestro *broker* es una instancia del *broker* Eclipse Mosquitto v2.0.18. El código fuente está disponible públicamente en GitHub [62] el cual extendemos mediante un *plugin* escrito en lenguaje C. Nuestro *plugin* extiende el *broker* básico con las siguientes capacidades:

- Autenticación para clientes MQTT v5.0 usando TOTP a través del marco *Enhanced Authentication*.

- Verificación de los TOTP recibidos usando nuestro generador TOTP sin el uso de una tercera entidad.

- *Plugin* de fácil integración a cualquier *broker* Mosquitto v2.0.x simplemente añadiendo una línea al archivo de configuración.

El código fuente del *plugin* está disponible en GitHub [72].

6.2. Resultados

En esta sección, evaluamos nuestra propuesta y comparamos el rendimiento de diferentes métodos de autenticación contra sistemas que no utilizan ningún tipo de seguridad. La métrica elegida para esta primera evaluación es la latencia. El análisis de otras métricas relevantes, como el consumo de energía y el uso de recursos computacionales (CPU, memoria), se deja como trabajo futuro. Posteriormente, examinamos la resistencia de nuestra propuesta frente a diversos tipos de ataques informáticos. Finalmente, verificamos nuestra implementación en relación con el cumplimiento de la especificación del estándar, para asegurarnos de que nuestra solución sea interoperable con diferentes implementaciones de *brokers* o clientes MQTT.

6.2.1. Latencia

La figura 6.3 muestra la configuración de red usada en los experimentos para evaluar el prototipo. El cliente y el *broker* se encuentran en redes diferentes unidas a través del Internet global. El cliente se ejecuta en un microcontrolador ESP32 S2 Mini. El *broker* Mosquitto se ejecuta en una computadora DELL Studio XPS 9100 con Ubuntu 16.04 LTS.

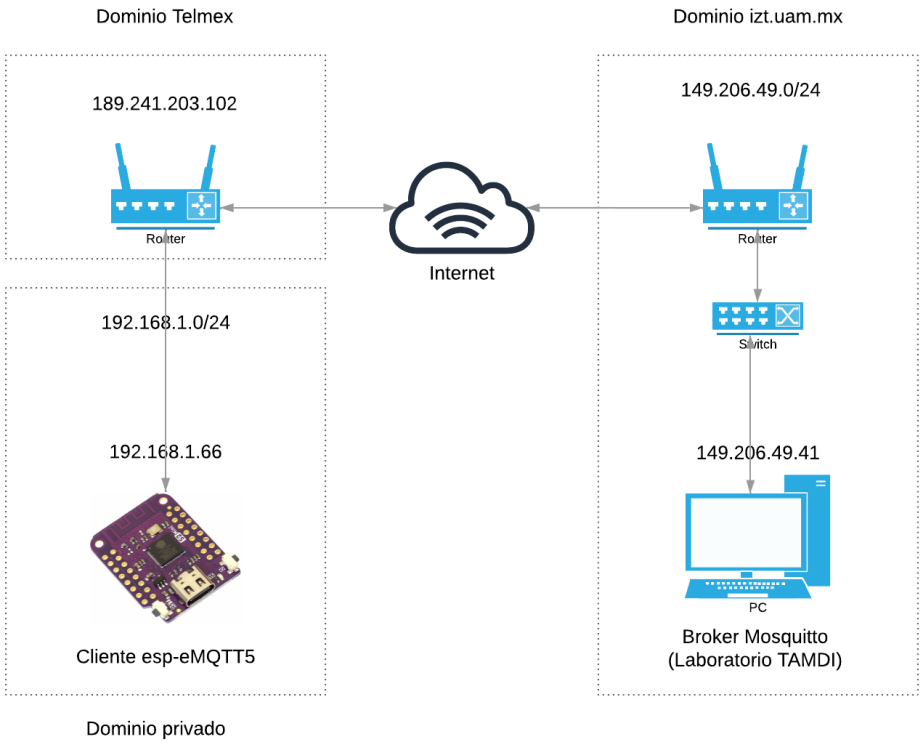

Figura 6.3. Configuración de red de los experimentos

La tabla 6.4, lista los esquemas de autenticación que se utilizaron para evaluar el rendimiento de nuestra propuesta. Todos los esquemas se implementaron en el cliente esp-eMQTT5 instalado en el microcontrolador ESP32.

