TESIS DOCTORAL

2021

MQTT-SCACAUTH: Esquema de seguridad para el protocolo MQTT y su uso en el entorno del IIoT

Eduardo Buetas Sanjuan

Programa de Doctorado en Ingeniería de Sistemas y Control (Código 9612)
Ismael Abad Cardiel
Jose Antonio Cerrada Somolinos

Esta tesis está dedicada a Pi, pieza indispensable de mi vida y apoyo incondicional e irreemplazable para la realización de esta tesis, sin tu apoyo no hubiera sido posible

A mí familia por su apoyo durante la realización de esta tesis y siempre

.....y a Carmen y David que desde aquel día de septiembre de 2019 en Galway han estado convencidos, mucho más que yo, de que llevaría a buen término esta tesis

Agradecimientos

Quiero mostrar mi agradecimiento, en primer lugar, a Pilar, mi compañera en la vida, que durante estos años de tanto trabajo ha tenido una paciencia infinita y siempre me ha animado y apoyado.

A mi familia y amigos, que siempre han confiado en mí y me han dado ánimos para la realización de esta tesis.

A mis directores de tesis, Ismael Abad Cardiel y José Antonio Cerrada Somolinos, su apoyo y colaboración que me han permitido realizar este trabajo de investigación.

Ambos me animaron a inscribirme en este programa de doctorado en el momento de la finalización de mi Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos, sin sus palabras de ánimo y confianza ni siquiera hubiera empezado esta investigación.

Ambos, junto a Carlos Cerrada Somolinos, han sido coautores de las publicaciones presentadas durante el proceso de realización de este doctorado. Estas publicaciones sin su colaboración no hubieran sido posibles.

Contenido

Índice de tablas

Índice de figuras

Acrónimos

AES: Advanced Encryption Standard
AMQP: Advanced Message Queuing Protocol
APDU: Application Protocol Data Unit
API: Aplication Programming Interface
ARM: Architecture developed by Andvancec RISC Machines
ATR: Answer To Reset
AWS: Amazon Web Services
CAN: Controller Area Network
CBC: Cipher Blocker Chaining
CoAP: Constrained Application Protocol
CPS: Cyber-Physical System
DDS: Data Definition Specifications
DES: Data Encryption Standard
DM: Diagnostic Message
DPA: Differential Power Analysis
DTC: Diagnostic Trouble Code
ECC: Elliptic Curve Cryptography
ECU: Electronic Control Unit
EDC: Error Detection Code
EDL: Eclipse Distribution License
EEPROM: Electrically Erasable Programmable Read-Only Memory
EPL: Eclipse Public License
ETSI: European Telecommunications Standards Institute
FMI: Failure Mode Identifier
FUB: Funktionsplan
GCP: Google Cloud Platform
GPS: Global Positioning System
HSM: Hardware Security Module
HTTP: Hypertext Transfer Protocol
IBC: Identity-Based Cryptography
IDE: Integrated Development Environment
IEC: International Electrotechnical Commisiion
IEEE: Institute of Electrical and Electronics engineers
IFSC: Information Field Size for the Card

IIoT: Industrial Internet of Things
IoT: Internet of Things
IP: Internet Protocol
ISO: International Organization for Standarization
IT: Information Technology
JCR: Jornal Citation Reports
JCVM: Java Card Virtual Machine
JSON: JavaScript Object Notation
KOP: Kontaktplan
LAPTAS: Lightweight Anonymous Privacy-preserving Three-factor Authentication Scheme
LDAP: Lightweight Directory Access Protocol
LRC: Longitudinal Redundancy Check
LSB: Least Significant Byte
MQTT: Message Queing Telemetry Transport
MQTT-SN: Message Queing Telemetry Transport for Network Sensors
MSB: Most Significant Byte
MUSCLE: Movement for the Use of Smart Cards in Lunux Environment
NIST : National Institute of Standards and Technology
NL: No Local
OASIS : Organization for the Advancement of Structured Information Standards
OAuth : Open Authorization
OT: Operational Technology
PC: Personal Computer
PCB: Protocol Control Byte
PCSC: Personal Computer Smart Card
PDU : Protocol Data Unit
PGN: Parameter Groups Numbers
PKCS: Public-Key Cryptography Standards
PLC: Programmable Logic Controller
PPS: Protocol and Parameter Selection
PWM: Pulse Width Modulation
QoS: Quality of Service
RAM: Random Access Memory
RAP: Retain as Published
REST: Representational State Transfer
RFU: Reserved for Future Use

RISC: Reduced Instruction Set Computer
RN: Random Number
RSA: Rivest, Shamir y Adleman
RTU: Remote Terminal Unit
SAE: Society of Automotive Engineers
SCACAUTH: Smart Card Asymmetric Cryptography Authentication
SCADA: Supervisory Control and Data Acquisition
SCL: Structured Control Language
SCOS: Smart Card Operating System
SCP: Secure Channel Protocol
SDK: Software Development Kit
SoC: System on Chip
SPA: Simple Power Analysis
SPN: Suspect Parameter Numbers
SRAM: Static Random Access Memory
SSID: Service Set Identifier
SW: Status Word
TCP: Transmission Control Protocol
TEA: Tiny Encryption Algorithm
TLS : Transport Layer Security
TS: Technical Specification
UART: Universal Asynchronous Receiver/Transmitter
UID: Unique identifier
USB: Universal Serial Bus
UTF: Unicode Transformation Format
WAN: Wide Area Network
WSN: Wireless Sensor Networks

MQTT-SCACAUTH: Esquema de seguridad para el protocolo MQTT y su uso en el entorno del IIoT

Eduardo Buetas Sanjuan

Resumen

Con la imparable introducción del internet de las cosas en la industria, cada vez son más los equipos que deben enviar datos desde las líneas de producción a los servidores superiores de toma de decisiones. Estos datos son ya imprescindibles para la toma de decisiones dependiendo de la situación de las líneas de producción, el mercado, el estado de los suministros y los procesos logísticos. En el entorno industrial es imprescindible que las comunicaciones sean seguras, ya que los ataques exitosos a las comunicaciones industriales pueden suponer grandes pérdidas económicas. Por estas razones en esta tesis se desarrolla un sistema de comunicación segura partiendo del protocolo Message Queue Telemetry Transport (MQTT). Se introduce una autenticación mutua segura entre el broker y el cliente basada en criptografía asimétrica y un esquema para asegurar el ocultamiento y la confianza en los datos intercambiados utilizando criptografía simétrica con claves de un solo uso. Todo el desarrollo se realiza siguiendo el último estándar publicado del protocolo MQTT.

Una vez expuesto de manera teórica el esquema de seguridad propuesto, se realizan todos los prototipos necesarios para la experimentación con este esquema de seguridad, creando prototipos cliente que cubren la gran mayoría de los sistemas de control de los sistemas productivos.

En la presente tesis también se realizan las mediciones necesarias para validar la posibilidad de uso de este esquema de seguridad en los entornos industriales reales.

Palabras Clave: IIoT, MQTT, Autenticación Mutua, Smart Card, JavaCard, Protocolo Seguro.

MQTT-SCACAUTH: Security schema for MQTT protocol and it use in IIoT environment

Eduardo Buetas Sanjuan

Abstract

With the unstoppable introduction of the Industrial Internet of Things, more and more equipment must send data from the production lines to superior decision-making servers. These data are already essential for decision-making depending on the situation of the production lines, the market, the state of supplies and logistics processes. In the industrial environment, communications must be secure since successful attacks on industrial communications can lead to great economic losses. For these reasons, this thesis developed a secure communication system based on the Message Queue Telemetry Transport Protocol (MQTT). It is made by introducing a process to secure mutual authentication between the broker and the clients based on asymmetric cryptography and a scheme to secure concealment and trust in data exchanged using symmetric cryptography with one-time keys. All development is done following the latest published standard of the MQTT protocol.

Once the proposed secure schema has been exposed in a theoretical way, the thesis shows all the prototypes necessary for the experimentation with this safety scheme, creating client prototypes that cover the vast majority of the control systems of the production systems used in current industrial environments.

This thesis also makes measurements needed to validate the possibility of using this security scheme with the industrial current environment.

Keywords: IIoT, MQTT, Mutual Autentication, Smart Card, JavaCard, Secure Protocol.

Capítulo 1

Introducción

En este capítulo se presenta la motivación que ha llevado a la realización de esta tesis, los objetivos que se han alcanzado durante la realización de la misma y su estructura.

1.1. Motivación

En los últimos años los procesos de fabricación deben enviar gran cantidad de datos a los servidores superiores de la industria para su recopilación y análisis, con el objetivo de mejorar y flexibilizar los procesos de fabricación [1], en la mayoría de los casos en modelos de producción multi-planta. Estos datos también son ampliamente utilizados para mejorar los procesos de mantenimiento correctivo y preventivo [2], indispensables para mantener la producción. Y evitar en lo posible que los tiempos de no disponibilidad de los elementos productivos provoquen disfunciones en la cadena de suministro con sus consecuentes pérdidas económicas.

No cabe duda de que, en los nuevos paradigmas de la industria, los sistemas de control de la producción (OT) deben estar integrados en los sistemas de información (IT). Y así, poder recopilar los datos fundamentales para el correcto desarrollo de los procesos industriales [3] caracterizados cada vez más por la flexibilidad y la orientación a la producción según la demanda [4]. En este momento las decisiones para la optimización de los procesos productivos se deben tomar teniendo en cuenta la demanda, las incidencias en la cadena de suministro y los cuellos de botella provocados tanto por las especificidades de las características de la producción como por las posibles incidencias en los medios productivos. De este modo las fábricas actuales puedan modificar su producción dependiendo de los entornos productivos generados a partir de las incidencias detectadas, para adecuarse lo mejor posible tanto a las capacidades productivas de cada planta en cada momento, como a la demanda por parte del mercado.

Para poder alcanzar estos objetivos, es imprescindible que la comunicación entre los sistemas productivos, cadena de suministro, departamentos de ventas, gestión de personal, mantenimiento, gestión de almacén y, en definitiva, todos los sistemas relacionados con el negocio, estén suministrando datos de manera constante a los sistemas de toma de decisión.

En la última década, los sistemas de producción han evolucionado con la imparable llegada de la industria 4.0. En este nuevo entorno los sistemas de control de los medios productivos ya han dejado de ser únicamente PLCs y RTUs. En este momento forman parte de ellos todo un ecosistema de dispositivos comunicados entre sí y con el resto de sistemas. Desde sensores conectados para la toma de variables

de entorno hasta los denominados Cyber-Physical Systems (CPS), que interconectan diversos sensores y actuadores que toman datos y actúan sobre el mundo físico. Estos sistemas se conectan a su vez con los sistemas de toma de decisiones bien locales o remotos [5], con el objetivo de controlar los sistemas productivos.

Todos estos elementos, tanto actuales, pertenecientes a la industria 4.0, como los heredados de la industria 3.0, generan una gran cantidad de datos que deben ser enviados a los sistemas de toma de decisión para su procesamiento.

Esto supone un nuevo reto en el ámbito de las comunicaciones industriales entre los sistemas que controlan las líneas de producción (OT) y los sistemas de análisis de estos datos y toma de decisiones a partir de ellos (IT).

La literatura existente trata de resolver cada una de las partes de las comunicaciones entre sistemas por separado: protocolos de envío de datos, seguridad en las comunicaciones, generación de código automático para el envío de datos, gestión de redes, etc... como podremos ver en el estado del arte actual (punto 2 del presente documento).

Una encuesta realizada por Software AG [6] a 125 fabricantes norteamericanos de la industria pesada y la automotriz, demuestra la importancia que los fabricantes dan a estos nuevos retos. Esta encuesta desvelo que el 80 % de los encuestados creen que los procesos alrededor de las plataformas IIoT deben optimizarse o se enfrentan a una desventaja competitiva.