Tabla 6.4
Métodos de autenticación cliente-broker

Metodo de autenticación	Descripción
Sin autenticación	Ninguna seguridad
Usuario-contraseña simple	Método por defecto usado por MQTT
HOTP (SHA1)	Simple HOTP usando SHA1 como función hash subyacente
TOTP (uQUARK)	TOTP usando uQUARK como función hash subyacente (nuestra propuesta)

En la tabla 6.4, el método de autenticación listado como “Usuario-contraseña simple” es el esquema más simple de autenticación que usa la identidad de cliente y una contraseña para verificar la identidad del cliente. El método HOTP es una implementación de nuestra arquitectura, pero utilizando la biblioteca SimpleHOTP de Arduino sin ninguna modificación, es decir, usando SHA1 como función *hash* subyacente y un contador en vez del tiempo del sistema. El método TOTP se trata de la implementación de nuestra propuesta en el contexto de MQTT v5.0 tal y como se describe en la sección 5.4. La latencia se mide desde que se envía el mensaje CONNECT desde el cliente hasta que este recibe un mensaje CONNACK con el código de respuesta “0” (SUCCESS).

La tabla 6.5 muestra las latencias de los experimentos, donde, podemos ver que la latencia de los 2 métodos más simples es mucho menor que nuestra propuesta, lo cual es de esperar ya que ambos métodos sólo requieren el intercambio de dos mensajes a diferencia de los métodos que utilizan el marco *Enhanced Authentication* de MQTT v5, que requieren el intercambio de cuatro mensajes. El esquema sin autenticación toma en promedio 25.070 ms, lo que es básicamente el tiempo de ida y vuelta entre nuestro cliente y el *broker* ubicado en el laboratorio TAMDI.

Tabla 6.5
Latencias promedio (en ms) para varios esquemas de autenticación de MQTT v5.0

Método de autenticación	Latencia de autenticación promedio (ms)
Sin autenticación	25.070
Usuario-contraseña simple	29.170
HOTP (SHA1)	85.178
TOTP (uQUARK)	400.595

El método usuario-contraseña simple toma 29.170 ms, el método HOTP demanda 85.178 ms y el esquema TOTP requiere 400.595 ms. Descontando el tiempo de ida y vuelta promedio de los 2 intercambios de mensajes, el esquema HOTP (SHA1) requiere de alrededor de 35 ms adicionales y el esquema TOTP (u-QUARK) demanda 350 ms adicionales. Comparados con el esquema usuario-contraseña simple, los esquemas propuestos requieren sólo algunos milisegundos más y logran una seguridad mucho más robusta. Sin embargo, no deja de llamar la atención que nuestra

propuesta de reemplazar SHA1 con u-QUARK en el algoritmo HMAC resulta en una latencia casi 10 veces mayor.

Una revisión más profunda de la literatura nos reveló que nuestra selección de la función *hash* de tipo esponja u-QUARK puede que no haya sido la más adecuada, pues, aunque los resultados de rendimiento comparada con otras funciones esponja son satisfactorios, estos se basan en implementaciones en *hardware* de la función y no en *software*. Nuestra revisión nos lleva a concluir que el uso de otra función *hash* de tipo esponja que tenga implementaciones con mejor rendimiento en *software* como Keccak [64] hubiera sido una mejor opción.

Sin embargo, nuestro trabajo sienta el precedente de que es posible reemplazar la función *hash* subyacente del algoritmo HMAC por una función *hash* de tipo esponja y tener un buen rendimiento en una implementación en un dispositivo de gama baja.

6.2.2. Resistencia a ataques

Nuestra propuesta debe cumplir con los requerimientos de resistencia a los variados tipos de amenazas posibles. Como se describe en la sección 3.4, hay numerosos ataques informáticos a los que MQTT puede estar expuesto y esos mismos se consideran para nuestra evaluación. Nuestra solución es resistente a los siguientes ataques:

6.2.2.1 Suplantación de identidad

Las dos entidades que se comunican se autentican mutuamente utilizando TOTPs. Estos se basan en pares de información secreta (no conocida por el intruso) y en un número único aleatorio o pseudoaleatorio válido solo para un único uso. Por lo tanto, un nodo sin conocimiento de la información personalizada no puede ser autenticado ni suplantar la identidad de un usuario legítimo.

6.2.2.2 Ataque de repetición

El hecho de que el TOTP sea válido para un solo uso protege el sistema contra usuarios malintencionados que intentan reenviar (repetir) los mismos mensajes para hacer un uso no autorizado del sistema. La generación del TOTP incluye el tiempo y, dado que dos paquetes CONNECT no pueden tener el mismo número, no es posible un ataque de repetición.

6.2.2.3 Ataque de Hombre en el Medio (MitM)

Nuestro protocolo de autenticación no puede proteger el sistema cuando un intruso interrumpe el tráfico durante una comunicación, sin embargo, si un intruso obtiene todos los mensajes intercambiados, no puede explotarlos para obtener información secreta ni falsificar un mensaje, ya que garantizamos la integridad del TOTP mediante su *hash*. El algoritmo de generación HMAC-uQUARK es robusto porque utilizamos el tiempo, que es dinámico, además del *nonce* que añadimos en la generación del TOTP.

6.2.2.4 Ataque de fuerza bruta

El TOTP es válido solo para un nodo y dura únicamente una sesión. Recuperar claves o contraseñas de un solo uso para una sesión de comunicación de unos pocos segundos mediante un ataque de fuerza bruta es casi imposible. Según [65], encontrar una contraseña de 9 caracteres (72 bits) usando *hardware* de alto rendimiento puede tardar 24 minutos. Nuestro OTP tiene 9 caracteres y sólo tiene una duración de 30 segundos. A pesar de la evolución de las tecnologías en cuanto a la potencia de cálculo, el carácter dinámico de las contraseñas intercambiadas es una ventaja.

6.2.3. Cumplimiento del estándar e interoperabilidad

Por último, nos aseguramos de que nuestra propuesta siga la especificación del estándar MQTT v5, para garantizar la interoperabilidad con otras implementaciones de TOTP que pudieran surgir en el futuro. Aún no existe otra implementación pública completa que pueda utilizarse como punto de referencia, por lo que probamos el cumplimiento capturando una traza de los mensajes intercambiados entre el cliente y el *broker* y los comparamos con la especificación. Para capturar los mensajes, usamos el *software Wireshark.*

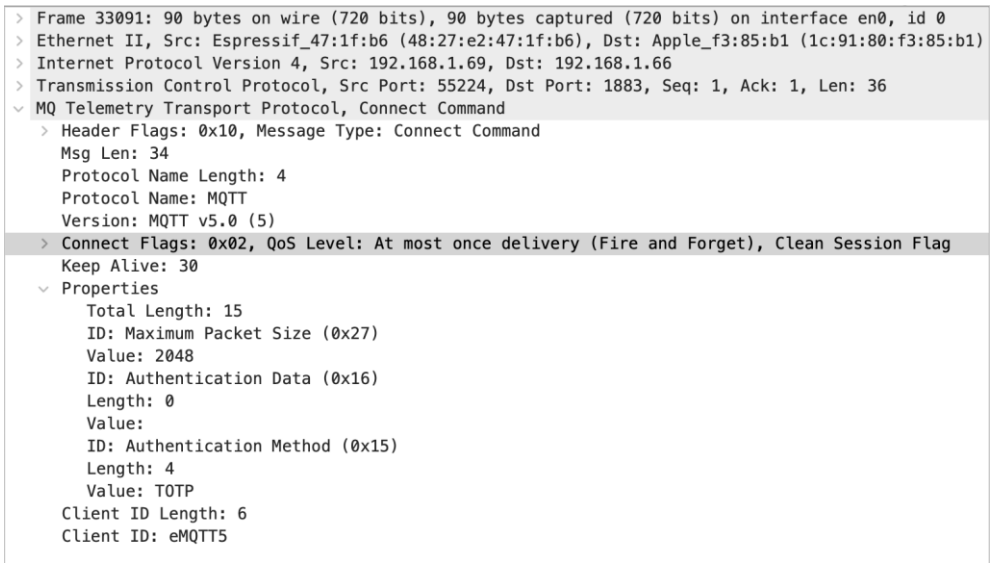

Figura 6.4. Paquete CONNECT

Primero, capturamos un mensaje CONNECT enviado por nuestro cliente esp-eMQTT5 al *broker* como se muestra en la figura 6.4. Nótese que la bandera *Clean Session* está establecida en 0 lo que indica que es una conexión nueva, el *Authentication Method* es “TOTP” y el *Authentication Data* está vacío.