Otro de los retos a los que se enfrenta esta nueva revolución industrial es el de la seguridad en todas sus comunicaciones. La dificultad para establecer mecanismos de seguridad en estos sistemas es doble. Por un lado, los sistemas industriales están interconectados entre sí, así como con sus clientes y proveedores, utilizando para ello redes públicas de comunicación. Y por otro, los nuevos paradigmas de producción modulares propios de la industria 4.0 integran en muchas ocasiones dispositivos con muy bajos recursos de procesamiento [7].

Es en este escenario donde se enmarca esta tesis, que pretende plantear un esquema para el envío de estos datos, producidos a nivel de planta en fase de fabricación, a los servidores empresariales con un enfoque holístico. Entendiendo esta comunicación como un esquema complejo único, y no la suma de sus diferentes partes por separado.

Este esquema pretende también integrar los nuevos elementos de control de los sistemas productivos propios de la industria 4.0 con los sistemas anteriores basados en PLCs y RTUs. Esta integración se considera de gran importancia porque los sistemas de control industrial tienen una larga vida útil. En los próximos años, encontraremos la convivencia de elementos de control propios de la industrial 3.0 y elementos de control propios de la industria 4.0, generando datos desde ambos tipos de controladores y siendo necesario el envío de estos a los sistemas de toma de decisión. Los dispositivos heredados de la industria 3.0, cuando se realizan actualizaciones graduales de las instalaciones en la industria, suponen uno de los principales problemas de seguridad en la implantación de la IIoT [8], [9].

Para la adquisición de todos los datos relativos al producto en su fase de fabricación es necesaria la construcción de un sistema de comunicación de datos durante todo el proceso a los sistemas de toma de decisiones y de estos a los diferentes sistemas industriales, para así poder modificar su comportamiento dependiendo de las decisiones tomadas por estos.

1.2. Alcance y Objetivos

En la presente tesis se ha realizado un sistema de comunicación completo, que permite comunicar los datos necesarios a los sistemas de toma de decisión desde los sistemas productivos y desde estos sistemas de toma de decisión hasta los elementos de control de las líneas de producción.

Este sistema se basa en la utilización de un protocolo liguero, para que sea posible su integración en todos los elementos de la cadena de producción, permitiendo integrar la necesaria seguridad en sus comunicaciones.

Se pueden integrar en este sistema, tanto en elementos propios de la industria 4.0 como en los elementos heredados de la industria 3.0. Para, de este modo, abarcar toda fase de producción, desde sus primeras etapas hasta su finalización, almacenaje y envío para su distribución. Independientemente si los sistemas de control de estos procesos son sistemas englobados en el ecosistema IIoT o sistemas preexistentes pertenecientes a la industria 3.0 (PLCs, RTUs, etc..) .

Para ello se han generado los sistemas de comunicación necesarios para la recogida de los datos de producción de una forma segura. Teniendo así los datos de todo el proceso productivo para su análisis. Con el análisis de estos datos se tomarán decisiones para la mejora del proceso productivo.

En el desarrollo de estos objetivos se ha utilizado como base el protocolo MQTT [s1], ampliamente utilizado ya tanto en sistemas IoT [10] como en sistemas IIoT [11]. Dicho protocolo va a servir para conectar todos los elementos que generan información con los elementos que la consumen.

A este protocolo MQTT se le han añadido, sin modificar su estándar, una serie de elementos, tanto para la identificación univoca y segura de los participantes, así como para el cifrado de los mensajes enviados entre ellos. De modo que este protocolo se considera como un protocolo seguro que pueda acreditar que la procedencia y contenido de los mensajes intercambiados sea identificada de manera segura y no pueda ser leída ni modificada por ningún agente externo al sistema.

Para incluir la necesaria seguridad en las comunicaciones se utiliza un sistema de pares de claves públicas y privadas para las autenticaciones mutuas en la conexión de los dispositivos al sistema. Se realiza tanto la autenticación del dispositivo ante el controlador del sistema como de este ante el dispositivo. Por otro lado, para asegurar la integridad y el no descubrimiento por parte de un atacante de los datos transmitidos, se utiliza un sistema de cifrado de bloques con claves aleatorias y de un único uso, intercambiadas de forma segura entre los participantes de la comunicación.

Con el objetivo de posibilitar la utilización de criptografía, tanto simétrica como asimétrica, por parte de todos los elementos del sistema, y con la visión holística que se pretende alcanzar en esta tesis, se incluye dentro del sistema el hardware necesario para la ejecución de los algoritmos criptográficos, tanto simétricos como asimétricos, necesarios en el proceso.

Se dota, así, a todos los elementos del sistema de un hardware criptográfico, Smart Card criptográficas para los clientes (o cualquier μprocesador criptográfico seguro) y o bien Smart Card/μprocesador criptográfica/o o bien un módulo hardware de seguridad (HSM por sus siglas en inglés) para el controlador del sistema.

Para demostrar la validez de los elementos expuestos en esta tesis se ha realizado un sistema completo y funcional de comunicación. Como elementos en la cadena de

producción se han realizado prototipos funcionales de este método de comunicación en diversos elementos, un PLC, un μcontrolador y un μPC, elementos que cubren una gran gama de controladores de automatización posibles, tanto en la Industria 4.0 como los heredados de la Industria 3.0. Además, se ha realizado un prototipo para la comunicación de datos desde vehículos de transporte logístico, con el objeto de incluir en este sistema de comunicación los datos de los procesos logísticos que afectan a la fase de producción y expedición de los productos manufacturados.

1.3. Estructura de la Memoria de la Tesis

Esta tesis está estructurada para presentar el problema actual de las comunicaciones industriales seguras, partiendo de un estado del arte actual sobre el tema y desarrollando la solución aportada en la presente tesis.

- Capítulo 1: Introducción y motivación de la tesis, desgranando la problemática actual en las comunicaciones industriales seguras durante los procesos de fabricación y expedición de los productos finalizados.
- Capítulo 2: Estado del arte actual, realizando un estudio de las publicaciones actuales referentes al tema.
- Capítulo 3: Mejoras propuestas para el protocolo de comunicación seleccionado y desarrollo en detalle del sistema completo propuesto para dar solución a los problemas actuales de seguridad y compatibilidad del sistema de comunicación presentado.
- Capítulo 4: Detalle del protocolo utilizado para el desarrollo de la presente tesis, partiendo del estándar MQTT y desarrollando las técnicas utilizadas para su seguridad.
- Capítulo 5: Integración del sistema en elementos reales. Desarrollo de todos los software requeridos para el sistema completo, software criptográficos para Smart Card, librerías para una integración rápida por terceros, sistema de inicialización de Smart Card, prototipos de clientes y de broker.

- Capítulo 6: Análisis experimental de los prototipos desarrollados en la presente tesis, analizando los tiempos dedicados a las funciones criptográficas en los diferentes intercambios de mensajes así como el consumo eléctrico producido por los diferentes procesos criptográficos.
- Capítulo 7: Conclusiones del desarrollo de la tesis y futuras investigaciones.

Capítulo 2

Estado del Arte

En este capítulo se realiza un estudio del arte de los temas tratados en esta tesis, realizando el análisis de una selección de artículos relevantes publicados en los últimos años.

En la actualidad existen gran cantidad de publicaciones relacionadas con las comunicaciones tanto en el ámbito de la IoT como en el de la industria 4.0 o IIoT, siendo muchas de ellas estados del arte de este campo en sí mismas. Entre los estudios del arte cabe destacar el de Ercan *et al.* [12] revisando un total de 620 publicaciones relativas a la industria 4.0. publicados entre 2001 y 2018. Clasificando según las palabras de búsqueda utilizadas, como podemos ver en la siguiente tabla (2.1).

Tabla 2.1: Key words list (Ercan *et al.* [12])

Key words	**Number of publication reviewed**
Industry 4.0 in general	132
Cyber physical systems (CPS)	81
Cloud, cloud systems	53
Internet of things	110
M2M, machine to machine	12
Smart factory	45
Data mining, big data	38
ERP and business intelligence	55
Augmented reality, simulation	21
Virtual manufacturing	23
Intelligent robotics	30
Others (projects and national initiatives)	20
Total # of publication	620

Madakam y Takahiro en [13] realizan un profundo análisis de los procesos, aplicaciones en industria, evolución, aplicaciones reales, protocolos y literatura existente del IoT aplicado en la industria.

Figueroa *et al.* [14] realizan un análisis de los diferentes protocolos que pueden intervenir en las comunicaciones del IIoT y de sus vulnerabilidades. Analizando y clasificando 33 protocolos que pueden intervenir en las comunicaciones de planta, desde los protocolos deterministas específicos de los sistemas de automatización: CAN, Profinet, Siemens S7... hasta los protocolos propios de IT: Rest, HTTP, TLS, etc...

pasando por los protocolos del IoT más utilizados en las comunicaciones industriales: MQTT, CoAP, etc..

Este es un buen punto de partida para la comprensión de la gran variedad de protocolos que son utilizados en la actualidad para las comunicaciones en la industria y sus vulnerabilidades, revelando la disparidad de protocolos utilizados en las diferentes etapas de la comunicación. Por un lado, la adquisición de datos en planta y por otro la comunicación de estos con los sistemas destinatarios de estos datos.

Jaloudi, en su artículo [15], realiza una comparativa entre cuatro de los protocolos más utilizados en el ámbito de la IIoT: HTTP, CoAP, MQTT y Modbus TCP y plantea la posibilidad de utilizar un protocolo para la comunicación de un sistema de automatización con su periferia (ModBus TCP) y otro protocolo para comunicar los datos generados por el sistema a los servidores superiores (MQTT). Este planteamiento, a juicio del autor de esta tesis, es el correcto. El protocolo utilizado para la comunicación de los datos generados por los sistemas de automatización a los sistemas superiores de planta no debe ser el mismo que el utilizado por los sistemas de automatización con los elementos de periferia propios de la automatización.

Para la comunicación a nivel de automatización, el protocolo utilizado debe tener una serie de características muy específicas. Especialmente su determinismo que no solo no son necesarios para la comunicación con los servidores superiores de planta, sino que pueden ser incluso contraproducentes para esta comunicación, debido a que su complejidad y especificidad puede hacer imposible su implementación en todos los elementos del sistema.

Li Ma y Zhaoyuan Xiu, en su artículo [16], tratan de dar solución a un problema que aparece en los sistemas complejos de comunicación IIoT, que es la integración de elementos, muy especialmente sensores de variables de entorno, que no tienen capacidad de comunicación IP y que se comunican por diversas redes inalámbricas de comunicación, muchas veces propietarios de fabricantes determinados e incompatibles entre ellas. Para ello desarrolla una Gateway multiprotocolo para pasar los datos obtenidos de esas redes de sensores a redes IP.

Ala-Laurinaho *et al.* [17] proponen en su artículo un sistema abierto de control de sensores en el ámbito del IIoT, "OSEMA". Este sistema permite la gestión remota de sensores y la recopilación de los datos que generan. Un punto interesante de este artículo es la propuesta de la generación de código para los sensores desde el servidor

del sistema en respuesta a cambios de configuración realizados para los sensores.

Barrientos-Avendaño *et al.* , en su artículo [18], realizan el estudio de una aplicación de los conceptos de la industria 4.0 sobre una empresa real "Tu Pan Gourmet SAS". Es interesante ver estudios sobre la aplicación real de los conceptos de la industria 4.0 y las posibilidades de dicha integración.

En el artículo de Cohen *et al.* [19] se realiza un interesante estudio de las tecnologías aplicadas en el denominado "Assembly 4.0", la parte de la industria 4.0 centrada específicamente en las tecnologías aplicadas a los procesos de fabricación, robots colaborativos, realidad aumentada, sensores y actuadores inteligentes y conectados, fabricación aditiva, etc...

Paniagua *et al.* , en su artículo [20], realiza un estudio del estado del arte en relación a los diferentes frameworks existentes en el ámbito de la IIoT.

2.1. Seguridad en el Ámbito del IIoT

Existe un amplio volumen de publicaciones sobre la seguridad en las comunicaciones en el IIoT. Una de las partes más importantes para estudiar este tema son los métodos de análisis de seguridad de los sistemas. Kim *et al.* [21], realizan una exposición en la cual afirman que estos sistemas no pueden ser estudiados aplicando técnicas de White-Box ni Grey-Box, realizando una propuesta para realizar análisis de seguridad sobre sistemas de comunicación en el ámbito del IIoT, utilizando una técnica Black-Box Fuzzing.