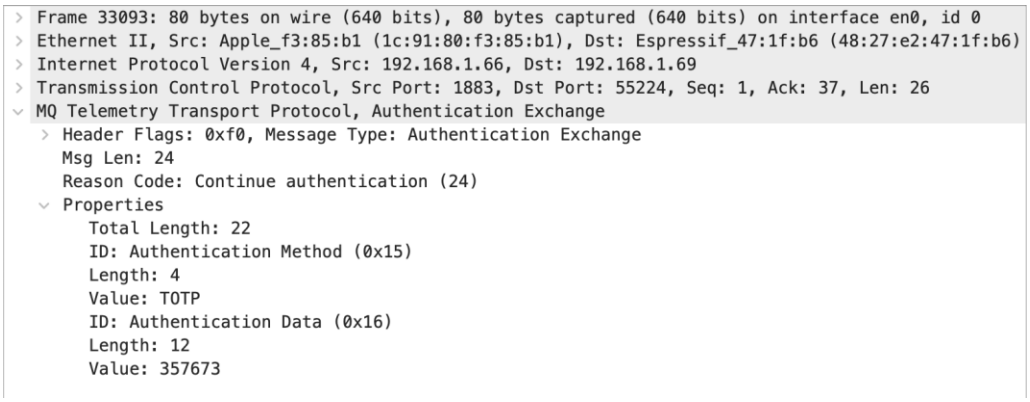

Figura 6.5. Paquete AUTH del broker

El *broker* responde con un paquete AUTH con el *Authentication Method* establecido en “TOTP”, el *Reason Code* establecido en 24 (*Continue Authentication*), y el *Authentication Data*

conteniendo el *nonce* de 6 caracteres del *broker*, precedido por su longitud (figura 6.5). Luego, el cliente responde con un paquete AUTH similar que contiene el TOTP de 9 dígitos en el campo *Authentication Data* (figura 6.6).

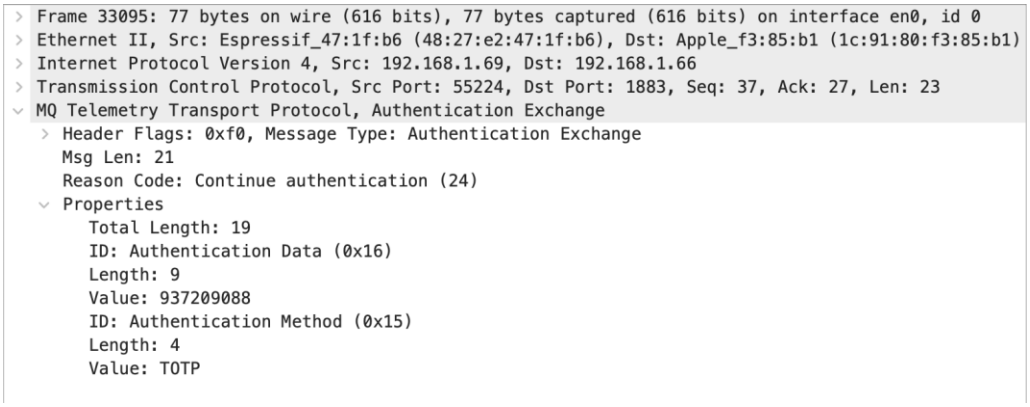

Figura 6.6. Paquete AUTH del cliente

Finalmente, el *broker* responde con un mensaje CONNACK con el *Reason Code* establecido en 0 (*Success*) y el *Authentication Method* establecido en “TOTP” (figura 6.7).

A partir de este intercambio de mensajes, se muestra que nuestra solución cumple completamente con el estándar MQTT v5.

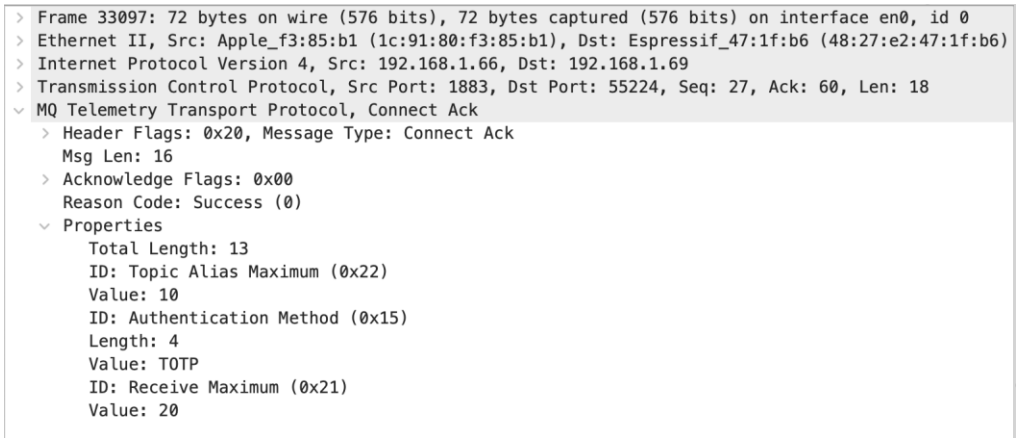

Figura 6.7. Paquete CONNACK

Capítulo 7

7. Conclusiones y trabajo futuro

7.1. Conclusiones

El objetivo de este trabajo fue diseñar una arquitectura de autenticación basada en TOTP e incorporarla al protocolo MQTT, lo que le añade una capa extra de autenticación y autorización. Para alcanzar la meta, aprovechamos la infraestructura *Enhanced Authetication* que nos ofrece la versión 5.0 de MQTT. También modificamos el algoritmo HMAC de TOTP para utilizar una función *hash* subyacente de tipo esponja para así reducir la carga computacional causada por funciones *hash* más costosas sin comprometer la seguridad. Construimos nuestro cliente aprovechando la biblioteca esp-eMQTT5 creada por Cyrill Russo lo que nos permitió usar los microcontroladores ESP32. Además, creamos un *plugin* que permite que el *broker* de código abierto Mosquitto soporte nuestra implementación de TOTP, *plugin* que es muy sencillo de instalar en cualquier instancia de Mosquitto, lo que hace que la arquitectura que hicimos sea fácil de utilizar y simple de migrar a partir de instalaciones inseguras existentes.

Finalmente, evaluamos el rendimiento de nuestra propuesta en términos de latencia, resistencia a ataques y cumplimiento con el protocolo. Los resultados muestran que instalar la implementación que aquí expusimos en dispositivos de gama baja es viable y el mayor consumo de recursos ocurre en la generación del TOTP. Y, a pesar de la inadecuada selección de la función esponja que utilizamos, dado que MQTT está basado en conexiones TCP persistentes, el uso adicional de recursos se compensa fácilmente a través de las sesiones debido a que no es necesario repetir el proceso de autenticación cada vez que hay un intercambio de información.

7.2. Trabajo futuro

Hay varios aspectos que vale la pena explorar en futuros proyectos para mejorar el rendimiento de la implementación, aumentar el nivel de seguridad, optimizar el tamaño del código y proporcionar una evaluación más detallada. Enseguida listamos algunos de ellos:

- Buscar una función esponja más apropiada que u-QUARK para implementarse en *hardware* para reemplazar a SHA1 en el algoritmo HMAC.

- Optimizar y depurar el plugin con el objetivo de presentarlo a la comunidad de Mosquitto para su consideración y uso.

- Realizar pruebas de rendimiento con respecto a otras métricas de interés como consumo de energía, uso de CPU, memoria y uso de red.

Referencias

[1] A. Khanna y S. Kaur, “Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture”, Computers and Electronics in Agriculture, vol. 157, pp. 218–231, feb. 2019, doi: 10.1016/j.compag.2018.12.039.

[2] S. Kumar, P. Tiwari, and M. Zymbler, “Internet of Things is a revolutionary approach for future technology enhancement: a review,” J Big Data, vol. 6, no. 1, Dec. 2019, doi: 10.1186/s40537-019-0268-2.

[3] Z. Yan, P. Zhang, and A. v. Vasilakos, “A survey on trust management for Internet of Things,” Journal of Network and Computer Applications, vol. 42, pp. 120–134, 2014, doi: 10.1016/j.jnca.2014.01.014.

[4] S. Patel, C. Salazar, K. K. Patel, S. M. Patel, and P. G. Scholar, “Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges,” International Journal of Engineering Science and Computing, 2016, doi: 10.4010/2016.1482.

[5] A. el Hakim, “Internet of Things (IoT) System Architecture and Technologies, White Paper. Internet of Things (IoT) System Architecture and Technologies”, doi: 10.13140/RG.2.2.17046.19521.

[6] M. Cabric, “Confidentiality, Integrity, and Availability,” in Corporate Security Management, Elsevier, 2015, pp. 185–200. doi: 10.1016/b978-0-12-802934-3.00011-1.

[7] K. Y. Chai and M. F. Zolkipli, “Review on Confidentiality, Integrity and Availability in Information Security,” Journal of ICT In Education, vol. 8, no. 2, pp. 34–42, Jul. 2021, doi: 10.37134/jictie.vol8.2.4.2021.

[8] M. binti Mohamad Noor y W. H. Hassan, “Current research on Internet of Things (IoT) security: A survey”, Computer Networks, vol. 148, pp. 283–294, ene. 2019, doi: 10.1016/j.comnet.2018.11.025.

[9] Sergey Babkin and Anna Epishkina. Authentication protocols based on one-time passwords. 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2019.