Existen diversos artículos en los que se describen implementaciones de sistemas de testeo de la seguridad (Testbed) en el ámbito de la IIoT [9], [22]-[24]. El artículo de Al-Hawawreh *et al.* [9] realiza un estudio para la creación de lo que denomina "Brown-IIoTbed", este sistema de pruebas está construido con un especial énfasis a los sistemas industriales con dispositivos heredados como PLC, RTU, SCADA, etc...

Spathoulas y Katsikas en su artículo [8] realizan un amplio estudio de los retos de seguridad en la implantación del IIoT.

Abbas *et al.* [25] desarrollan una taxonomía de los ataques a los sistemas IIoT, dividiendo estos ataques en 11 capas y 94 dimensiones describiendo una o más técnicas de ataque para cada una de estas dimensiones. Teniendo en cuenta la convivencia de los nuevos dispositivos propios del IIoT con las tecnologías heredadas. Para realizar esta taxonomía los autores han tenido en cuenta las relaciones existentes entre los dispositivos IIoT con las tecnológicas SCADA preexistentes.

En la literatura publicada actual se pueden encontrar diversas aproximaciones para la seguridad de las comunicaciones en el ámbito del IIoT. A continuación, podemos ver algunos de los más relevantes publicados en los últimos años.

Gutpa *et al.* en su artículo [26] desarrollan un sistema de autenticación mutua para los dispositivos propios del IIoT con un sistema de criptografía basado en la identidad (IBC por sus siglas en inglés) [27] y criptografía de curvas elípticas.

Li *et al.* [28] realizan un estudio de la aplicación del protocolo ID-2PAKA en los dispositivos IIoT.

Leevinson *et al.* [29] proponen un sistema de seguridad en el IIoT aplicando una plataforma de cadena de bloques en las comunicaciones entre los dispositivos del sistema.

Xiong *et al.* [30] realizan el desarrollo de un nuevo protocolo de autenticación que pretende preservar la privacidad en sistemas heterogéneos en el IIoT.

Hossein *et al.* [31] proponen un nuevo esquema de autenticación dirigido a las redes de sensores (WSN por sus siglas en inglés) del IIoT. Este nuevo esquema es denominado LAPTAS.

Patel *et al.* [32] definen un sistema para asegurar el protocolo MQTT. Realizan una autenticación entre el cliente y el broker, utilizando para ello el envío de credenciales cifradas con criptografía asimétrica en un componente JSON. El intercambio de mensajes se realiza mediante publicaciones previas a la primera suscripción.

Capítulo 3

MQTT-SCACAUTH: Comunicación Segura con MQTT

En esta sección se describe el método utilizado para asegurar la comunicación de datos en el ámbito de la industria 4.0. Se realiza la descripción de las adaptaciones del protocolo MQTT presentadas en esta tesis, que sin modificar en ningún caso el estándar de dicho protocolo, aseguran la identificación mutua entre los clientes y el broker, así como el cifrado de los datos intercambiados en las subsiguientes publicaciones, tanto entre los clientes y el broker como entre el broker y estos.

3.1. Protocolo MQTT

El protocolo MQTT (*Message Queuing Telemetry Transport*) fue definido por el Dr. Andy Stanford-Clark de IBM y por Arlen Nipper de Arcom (ahora Eurotech) en 1999 como un protocolo para el intercambio de datos de diferentes dispositivos de la industria de petróleo y gas con los sistemas SCADA. Estos datos eran enviados a través de conexiones vía satélite y se pretendió crear un protocolo interoperable para los diferentes dispositivos y con un alto ratio de información sobre el total de datos enviados y funcional con baja latencia para reducir el coste de conexión satelital. Este protocolo forma parte de un grupo de productos de IBM denominado "MQ Series".

En principio, este protocolo fue propietario de IBM, pero a partir de marzo de 2013 el protocolo MQTT está bajo proceso de normalización por parte de OASIS (*Advancing Open Standards for the Information Society*). El 29 de octubre de 2014 se publicó la primera estandarización abierta de este protocolo, en su versión v3.1.1 [s2], la última versión publicada por esta institución es la versión 5.0 [s1] publicada por primera vez el 25 de diciembre de 2017 y cuya última actualización, en el momento de la publicación de esta tesis, se produjo el 7 de marzo de 2019.

Es un protocolo de capa de aplicación, cuyo transporte se realiza sobre TCP/IP. Existe un protocolo creado para la integración de dispositivos que no tengan capacidad de comunicación vía TCP/IP con los sistemas basados en MQTT. Este protocolo se denomina MQTT-SN (*MQTT for Network Sensors*) y también esta estandarizado por parte de OASIS [s3]. Especialmente diseñado para cubrir la comunicación de redes de sensores inalámbricos (WSN), cuya comunicación se basa en otras tecnologías diferentes al TCP a nivel de capa de transporte como ZigBee, Z-wave u otras.

Los puertos estándar para este protocolo son 1883 para su transmisión directamente sobre TCP y el puerto 8883 para su transmisión sobre TLS [33].

El protocolo MQTT está basado en un esquema de publicación $\longleftrightarrow$ suscripción con dos tipos de participantes en la red.

- Broker: es el encargado de gestionar tanto las autorizaciones de conexión a los

clientes como de distribuir las publicaciones aceptadas para todos los suscriptores de dichas publicaciones.

- Clientes: son todos aquellos interlocutores que se conectan al broker para publicar mensajes (publicadores), recibir mensajes por medio de sus suscripciones (suscriptores) o ambas cosas.

Esta red tiene una topología en estrella (fig.3.1), donde el broker es el centro de dicha estrella, por lo que se tiene una total desconexión entre los publicadores y los suscriptores de los mensajes, no habiendo conexión directa entre los clientes participantes en la red.

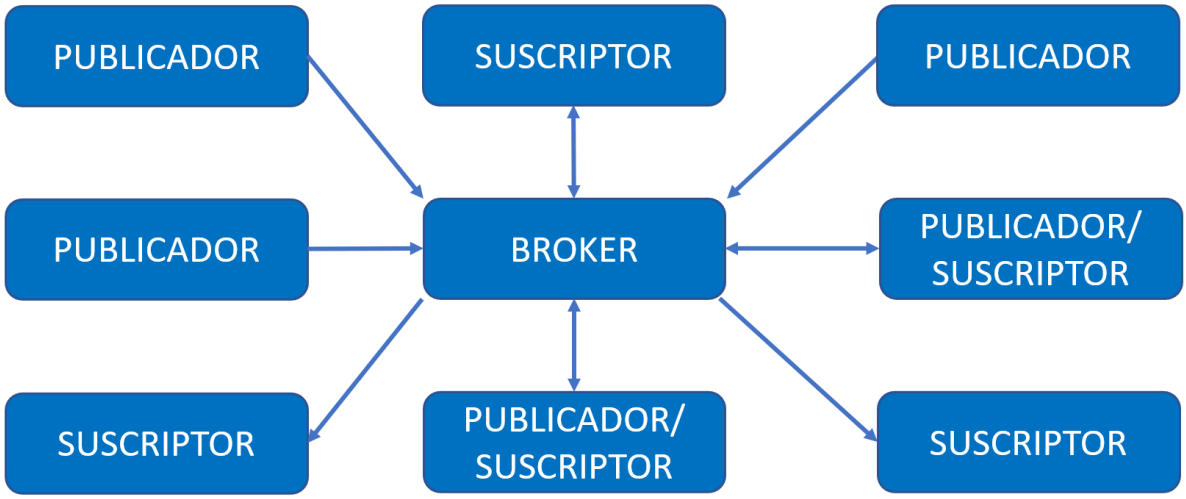


Figura 3.1: Topología MQTT

Una de las ventajas de este tipo de protocolos de publicación $\longleftrightarrow$ suscripción, con un mediador como centro de la red, es que los publicadores no tienen necesariamente que poder conectarse, ni conocer, a los suscriptores y viceversa. Solo es necesario que todos los publicadores y suscriptores (clientes) de la red puedan conectarse al broker.

Los mensajes se organizan en los denominados *topics*. Los clientes envían los datos que quieren transmitir como *payload* de un mensaje de publicación, asociados a un determinado *topic*, estos mensajes son recibidos, con la intermediación del broker,

por los clientes que se hayan suscrito a ese *topic*. Un ejemplo de este proceso sería el siguiente:

1. El cliente A se conecta al broker y es aceptado por este.
2. El cliente B se conecta al broker y es aceptado por este.
3. El cliente C se conecta al broker y es aceptado por este.
4. El cliente A se suscribe al *topic* "prueba" enviando una petición de suscripción al broker.
5. El cliente B se suscribe al *topic* "prueba" enviando una petición de suscripción al broker.
6. El cliente C realiza una publicación cuyo *topic* asociado es "prueba" y cuyo *payload* es "esto es una prueba".
7. El broker, una vez recibido el mensaje de publicación por parte del cliente C, envía un mensaje de publicación asociado al *topic* "prueba" y cuyo *payload* es "esto es una prueba" a todos los clientes que se han suscrito al *topic* "prueba", en este caso el cliente A y el cliente B.

Los *topics* se organizan por niveles de forma jerárquica (fig. 3.2), de tal modo que los suscriptores se pueden suscribir a un *topic* concreto o bien, y con la ayuda de caracteres especiales (*wilcard*), a los *topics* jerárquicamente dependientes de un *topic* dado.

Los *topics* son definidos como cadenas de caracteres codificadas en formato UTF-8, con distinción entre mayúsculas y minúsculas y con una longitud máxima de 65535 bytes. La separación de los niveles dentro de un *topic* se realiza con el carácter / (U+002F).

Existen dos *wildcard* validos disponibles para los suscriptores, multinivel # (U-0023) y para un solo nivel + (U+002B). Las suscripciones que se codifiquen con el carácter final # recibirán todos los mensajes jerárquicamente dependientes del *topic* anterior a dicho carácter.

La *wildcard* + sustituye a cualquier nivel jerárquico dentro de un *topic*.

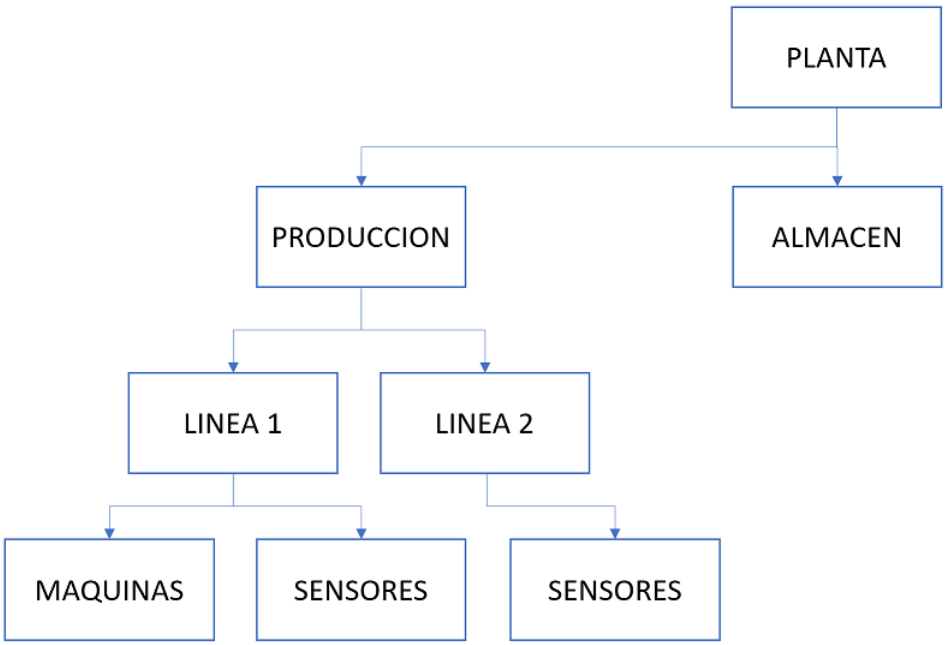


Figura 3.2: Ejemplo jerarquía *topics* MQTT

Partiendo del ejemplo de la fig. 3.2, un suscriptor que se suscriba al *topic* $PLANTA/PRODUCCION/\#$ recibirá todas las publicaciones realizadas en los *topics* jerárquicamente dependientes, es decir:

- PLANTA/PRODUCCION
- PLANTA/PRODUCCION/LINEA 1
- PLANTA/PRODUCCION/LINEA 1/MAQUINAS
- PLANTA/PRODUCCION/LINEA 1/SENSORES
- PLANTA/PRODUCCION/LINEA 2
- PLANTA/PRODUCCION/LINEA 2/SENSORES

Por otro lado, un suscriptor que se suscribiera al *topic* $PLANTA/PRODUCCION/+$, recibirá los mensajes publicados en los *topics*:

- PLANTA/PRODUCCION/LINEA 1
- PLANTA/PRODUCCION/LINEA 2

Pero no recibirá los mensajes publicados en:

- PLANTA/PRODUCCION
- PLANTA/PRODUCCION/LINEA 1/MAQUINAS
- PLANTA/PRODUCCION/LINEA 1/SENSORES
- PLANTA/PRODUCCION/LINEA 2/SENSORES

Este *wildcard* puede ser colocado para sustituir un nivel intermedio del *topic*, así un suscriptor al *topic* $PLANTA/PRODUCCION/+/SENSORES$ recibirá las publicaciones al *topic*:

- PLANTA/PRODUCCION/LINEA 1/SENSORES
- PLANTA/PRODUCCION/LINEA 2/SENSORES

Y no recibirá las publicaciones asociadas al resto de *topics*.