[10] Steven Feltner. (Dec 2016) Single-factor authentication (sfa) vs. multi-factor authentication (mfa), [en línea], disponible: https://delinea.com/blog/sfa-mfa-difference

[11] Jae-Jung Kim and Seng-Phil Hong. A method of risk assessment for multi-factor authentication. Journal of Information Processing Systems, 7(1):187–198, 2011

[12] AIMEE O'DRISCOLL. (marzo 2023) 25+ password statistics that may change your password habits, [en línea], disponible: https://www.comparitech.com/blog/information-security/password-statistics/

[13] Lindell, Andrew Y. "Time versus Event Based One-Time Passwords." (2007).

[14] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. Totp:Time-based one-time password algorithm. Internet Request for Comments, page 685E, 2011.

[15] Futurex, "How Will Lightweight Cryptography Impact You?" Consultado: el 24 de octubre de 2023. [En línea]. Disponible en: https://www.futurex.com/blog/how-will-lightweight-cryptography-impact-you/

[16] "mqtt-v3.1.1-errata01-os-complete Specification URIs," 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.doc

[17] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, "Performance evaluation of CoAP and MQTT with security support for IoT environments," Computer Networks, vol. 197, Oct. 2021, doi: 10.1016/j.comnet.2021.108338.

[18] "mqtt-v3.1.1-os Specification URIs," 2014. [Online]. Available: http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.doc

[19] E. Atilgan, I. Ozcelik, and E. N. Yolacan, "MQTT Security at a Glance," in 14th International Conference on Information Security and Cryptology, ISCTURKEY 2021 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 138–142. doi: 10.1109/ISCTURKEY53027.2021.9654337.

[20] M. M. Raikar and S. M. Meena, "SSH brute force attack mitigation in Internet of Things (IoT) network : An edge device security measure," in ICSCCC 2021 - International Conference on Secure Cyber Computing and Communications, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 72–77. doi: 10.1109/ICSCCC51823.2021.9478131.

[21] V. Tewolde, “Comparison of authentication options for MQTT communication in an IoT based smart grid solution,” p. 46.

[22] P. Verma, “MQTT Security Challenges and Solutions”, International Journal of Enhanced Research in Science, Technology & Engineering vol. 11, abr. 2022.

[23] I. Sahmi, A. Abdellaoui, T. Mazri, y N. Hmina, “MQTT-PRESENT: Approach to secure internet of things applications using MQTT protocol”, IJECE, vol. 11, núm. 5, p. 4577, oct. 2021, doi: 10.11591/ijece.v11i5.pp4577-4586.

[24] B. Hammi, A. Fayad, R. Khatoun, S. Zeadally, and Y. Begriche, “A Lightweight ECC-Based Authentication Scheme for Internet of Things (IoT),” IEEE Systems Journal, Jan. 2020.

[25] “Research on Diffie-Hellman key exchange protocol.”

[26] U. Blumenthal and P. Goel, “Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS),” Request for Comments RFC 4785, Internet Engineering Task Force, Jan. 2007.

[27] S. Amnalou and K. Azmi, “Lightweight Security Mechanism over MQTT Protocol for IoT Devices,” International Journal of Advanced Computer Science and Applications, vol. 11, no. 7, 2020.

[28] Z. Yusoff, M. K. Ishak, L. Rahim, and O. Ali, “Elliptic curve cryptography based Security on MQTT system for Smart Home application,” pp. 1–4, May 2022.

[29] M. T. Hammi, E. Livolant, P. Bellot, A. Serhrouchni, and P. Minet, “A lightweight IoT security protocol,” in 2017 1st Cyber Security in Networking Conference (CSNet), pp. 1–8, Oct. 2017.

[30] M. T. Hammi, E. Livolant, P. Bellot, A. Serhrouchni, and P. Minet, “A Lightweight Mutual Authentication Protocol for the IoT,” in Mobile and Wireless Technologies 2017 (K. J. Kim and N. Joukov, eds.), Lecture Notes in Electrical Engineering, (Singapore), pp. 3–12, Springer, 2018.

[31] J.-H. Han and J. Kim, “A lightweight authentication mechanism between IoT devices,” in 2017 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1153–1155, Oct. 2017.

[32] Ibn Tofail University/Engineering sciences laboratory, Kenitra, Morocco, H. EL Makhtoum, and Y. Bentaleb, “An improved IOT Authentication Process based on Distributed OTP and Blake2,” IJWMT, vol. 11, pp. 1–8, Oct. 2021.

[33] Adhitya Bhawiyuga et al. 2017. Architectural design of token based authentication of "MQTT" protocol in constrained "IoT" device. In TSSA.

[34] Marco Calabretta et al. 2018. A Token-based Protocol for Securing "MQTT" Communications. In SoftCOM.