Por definición, este protocolo puede intercambiar datos con tres niveles de calidad de servicio (QoS). Podemos ver un resumen del intercambio de mensajes necesarios dependiendo de cada nivel de calidad de servicio en la fig. 3.3.

- QoS 0 At most once delivery: El mensaje se entrega de acuerdo con las capacidades de la red. No se envía ninguna respuesta por parte del receptor y el remitente no realiza ningún reintento. El mensaje llega al receptor una vez o ninguna.
- QoS 1 At least once delivery: El mensaje se entrega al menos una vez al receptor, el receptor debe enviar una respuesta al remitente, el remitente realiza reintentos si no llega la respuesta del receptor, por lo que el receptor puede recibir más de una vez el mensaje.

- QoS 2 Exactly once delivery: El mensaje se entrega una y solo una vez al receptor. El receptor envía una respuesta al remitente, el remitente realiza reintentos si no llega la respuesta del receptor. Cuando llega la respuesta del receptor, el remitente envía una confirmación de recepción de respuesta y el receptor debe enviar una respuesta a esta confirmación. De este modo, los dos interlocutores saben que el mensaje ha sido correctamente entregado. El mensaje llega una y una sola vez al receptor.

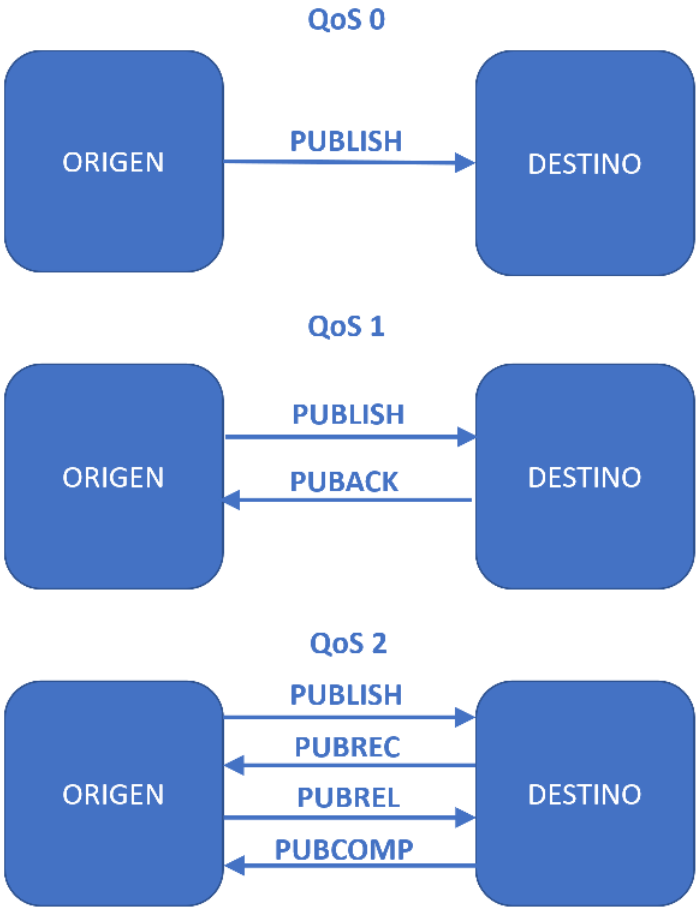


Figura 3.3: QoS disponibles en MQTT

Otra característica fundamental de este protocolo es su alto ratio de información útil transmitida en relación con el tamaño de los mensajes intercambiados. Los quince

tipos de mensajes que componen este protocolo (tabla 3.1) tienen una cabecera fija de entre 2 y 5 bytes y una cabecera variable con una componente necesaria por definición del protocolo y una parte de propiedades que pueden acompañar a determinados tipos de mensaje. Como final del mensaje se envía (en los mensajes que lo requieran) el *payload* con los datos a transmitir.

Tabla 3.1: Mensajes del protocolo MQTT [s1]

Nombre	**Valor**	**Flujo**	**Descripcion**
Reservado	0	No usado	Reservado
CONNECT	1	$C \Rightarrow B$	Petición de conexión
CONNACK	2	$B \Rightarrow C$	Respuesta a conexión
PUBLISH	3	$B \Rightarrow C$ o $C \Rightarrow B$	Publicación de mensaje
PUBACK	4	$B \Rightarrow C$ o $C \Rightarrow B$	Respuesta a una publicación (QoS 1)
PUBREC	5	$B \Rightarrow C$ o $C \Rightarrow B$	Respuesta a una publicación (QoS 2 parte 1)
PUBREL	6	$B \Rightarrow C$ o $C \Rightarrow B$	Respuesta a una publicación (QoS 2 parte 2)
PUBCOMP	7	$B \Rightarrow C$ o $C \Rightarrow B$	Respuesta a una publicación (QoS 2 parte 3)
SUBSCRIBE	8	$C \Rightarrow B$	Petición de suscripción
SUBACK	9	$B \Rightarrow C$	Respuesta a una petición de suscripción
UNSUBSCRIBE	10	$C \Rightarrow B$	Petición de anulación de suscripción
UNSUBACK	11	$B \Rightarrow C$	Respuesta a una petición de anulación de suscripción
PING	12	$C \Rightarrow B$	Petición de PING
PINGRESP	13	$B \Rightarrow C$	Respuesta a una petición de PING
DISCONNECT	14	$B \Rightarrow C$ o $C \Rightarrow B$	Notificación de desconexión
AUTH	15	$B \Rightarrow C$ o $C \Rightarrow B$	Intercambio de datos para la autentificación

En el siguiente ejemplo podemos ver los datos añadidos a la longitud del *payload* para el envío de datos pertenecientes al *topic* "PLANTA/ALMACEN/LINEA 1/SENSORES/SENSOR 1", para cada uno de los tres niveles de calidad del servicio disponibles, suponiendo un *payload* de 1 kilobyte de longitud. Los cálculos están realizados según el estándar MQTT v5.0 [s1], sin añadir propiedades adicionales a los envíos.

En este estudio se consideran los 40 bytes que componen el *topic* como datos no útiles parte del protocolo, si bien en muchas ocasiones el *topic* podría considerarse como datos útiles, ya que forman parte de la definición del significado del *payload*.

QoS 0: En este nivel de calidad de servicio se añaden como parte del protocolo 48 bytes a los 1024 bytes de datos útiles. Datos Totales, 1072 bytes, ratio: 95,522388 % datos útiles.

Los datos mínimos de protocolo para esta calidad de servicio, enviando un *payload* con 0 bytes, son 7 bytes, más la longitud de la cadena que defina el *topic* al que se envía.

- Mensaje PUBLISH:

  CABECERA FIJA: {0x30 0x84 0x2D}. DUP (indicador de reenvío):0 QoS:0 Retain:0, Longitud:3 bytes.

  CABECERA VARIABLE: {0x00 0x28} + *topic* + {0x00 0x01 0x00}. PacketId: 1, Longitud: 45 bytes.

  PAYLOAD: datos a transmitir. Longitud: 1024 bytes.

QoS 1: En este nivel de calidad de servicio se añaden como parte del protocolo 52 bytes a los 1024 bytes de datos útiles. Datos Totales, 1076 bytes, ratio: 95,167286 % datos útiles.

Los datos mínimos de protocolo en esta comunicación, para mandar un *payload* con 0 bytes, son 11 bytes, más la longitud de la cadena que defina el *topic* al que pertenece.

- Mensaje PUBLISH:

  CABECERA FIJA: {0x32 0x84 0x2D}. DUP:0 QoS:1 Retain:0, Longitud:3 bytes.

CABECERA VARIABLE: {0x00 0x28} + *topic* + {0x00 0x01 0x00}. PacketId: 1, Longitud: 45 bytes.

PAYLOAD: datos a transmitir. Longitud: 1024 bytes.

- Mensaje PUBACK:

  CABECERA FIJA: {0x40 0x02}. Longitud: 2 bytes.

  CABECERA VARIABLE: {0x00 0x01}. PacketId: 1. Longitud: 2 bytes[1]

QoS 2: En este nivel de calidad de servicio se añaden como parte del protocolo 60 bytes a los 1024 bytes de datos útiles. Datos Totales, 1084 bytes, ratio: 94,464944 % datos útiles.

Los datos mínimos de protocolo en esta comunicación, para mandar un *payload* con 0 bytes, son 19 bytes, más la longitud de la cadena que defina el *topic* al que pertenece.

- Mensaje PUBLISH:

  CABECERA FIJA: {0x34 0x84 0x2D}. DUP:0 QoS:2 Retain:0, Longitud:3 bytes.

  CABECERA VARIABLE: {0x00 0x28} + *topic* + {0x00 0x01 0x00}. PacketId: 1, Longitud: 45 bytes.

  PAYLOAD: datos a transmitir. Longitud: 1024 bytes.

- Mensaje PUBREC:

  CABECERA FIJA: {0x50 0x02}. Longitud: 2 bytes.

  CABECERA VARIABLE: {0x00 0x01}. PacketId: 1. Longitud: 2 bytes[1]

- Mensaje PUBREL:

  CABECERA FIJA: {0x62 0x02}. Longitud: 2 bytes.

  CABECERA VARIABLE: {0x00 0x01}. PacketId: 1. Longitud: 2 bytes[1]

[1]Según especificación [s1] en los mensajes PUBACK, PUBREL, PUBREC y PUBCOMP cuando la longitud de la cabecera variable es 2 no se incluye *Reason Code* y su valor se tiene que entender como 0x00 *Success*.

- Mensaje PUBCOMP:

  CABECERA FIJA: {0x70 0x02}. Longitud: 2 bytes.

  CABECERA VARIABLE: {0x00 0x01}. PacketId: 1. Longitud: 2 bytes[1]

La seguridad en el protocolo MQTT siempre ha sido un tema controvertido ya que en la especificación del protocolo la única autenticación nativa que existe es mediante usuario y contraseña, produciéndose así la autenticación del cliente en el broker pero imposibilitando la autenticación del broker por parte del cliente. Estos datos de autenticación viajaran en texto plano si utilizamos directamente el protocolo sobre TCP. Es por ello recomendable usarlo sobre TLS [33], pero la utilización de este protocolo en dispositivos de bajos recursos hace en muchas ocasiones imposible su utilización sobre TLS [34], debido a la gran cantidad de recursos necesarios para la utilización de este estándar. En la sección 5.4.1 de la especificación [s1] se muestra la posibilidad de utilizar un sistema externo de autenticación como LDAP [s4] o OAuth [35].

Por otro lado, en la versión 3.1.1. del protocolo [s2], la autenticación se debe realizar en un solo mensaje enviado por el cliente al servidor (CONNECT), donde se envía el usuario y password, y su respuesta (CONNACK). En la versión 5.0 del protocolo [s1] se incluye el mensaje AUTH para poder realizar autenticaciones en varios pasos, facilitando así la posibilidad de implementar protocolos de autenticación más complejos y la realización de autenticaciones mutuas, Broker-Cliente y Cliente-Broker.

3.2. Ventajas del protocolo MQTT

Existen diversos protocolos que aparecen en la bibliografía para su utilización en las comunicaciones relacionadas con el entorno del IoT.

En la mayoría de los artículos que realizan un estudio de los protocolos utilizados en el ámbito de la IoT se incluyen los siguientes protocolos MQTT, CoAP, AMQP y XMPP [36]-[40], además de estos protocolos comunes en algunos de estos artículos también se incluyen protocolos como DDS o API REST. En estos artículos podemos encontrar una descripción de estos protocolos, así como la comparación entre ellos.

Se considera la utilización de un protocolo basado en publicaciones con un intermediario central como la mejor opción para el intercambio de datos en el ámbito del IIoT. Este tipo de protocolos introduce diversas ventajas en la comunicación de datos de producción en la industria.

- Desconexión entre los clientes: Este modo de funcionamiento permite una mejor segmentación de las redes de las factorías, siendo únicamente necesaria la conexión de los clientes con el broker pero no entre ellos.
- Número y configuración de los receptores transparente para el productor de los datos: La distribución de los mensajes de un productor a varios receptores de manera transparente para el productor del dato es importante en el ámbito industrial. Un dato puede ser empleado para diversas funciones desde diversos sistemas receptores del mismo. El hecho de que los receptores de los mensajes puedan cambiar de manera transparente para el productor del dato es una ventaja en el momento de integrar nuevos sistemas o actualizar los existentes. Permite realizar estas modificaciones sin intervenir en los dispositivos productores de los datos.
- Datos jerarquizados: la posibilidad de realizar suscripciones a datos de manera jerarquizada permite que se introduzcan nuevos productores de datos en un área y que sus datos lleguen a los receptores interesados en ellos sin realizar ninguna modificación en los sistemas receptores.

Además de estas características para la aplicación de un protocolo en el ámbito industrial debemos tener en cuenta que se pueda utilizar con una calidad de servicio que nos asegure que los mensajes lleguen una y una sola vez a su destinatario. Por otro lado, para su introducción en dispositivos de bajos recursos, es necesario que su implementación tengo una baja huella de programa y que el intercambio de mensajes tenga un bajo coste computacional y de memoria.

El protocolo MQTT es el protocolo con una implementación más sencilla que cumpla con las características descritas. La sencillez en su implementación facilita su integración en dispositivos con bajos recursos. Por ello ha sido el protocolo utilizado en la realización de esta tesis.

3.3. Esquema General MQTT-SCACAUTH

Para alcanzar la seguridad en las comunicaciones en el protocolo MQTT, añadimos a los elementos estándar del protocolo tres nuevos elementos, que se enumeran a continuación (fig 3.4).

- Smart Card Criptográfica para clientes: Los clientes MQTT deberán incorporar una Smart Card Criptográfica (o un μcontrolador criptográfico equivalente, de bajo consumo y coste) que deberá comunicarse con el microcontrolador que maneje las comunicaciones con el broker.
- Sistema Criptográfico para el broker: El broker debe tener un sistema criptográfico, o bien una Smart Card criptográfica o bien un HSM (Hardware Security Manager) [41] o bien algún otro sistema criptográfico capaz de ejecutar los cifrados, tanto asimétricos como simétricos descritos posteriormente en este capítulo.

  El broker realiza varias funciones criptográficas para todos los mensajes enviados desde o hacia los clientes. Si el número de clientes y la frecuencia de mensajes provoca la superposición de mensajes, el broker deberá ejecutar varias operaciones criptográficas en paralelo con la mayor velocidad posible para ralentizar las comunicaciones lo mínimo posible. Debido a esta situación, salvo en instalaciones con muy pocos clientes o bien con muy pocos intercambios de datos entre ellos, el sistema criptográfico asociado al broker deberá ser un HSM o bien otro sistema criptográfico capaz de ejecutar varias funciones criptográficas en paralelo. No siendo adecuadas para este tipo de sistemas la utilización de Smart Card criptográficas. Para este tipo de sistemas también será necesaria la implementación del broker en sistemas balanceados para no llegar a ralentizar las comunicaciones por la falta de recursos del sistema donde se implemente el broker.
- Repositorio de claves públicas: El sistema debe tener un repositorio de claves públicas accesible mediante un protocolo seguro por parte del broker.

Para la implementación de este sistema se debe seleccionar los algoritmos criptográficos permitidos en las comunicaciones, tanto los algoritmos asimétricos utilizados en la autenticación (RSA_NOPAD, RSA_PKCS1, ECC, etc.) [42], como los

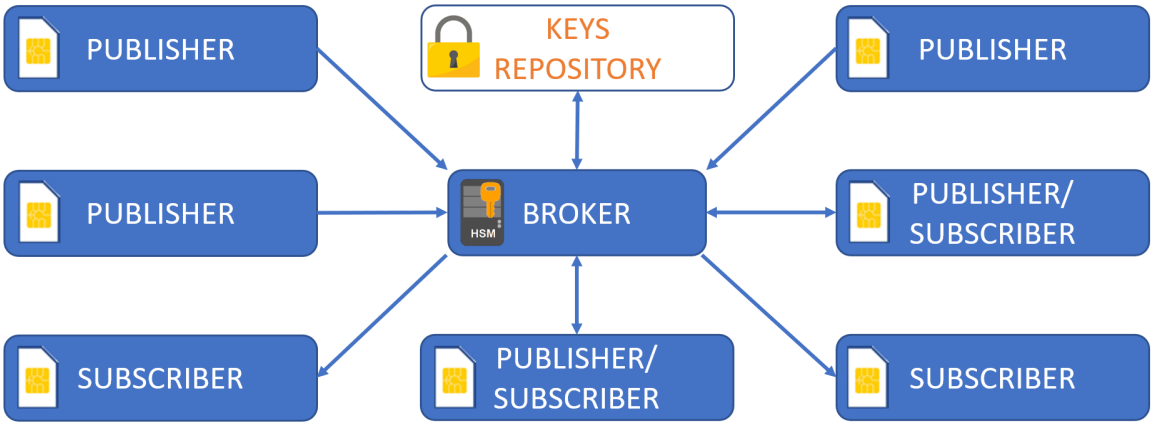


Figura 3.4: Elementos MQTT-SCACAUTH.

algoritmos simétricos de cifrado de bloques para el cifrado de los *payloads* y *topics* intercambiados (AES, DES, TEA, NOEKEON,etc.) [43].

La pareja de claves (pública y privada) para realizar la autenticación y los números aleatorios usados por los clientes del sistema deben ser generados directamente en las Smart Card criptográficas. La clave privada generada nunca debe comunicarse fuera de la Smart Card [44] y los números aleatorios solo se comunican fuera de la Smart Card cifrados, y se generan por medio de generadores de números aleatorios seguros.

Del mismo modo, el par de claves del broker y sus números aleatorios deben ser generados en el sistema criptográfico asociado al broker, su clave privada nunca debe abandonar el sistema criptográfico y los números aleatorios solo abandonan el sistema criptográfico cifrados y son generados por medio de generadores de números aleatorios seguros.

De esta manera, evitamos que estos datos sean revelados por un atacante usando técnicas de Man-in-the-Middle, definidas por NIST [45].

Todas las claves públicas son publicadas en el repositorio de claves del sistema. Cada clave pública esta asociada a un único identificador (UID), que identifica de manera única a cada uno de los elementos del sistema. Para los clientes del sistema el UID coincide con el identificador de cliente del protocolo MQTT, para el broker

su identificador es $BROKER$.

Por otro lado, la clave pública del broker es almacenada en todas las Smart Card criptográficas asociadas a todos los clientes que se conectan al sistema, antes de su primera conexión. Por otro lado, las claves de cada cliente se generan en su Smart Card criptográfica, y su clave publica se envía al almacén de claves de la instalación, mientras que su clave privada no abandona en ningún momento el entorno seguro de la Smart Card.

Todo el proceso de generación y almacenaje de claves de los clientes se realiza antes de la primera conexión del cliente al sistema.

De este modo, todas las Smart Card criptográficas de los clientes tienen almacenadas:

- Clave Privada Cliente. Se genera en la propia Smart Card y nunca se comunica fuera de esta.
- Clave Pública Cliente. Se genera en la propia Smart Card y se comunica de forma segura al repositorio de claves públicas del sistema.
- Clave Pública Broker. Se genera en el sistema criptográfico del broker y se comunica a la Smart Card criptográfica de forma segura.

Por su lado, el sistema criptográfico del broker almacena:

- Clave Privada Broker. Se genera en el propio sistema criptográfico y nunca se comunica fuera de este.
- Clave Pública Broker. Se genera en el propio sistema criptográfico y se comunica de forma segura al repositorio de claves públicas del sistema.

Por último, el repositorio de claves almacena todas las claves públicas válidas del sistema, relacionadas con el UID del elemento al que correspondan.

Es necesario que el intercambio de mensajes entre participantes asegure que cada mensaje sea recibido una vez, y sólo una vez, por el destinatario previsto, para aplicar el esquema de seguridad propuesto en este documento y garantizar la seguridad en

las comunicaciones. Por lo que se requiere el uso del protocolo MQTT con nivel de calidad de servicio (QoS) igual a dos [46].

En este capítulo y los siguientes usaremos la siguiente notación (Tabla 3.2) para referirnos a los procesos de intercambio de datos entre los participantes.

Tabla 3.2: Símbolos utilizados

Notación	**Descripción**
UID	Identificador único de cada participante
$Cpr_b(a)$	Resultado del cifrado del dato "a" con la clave privada del broker
$Cpu_b(a)$	Resultado del cifrado del dato "a" con la clave pública del broker
$Cpr_{cx}(a)$	Resultado del cifrado del dato "a" con la clave privada del cliente x
$Cpu_{cx}(a)$	Resultado del cifrado del dato "a" con la clave pública del cliente x
$BCE_x(a)$	Resultado del cifrado del dato "a" con el algoritmo de cifrado de bloques (criptografía simétrica) con la clave x
RN_a	Número aleatorio a
$A;B$	Dato A concatenado con Dato B
$B64(a)$	Función que codifica un array de bytes (a) en su representación en codificación Base64[s5] con conjunto de caracteres UTF-8

3.3.1. Autenticación

Se propone una autenticación mutua para este desarrollo, es decir se autenticará tanto la identidad del cliente por parte del broker, como la del broker por parte del cliente.

Para realizar esta autenticación mutua se propone un proceso en tres pasos que podemos ver gráficamente representado en la fig. 3.5, en este proceso además de realizar la autenticación los participantes intercambian los números aleatorios necesarios para iniciar el intercambio de mensajes de manera segura.

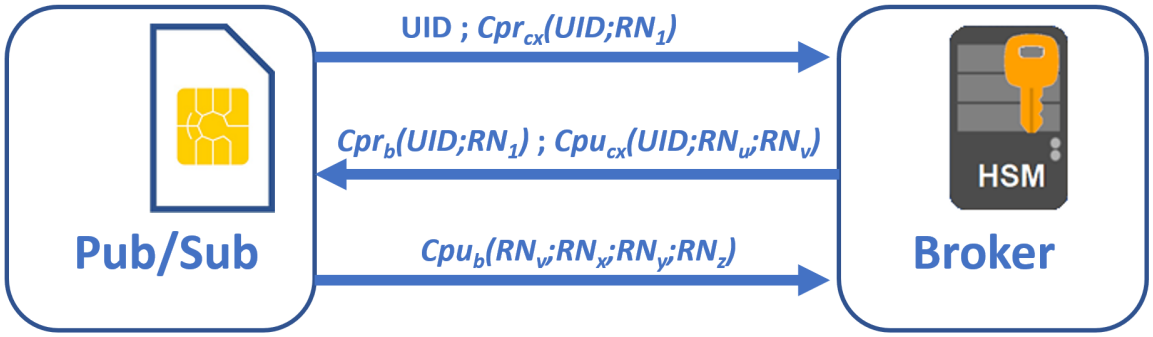


Figura 3.5: Proceso de autenticación mutua.