[35] Matteo Collina et al. 2012. "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST". In IEEE PIMRC.

[36] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, y A. Panya, “Authorization mechanism for MQTT-based Internet of Things”, en 2016 IEEE International Conference on Communications Workshops (ICC), may 2016, pp. 290–295. doi: 10.1109/ICCW.2016.7503802.

[37] Antonio La Marra et al. 2017. "Improving MQTT by Inclusion of Usage Control". In Security, Privacy, and Anonymity in Computation, Communication, and Storage.

[38] Meena Singh et al. 2015. Secure mqtt for internet of things "(IoT)". In Intl Conf. Comm. Sys & Netw. Tech.

[39] Ronald Chiwariro and S Rajendran. 2018. Security in Publish/Subscribe Protocol for Internet of Things. Pure and Applied Mathematics 118, 16 (2018), 231–242.

[40] Suja P Mathews and Raju R Gondkar. 2019. Protocol Recommendation for Message Encryption in "MQTT". In IconDSC.

[41] “Top IoT Trends in 2024 and What IoT Holds for the Future?” Consultado: el 29 de julio de 2024. [En línea]. Disponible en: https://www.antino.com/blog/top-9-iot-trends

[42] G. M. Bertoni, J. Daemen, M. Peeters, y G. Assche, “Sponge Functions”, ECRYPT hash Workshop 2007, ene. 2007.

[43] J.-P. Aumasson, L. Henzen, W. Meier, y M. Naya-Plasencia, “Quark: A Lightweight hash”, J Cryptol, vol. 26, núm. 2, pp. 313–339, abr. 2013, doi: 10.1007/s00145-012-9125-6.

[44] D. M'Raihi, J. Rydell, M. Pei, y S. Machani, "TOTP: Time-Based One-Time Password Algorithm", Internet Engineering Task Force, Request for Comments RFC 6238, may 2011. doi: 10.17487/RFC6238.

[45] Y. Naito y L. Wang, "Replacing SHA-2 with SHA-3 Enhances Generic Security of HMAC", en Topics in Cryptology - CT-RSA 2016, K. Sako, Ed., Cham: Springer International Publishing, 2016, pp. 397–412.

[46] H.-Y. Chien, "Highly Efficient Anonymous IoT Authentication using Composite hashing", en 2021 IEEE Conference on Dependable and Secure Computing (DSC), ene. 2021, pp. 1–7. doi: 10.1109/DSC49826.2021.9346263.

[47] H. Y. Chien, Two-Level-Composite-hashing Facilitating Highly Efficient Anonymous IoT and D2D Authentication, MDPI Electronics, Vol. 10, No. 7, Article No. 789, April, 2021.

[48] "MQTT 5.0 Enhanced Authentication | EMQX Docs". Consultado: el 8 de agosto de 2024. [En línea]. Disponible en: https://docs.emqx.com/en/emqx/latest/access-control/authn/scram.html

[49] R. Sandhu y J. Park, "Usage Control: A Vision for Next Generation Access Control", en Computer Network Security, vol. 2776, V. Gorodetsky, L. Popyack, y V. Skormin, Eds., en Lecture Notes in Computer Science, vol. 2776. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 17–31. doi: 10.1007/978-3-540-45215-7_2.

[50] S. Shin and K. Kobara, "Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2," Request for Comments RFC 6628, Internet Engineering Task Force, June 2012. Num Pages: 20.

[51] E. Toé, D. F. Somé, y T. Yélémou, "Lightweight and robust MQTT protocol authentication model suitable for connected portals", en 2023 IEEE Multi-conference on Natural and Engineering Sciences for Sahel's Sustainable Development (MNE3SD), feb. 2023, pp. 1–7. doi: 10.1109/MNE3SD57078.2023.10079830.

[52] "ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems". Consultado: el 23 de agosto de 2024. [En línea]. Disponible en: https://www.espressif.com/en/products/socs/esp32

[53] "Visual Studio Code - Code Editing. Redefined". Consultado: el 24 de agosto de 2024. [En línea]. Disponible en: https://code.visualstudio.com/

[54] “ESP-IDF Getting Started | Espressif Systems”. Consultado: el 24 de agosto de 2024. [En línea]. Disponible en: https://idf.espressif.com/

[55] S. Sharma, S. Jain, y B. Chandavarkar, “Nonce: Life Cycle, Issues and Challenges in Cryptography”, 2021, pp. 183–195. doi: 10.1007/978-981-15-7961-5_18.