3.3.1.1. Paso 1: Autenticación del Cliente en el Broker

La autenticación mutua se realiza a iniciativa del cliente, es por ello que el primer paso de esta autenticación se realiza por parte de este. El cliente manda un mensaje de conexión al broker, en este mensaje, el cliente envía su identificador único (UID) y concatenado a este añade el resultado del cifrado, mediante el algoritmo de criptografía asimétrica utilizado con su clave privada, del resultado de concatenar su UID con un número aleatorio generado por su Smart Card criptográfica, número que designaremos como RN_1. Este número aleatorio impide que en las diferentes autenticaciones que el cliente realice contra el broker el resultado del cifrado con su clave privada sea igual al resultado de cualquier otro intento de autenticación anterior.

$$Cliente_x \xrightarrow{UID;Cpr_{cx}(UID;RN_1)} Broker$$

El broker recibe este mensaje y con el UID, que viaja como texto plano en la primera parte de los datos enviados por el cliente, obtiene del repositorio público la correspondiente clave pública del cliente que, supuestamente, ha iniciado la conexión.

Una vez obtenida la clave pública del cliente, el broker continúa con el proceso (fig. 3.6), intentando descifrar la segunda parte del mensaje con dicha clave pública. Si descifra sin errores estos datos pasa a la siguiente fase del proceso.

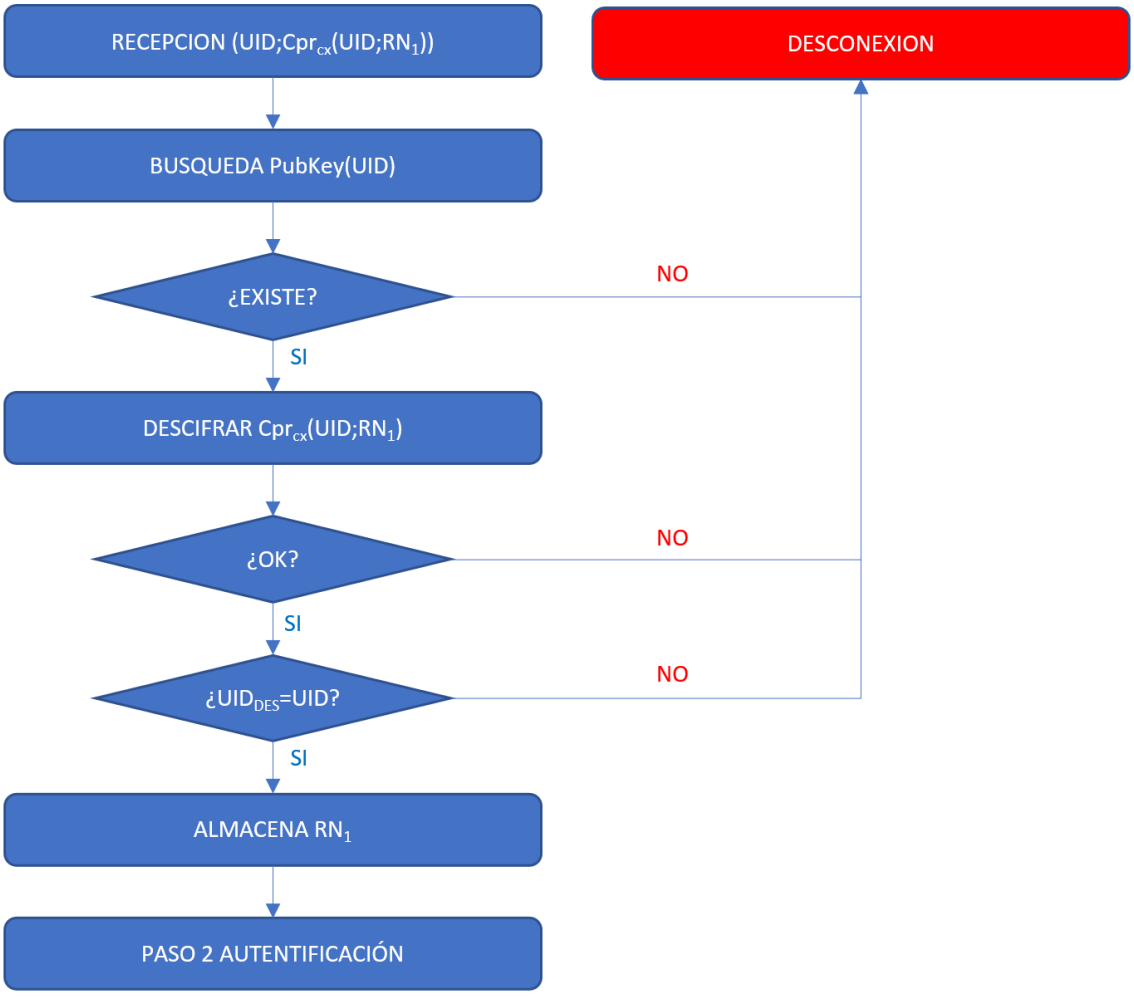


Figura 3.6: Comprobación paso 1 autenticación mutua.

Con la primera parte de los datos descifrados realiza una comparación con el UID de la primera parte que fueron transmitidos como texto plano. Si es correcto, significa que la segunda parte del mensaje ha sido cifrada con la clave privada del cliente correspondiente al UID leído, siendo la clave privada del cliente únicamente conocida por este, se ha realizado la autenticación del cliente por parte del broker.

El broker ha comprobado que el cliente es quien dice ser sin posibilidad de engaño.

La segunda parte del mensaje correspondiente al número aleatorio RN_1 se almacena para su futuro uso en el siguiente paso de la autenticación. Este número, al haber sido generado internamente en la Smart Card criptográfica del cliente, es conocido por ambos. Sin embargo, al haber sido descifrado con la clave pública del cliente, no aseguramos que no haya sido desvelado por un atacante, que pueda conocer dicha clave pública. Por lo que en los siguientes pasos de la autenticación este valor no es considerado un valor secreto.

Ante cualquier fallo en el proceso descrito anteriormente para la verificación de la identidad del cliente por parte del broker, este cortará la conexión con el cliente y este deberá volver a iniciar de nuevo el proceso de autenticación mutua para poder conectarse con el broker.

3.3.1.2. Paso 2: Autenticación del Broker en el Cliente

Una vez terminado el paso 1 de la autenticación, el broker crea y envía un mensaje al cliente para su autenticación.

Este mensaje se compone de dos partes concatenadas, la primera parte es el cifrado con la clave privada del broker de los datos enviados cifrados con la clave privada del cliente en el paso anterior de la autenticación. En la segunda parte de los datos enviados se cifra con la clave pública del cliente el UID del cliente concatenado con dos números aleatorios generados por el broker, que llamaremos RN_u y RN_v.

$$Broker \xrightarrow{Cpr_b(UID;RN_1);Cpu_{cx}(UID;RN_u;RN_v)} Cliente_x$$

En el momento que el cliente recibe estos datos, el primer paso que realiza es el descifrado de los datos de la primera parte con la clave pública del broker. Si el descifrado es posible y los datos resultantes de este descifrado corresponden con el dato enviado cifrado en el primer paso de la autenticación, aseguramos que el cifrado ha sido realizado con la clave privada del broker. Como esta clave es solo conocida por el broker, podemos asegurar que este mensaje ha sido enviado por este, realizando así la segunda parte de la autenticación mutua.

El cliente ha comprobado sin posibilidad de engaño que el broker es quien dice ser.

La segunda parte del envío se descifra por parte del cliente con su clave privada, así solo él que es el único conocedor de esta clave puede descifrarlo.

Comprobado que el proceso de descifrado ha finalizado de forma correcta, y además la primera parte del mensaje contiene su UID, se almacenarán los números aleatorios RN_u y RN_v. Estos números se pueden considerar secretos y únicamente conocidos por el broker y el cliente. El broker los ha generado dentro de su sistema criptográfico y únicamente han abandonado ese entorno cifrados con la clave publica del cliente. Este dato solo podrá ser descifrado por la clave privada del cliente por lo que estamos seguros de que únicamente son conocidos por el sistema criptográfico del broker que los ha generado y por la Smart Card criptográfica del cliente que los ha descifrado con su clave privada que solamente él conoce. Estos datos además nunca abandonan el entorno seguro de su Smart Card criptográfica, quien los ha descifrado, ni el entorno seguro del sistema criptográfico del broker quien los ha generado.

Podemos ver el diagrama de flujo de la comprobación del paso 2 de la autenticación por parte del cliente en la fig. 3.7.

3.3.1.3. Paso 3: Finalización de la Autenticación

Cuando el cliente finaliza la comprobación del paso dos de la autenticación, generará el array resultante de concatenar el RN_v, recibido en el paso anterior, con tres números aleatorios generados en su Smart Card criptográfica, que denominaremos RN_x, RN_y y RN_z. Este array resultante es encriptado con la clave pública del broker, siendo desarrollado todo este proceso en su Smart Card criptográfica.

Estos datos cifrados son enviados por parte del cliente al broker.

$$Cliente_x \xrightarrow{Cpu_b(RN_v;RN_x;RN_y;RN_z)} Broker$$

En el momento que el broker recibe estos datos, el primer paso que realiza es el descifrado de los datos con su clave privada. Si el descifrado es posible y la primera parte de los datos coinciden con el RN_v almacenado en el paso anterior, almacena

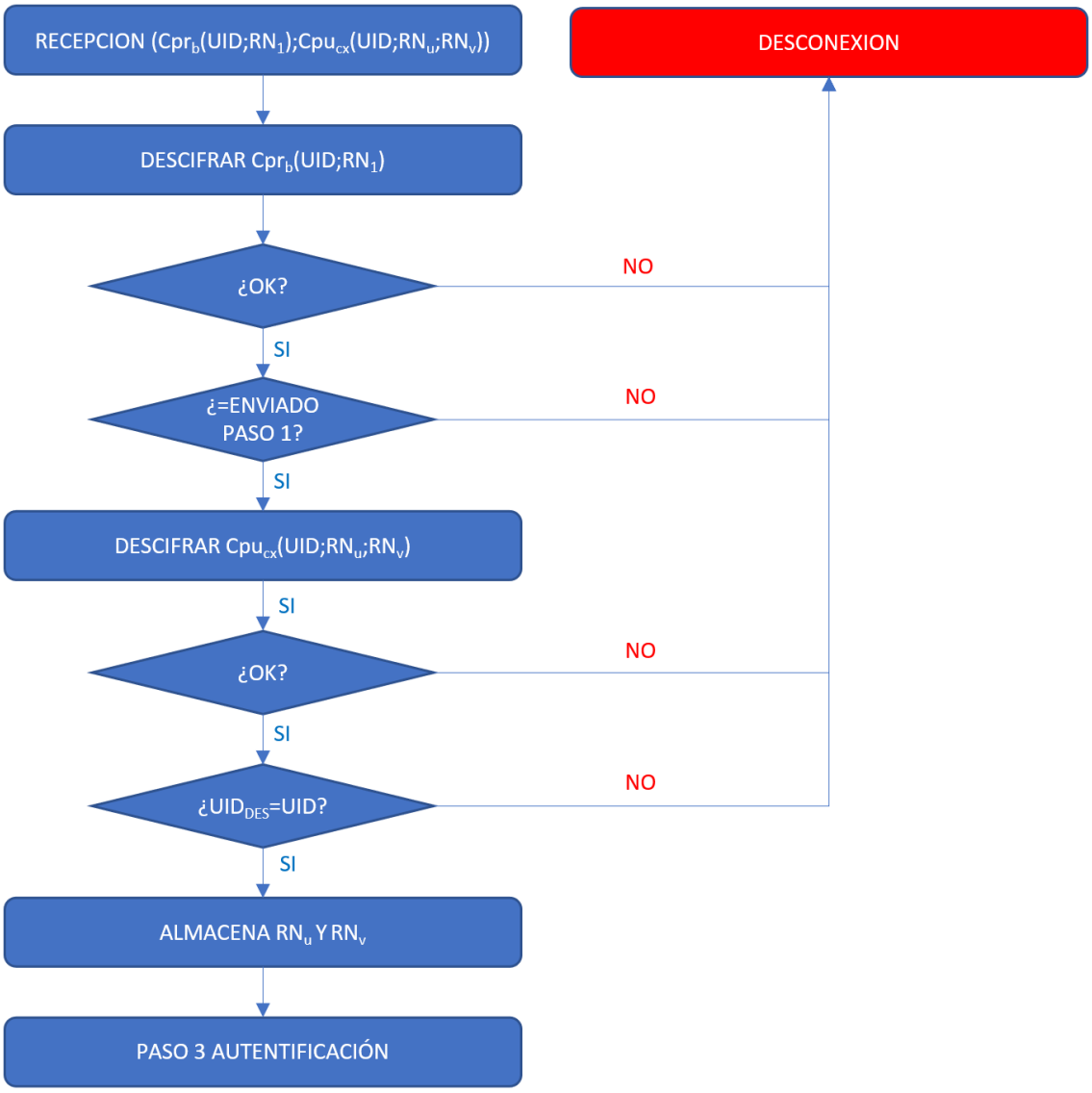


Figura 3.7: Comprobación paso 2 autenticación mutua.

para su posterior utilización los números aleatorios enviados por el cliente. Como estos números han sido generados en la Smart Card criptográfica del cliente y solo la han abandonado cifrados con la clave pública del broker, cifrado que solamente puede descifrar el broker con su clave privada que únicamente él conoce, se consi-

deran seguros y únicamente conocidos por el cliente y el broker participantes en la autenticación.

Si durante el descifrado o bien en la comparación del RN_v hay algún problema, el broker cortará la conexión y dará por finalizada con error la autenticación, si todo el proceso ha sido correcto almacenará los números aleatorios obtenidos y dará por finalizada la autenticación de forma satisfactoria, pudiéndose a partir de entonces intercambiar mensajes de publicación y suscripción.

Podemos ver el diagrama de flujo de la comprobación del paso 3 de la autenticación por parte del cliente en la fig. 3.8.

Los números aleatorios obtenidos durante este proceso, considerados seguros y conocidos por ambos participantes se utilizarán como clave de cifrado simétrico en los primeros cifrados de los datos a intercambiar:

- RN_u: clave de cifrado para el *payload* del primer mensaje PUBLISH enviado del cliente al broker.
- RN_v: clave de cifrado para el *topic* del primer mensaje PUBLISH enviado del cliente al broker.
- RN_x: clave de cifrado para el *payload* del primer mensaje PUBLISH enviado del broker al cliente.
- RN_y: clave de cifrado para el *topic* del primer mensaje PUBLISH enviado del broker al cliente.
- RN_z: clave de cifrado para el *topic* del primer mensaje SUBSCRIBE enviado del cliente al broker.

Como veremos en las siguientes secciones de este documento, estos números aleatorios únicamente son empleados en el primer mensaje de cada tipo, después se renuevan de forma segura en cada mensaje enviado.

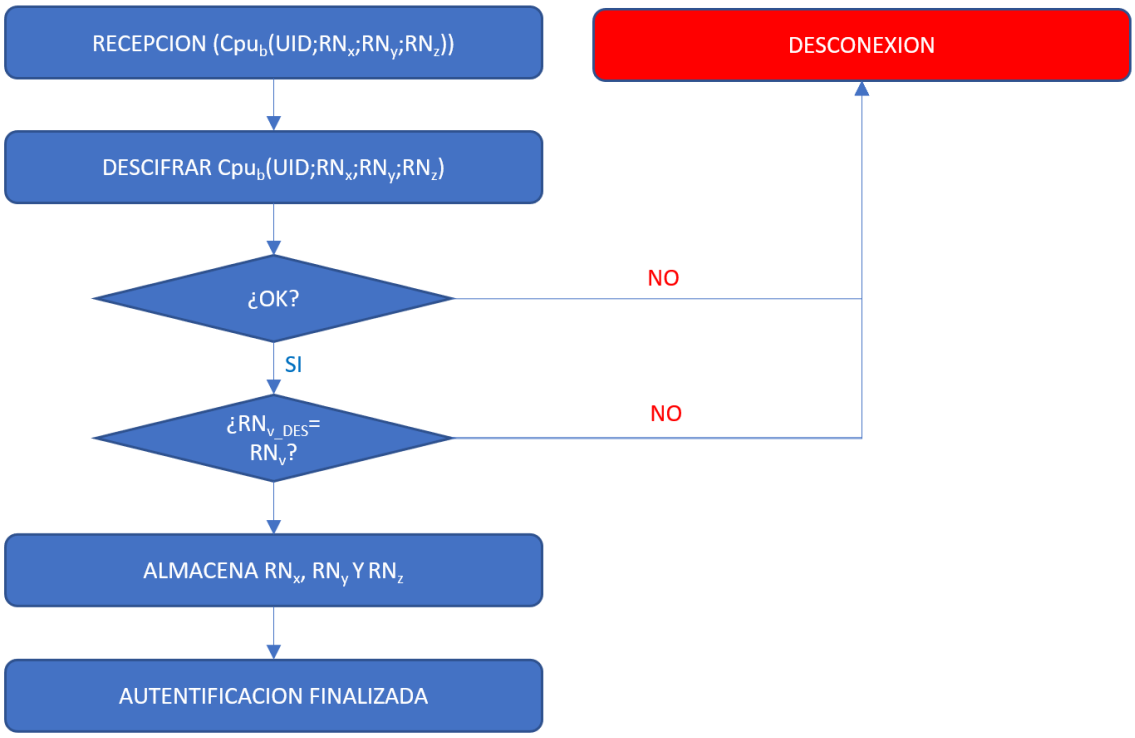


Figura 3.8: Comprobación paso 3 autenticación mutua.

3.3.2. Envío de Publicaciones

Para llevar a cabo el cifrado tanto del *payload* como de los *topics* de las publicaciones, usaremos un algoritmo simétrico de cifrado de bloques. Se utiliza este tipo de algoritmo porque su tiempo de procesamiento es mucho menor que los algoritmos de criptografía asimétrica [47].

3.3.2.1. Cliente a Broker

En el proceso de envío de publicaciones desde un cliente (publicador) hasta el broker (fig.3.9), el publicador envia su UID concatenado con el *payload* y el resulta-

do de esta concatenación se cifra con el algoritmo de cifrado de bloques seleccionando usando una clave aleatoria, únicamente utilizada en un intercambio de datos y únicamente conocida por el broker y el publicador, que llamaremos RN_{un}, en el primer mensaje de este tipo intercambiado RN_{un} es igual a RN_u obtenido en el proceso de autenticación. El *topic* donde se publica este *payload* es enviado también concatenado con el UID del publicador y cifrado con el algoritmo de cifrado de bloques seleccionado, usando una clave aleatoria únicamente utilizada para un intercambio de datos y únicamente conocida por el broker y el publicador, que llamaremos RN_{vn}, en el primer mensaje de este tipo intercambiado RN_{vn} es igual a RN_v obtenido en el proceso de autenticación.

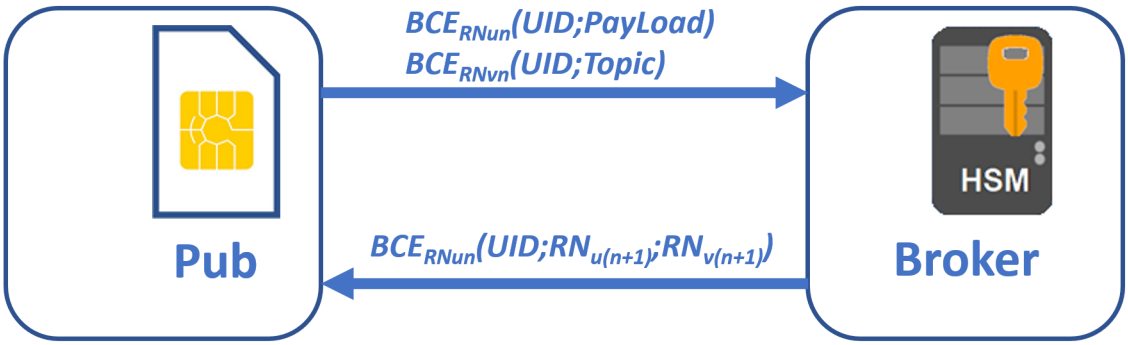


Figura 3.9: Publicación datos de cliente a broker.

La utilización de cada clave únicamente en un solo intercambio de datos previene, además de contra los ataques de tipo Man-in-the-Middle, contra los ataques de repetición de mensajes (replay-attack), expuesto por M. Shuai *et al.* en su artículo [48], así como ante ataques basados en análisis estadísticos, como los ataques *Statistical Disclosure Attack* (SDA por sus siglas en inglés) descritos por M. Emandoost *et al.* en [49].

$$Cliente_x \xrightarrow{payload=BCE_{RNun}(UID;Payload)\ \ topic=BCE_{RNvn}(UID;Topic)} Broker$$

Una vez el broker recibe este mensaje, trata de descifrar el mensaje con el algoritmo seleccionado, para el *payload* utiliza la clave RN_{un} y para el *topic* utiliza RN_{vn} asignadas al cliente, una vez realizados los descifrados de ambos componentes compara los $UIDs$ obtenidos con el UID del cliente que supuestamente ha enviado

el mensaje. Si es correcto trata el mensaje como si de un mensaje estándar se tratara (fig 3.10). En el caso de que las comprobaciones pertinentes no resulten satisfactorias el broker envía un mensaje de desconexión al cliente y cierra la conexión con este.

Figura 3.10: Comprobación por parte del broker de una publicación de un cliente.

En la respuesta a este mensaje, el broker envía al publicador un mensaje compuesto por tres componentes concatenados y cifrados con el algoritmo de cifrado de bloques configurado en el sistema y como clave de cifrado utilizaremos el número aleatorio RN_{un}, estos tres componentes son:

1. UID del publicador.
2. Número aleatorio generado por el sistema criptográfico del broker al que llamaremos $RN_{u(n+1)}$.
3. Número aleatorio generado por el sistema criptográfico del broker al que llamaremos $RN_{v(n+1)}$.

$$Broker \xRightarrow{BCE_{RNun}(UID;RN_{u(n+1)};RN_{v(n+1)})} Cliente_x$$

Una vez el mensaje haya sido enviado, el broker sustituye para futuros intercambios de mensajes los número aleatorios RN_{un} y RN_{vn} utilizado para este intercambio por los nuevos números aleatorios generados $RN_{u(n+1)}$ y $RN_{v(n+1)}$, para su utilización en los siguientes intercambios de mensajes con este cliente actuando como publicador.

El cliente descifra el mensaje con el algoritmo de bloques seleccionado y como clave utiliza RN_{un}, comprueba que el UID que aparece en la primera parte del mensaje descifrado es su UID y si es correcto almacena los números aleatorios obtenidos como los nuevos números que servirán de claves para siguientes intercambios de mensajes de este tipo (fig. 3.11). Si las comprobaciones pertinentes por parte del cliente no son satisfactorias, este envia un mensaje de desconexión al broker y cierra su conexión.