[56] Closing the AUTH exchange. Issue #2. (el 11 de febrero de 2024). C++. Consultado: el 31 de agosto de 2024. [En línea]. Disponible en: https://github.com/X-Ryl669/esp-eMQTT5/issues/2

[57] “Documentation for eMQTT5 client | X-Ryl669 personal blog”. Consultado: el 31 de agosto de 2024. [En línea]. Disponible en: https://blog.cyril.by/en/documentation/emqtt5-doc/emqtt5

[58] B. T. Hammad, N. Jamil, M. E. Rusli, y M. R. Zaba, “A survey of Lightweight Cryptographic hash Function”, International Journal of Scientific & Engineering Research, vol. 8, núm. 7, 2017.

[59] J. Lusky, jlusPrivat/SimpleHOTP. (el 28 de marzo de 2024). C++. Consultado: el 14 de septiembre de 2024. [En línea]. Disponible en: https://github.com/jlusPrivat/SimpleHOTP

[60] J.-P. Aumasson, veorq/Quark. (el 23 de julio de 2022). C. Consultado: el 31 de agosto de 2024. [En línea]. Disponible en: https://github.com/veorq/Quark

[61] X-Ryl669, X-Ryl669/esp-eMQTT5. (el 11 de febrero de 2024). C++. Consultado: el 31 de agosto de 2024. [En línea]. Disponible en: https://github.com/X-Ryl669/esp-eMQTT5

[62] eclipse/mosquitto. (el 14 de septiembre de 2024). C. Eclipse Foundation. Consultado: el 14 de septiembre de 2024. [En línea]. Disponible en: https://github.com/eclipse/mosquitto

[63] “Authentication plugins”, Eclipse Mosquitto. Consultado: el 17 de septiembre de 2024. [En línea]. Disponible en: https://mosquitto.org/blog/2013/07/authentication-plugins/

[64] A. Dolmeta, M. Martina, y G. Masera, “Comparative Study of Keccak SHA-3 Implementations”, Cryptography, vol. 7, núm. 4, Art. núm. 4, dic. 2023, doi: 10.3390/cryptography7040060.

[65] “Password Tester | Test Your Password Strength”, Bitwarden. Consultado: el 26 de agosto de 2024. [En línea]. Disponible en: https://bitwarden.com/password-strength/

[66] “Products”, NORVI Industrial Arduino. Consultado: el 23 de octubre de 2024. [En línea]. Disponible en: https://norvi.lk/products-2/

[67] “Smart Home Automation Products”, SONOFF Official. Consultado: el 23 de octubre de 2024. [En línea]. Disponible en: https://sonoff.tech/products/

[68] H. Krawczyk, M. Bellare, y R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, Internet Engineering Task Force, Request for Comments RFC 2104, feb. 1997. doi: 10.17487/RFC2104.

[69] J.-S. Coron, Y. Dodis, C. Malinaud, y P. Puniya, “Merkle-Damgård revisited: how to construct a hash function”, en Proceedings of the 25th annual international conference on Advances in Cryptology, en CRYPTO’05. Berlin, Heidelberg: Springer-Verlag, ago. 2005, pp. 430–448. doi: 10.1007/11535218_26.

[70] P. Rogaway y T. Shrimpton, “Cryptographic Hash-Function Basics: Definitions, Implications and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”, 2004, 2004/035. Consultado: el 13 de febrero de 2025. [En línea]. Disponible en: https://eprint.iacr.org/2004/035

[71] W. Eddy, “Transmission Control Protocol (TCP)”, Internet Engineering Task Force, Request for Comments RFC 9293, ago. 2022. doi: 10.17487/RFC9293.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00114
Matrícula: 2222800215

INTEGRACIÓN DE UN MECANISMO DE AUTENTICACIÓN TOTP EN UN AMBIENTE MQTT PARA DISPOSITIVOS DE GAMA BAJA.

En la Ciudad de México, se presentaron a las 11:00 horas del día 13 del mes de junio del año 2025 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. RUBEN VAZQUEZ MEDINA
MTRO. OMAR LUCIO CABRERA JIMENEZ
DR. LEONARDO PALACIOS LUENGAS

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)

DE: RUBEN DARIO GARCIA GONZALEZ

RUBEN DARIO GARCIA GONZALEZ
ALUMNO

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

REVISÓ

MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

Román Linares Romero

DR. ROMAN LINARES ROMERO

PRESIDENTE

DR. RUBEN VAZQUEZ MEDINA

VOCAL

MTRO. OMAR LUCIO CABRERA JIMENEZ

SECRETARIO

DR. LEONARDO PALACIOS LUENGAS