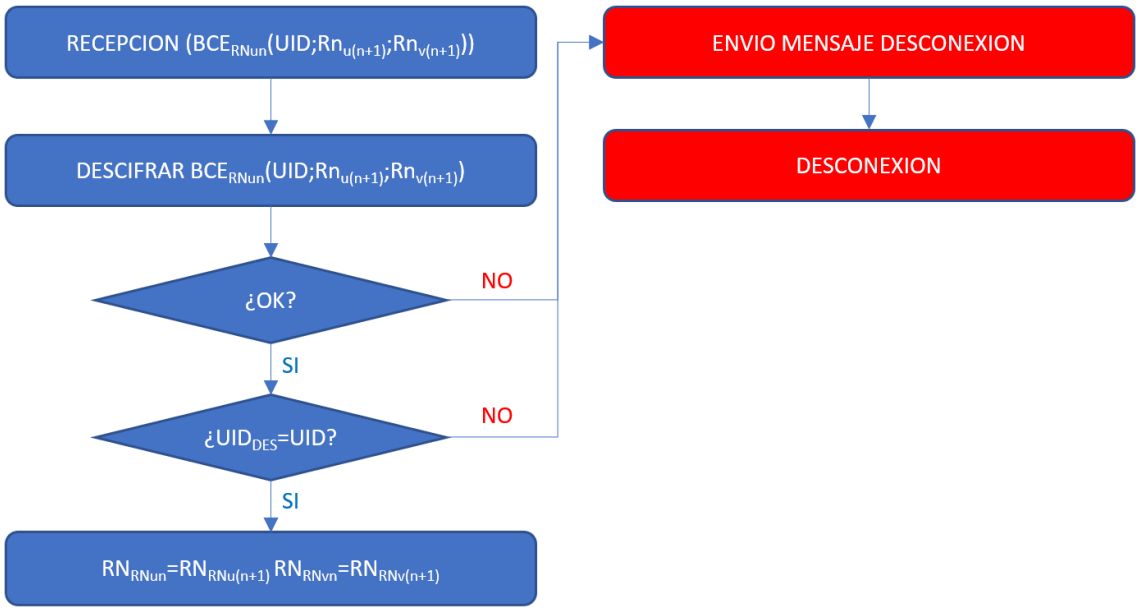


Figura 3.11: Comprobación por parte cliente del mensaje en respuesta a una de sus publicaciones.

3.3.2.2. Broker a Cliente

En el proceso de envío de mensajes de publicación desde el broker a un cliente (suscriptor) (fig. 3.12) se utiliza un proceso de cifrado similar al utilizado en el envío de un cliente al broker.

El broker envia el UID del suscriptor destinatario de la publicación concatenado con el *payload* y el resultado de esta concatenación se cifra con el algoritmo de cifrado de bloques seleccionando usando una clave aleatoria, únicamente utilizada en un intercambio de datos y únicamente conocida por el broker y el suscriptor, que llamaremos RN_{xn}, en el primer mensaje de este tipo intercambiado RN_{xn} es igual a RN_x obtenido en el proceso de autenticación. El *topic* donde se publica este *payload* es enviado también concatenado con el UID del suscriptor y cifrado con el algoritmo de cifrado de bloques seleccionado usando una clave aleatoria únicamente utilizada para un intercambio de datos y únicamente conocida por el broker y el suscriptor, que llamaremos RN_{yn}, en el primer mensaje de este tipo intercambiado RN_{yn} es

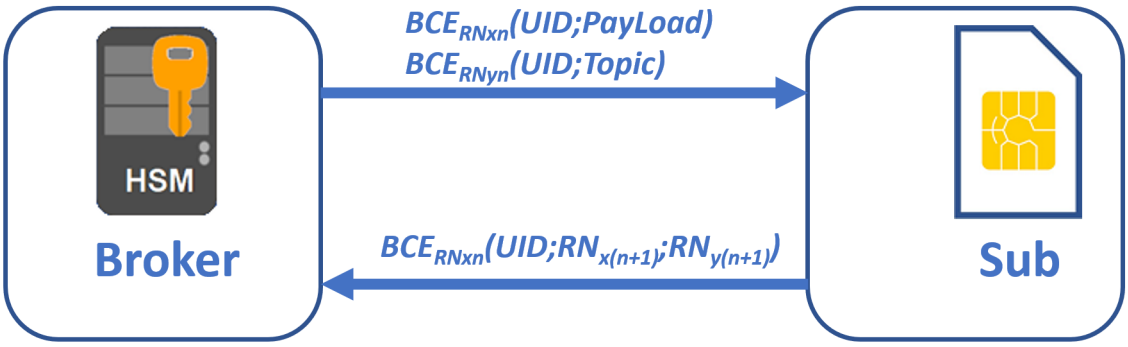


Figura 3.12: Publicación datos de broker a cliente.

igual a RN_y obtenido en el proceso de autenticación.

$$Broker \xrightarrow{payload=BCE_{RNxn}(UID;Payload)\ \ topic=BCE_{RNyn}(UID;Topic)} Cliente_x$$

Una vez el suscriptor recibe este mensaje, trata de descifrarlo con el algoritmo seleccionado, para el *payload* utiliza la clave RN_{xn} y para el *topic* utiliza RN_{yn} que tiene almacenadas en su Smart Card criptográfica, una vez realizados los descifrados de ambos componentes compara los $UIDs$ obtenidos con su UID, si es correcto trata el mensaje como si de un mensaje estándar se tratara (fig 3.13). En el caso de que las comprobaciones pertinentes no resulten satisfactorias, el cliente envía un mensaje de desconexión al broker y cierra la conexión con este.

Figura 3.13: Comprobación por parte del cliente del mensaje publicado por el broker.

En respuesta a este mensaje, el cliente suscriptor envía al broker un mensaje compuesto por tres componentes concatenados y cifrados con el algoritmo de cifrado de bloques configurado en el sistema y como clave de cifrado utiliza el número aleatorio RN_{xn}, estos tres componentes son:

1. UID del suscriptor.
2. Número aleatorio generado por Smart Card criptográfica del suscriptor al que llamaremos $RN_{x(n+1)}$.
3. Número aleatorio generado por Smart Card criptográfica del suscriptor al que llamaremos $RN_{y(n+1)}$.

$$Cliente_x \xrightarrow{BCE_{RNxn}(UID;RN_{x(n+1)};RN_{y(n+1)})} Broker$$

Una vez el mensaje haya sido enviado, el suscriptor sustituye para futuros intercambios de mensajes los número aleatorios RN_{xn} y RN_{yn} utilizado para este intercambio por los nuevos números aleatorios generados $RN_{x(n+1)}$ y $RN_{y(n+1)}$, para su utilización en los siguientes intercambios de mensajes de este cliente actuando como suscriptor.

El broker descifra el mensaje con el algoritmo de bloques seleccionado y como clave utiliza RN_{xn}, comprueba que el UID que aparece en la primera parte del mensaje descifrado corresponde con el UID del cliente suscriptor y, si es correcto, almacena los números aleatorios obtenidos como los nuevos números que servirán de claves para siguientes intercambios de mensajes de este tipo con este cliente (fig. 3.14). Si las comprobaciones pertinentes por parte del broker no son satisfactorias, este envía un mensaje de desconexión al suscriptor y cierra su conexión.

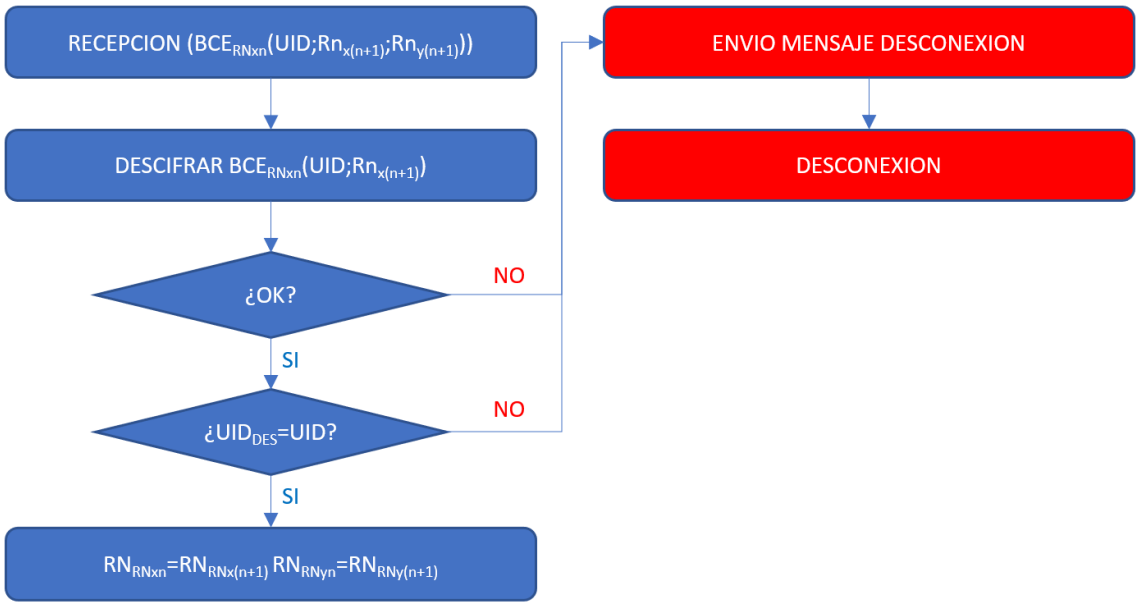


Figura 3.14: Comprobación por parte broker del mensaje de respuesta enviado por el cliente suscriptor.

3.3.3. Subscripciones

Los clientes deben enviar mensajes para suscribirse a los *topics* de los que desean recibir publicaciones. También tienen la opción, después de realizar una suscripción, de anular dicha suscripción (fig.3.15). En ambos casos el cliente suscriptor es el que inicia el intercambio de mensajes para realizar la acción y en ambos casos se envía el UID del cliente concatenado con el *topic* objeto de la suscripción cifrado con el algoritmo de cifrado de bloques seleccionado utilizando una clave aleatoria, únicamente utilizada en un intercambio de datos y únicamente conocida por el broker y el suscriptor, que llamaremos RN_{zn}, en el primer mensaje de este tipo intercambiado RN_{zn} es igual a RN_z obtenido en el proceso de autenticación.

$$Cliente_x \xrightarrow{BCE_{RNzn}(UID;Topic)} Broker$$

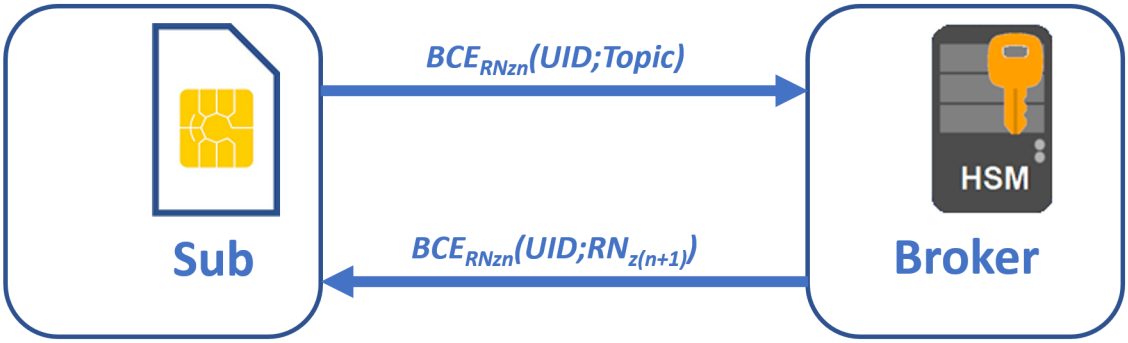


Figura 3.15: Mensaje de suscripción desde suscriptor a broker.

Una vez el broker recibe este mensaje, trata de descifrarlo con el algoritmo seleccionado, utiliza la clave RN_{zn} que tiene almacenada asociada al cliente que envía el mensaje, una vez realizado el descifrado compara el UID con el del cliente que supuestamente ha enviado el mensaje, si es correcto trata el mensaje como si de un mensaje estándar se tratara (fig 3.16). En el caso de que las comprobaciones pertinentes no resulten satisfactorias el broker envía un mensaje de desconexión al cliente y cierra la conexión con este.

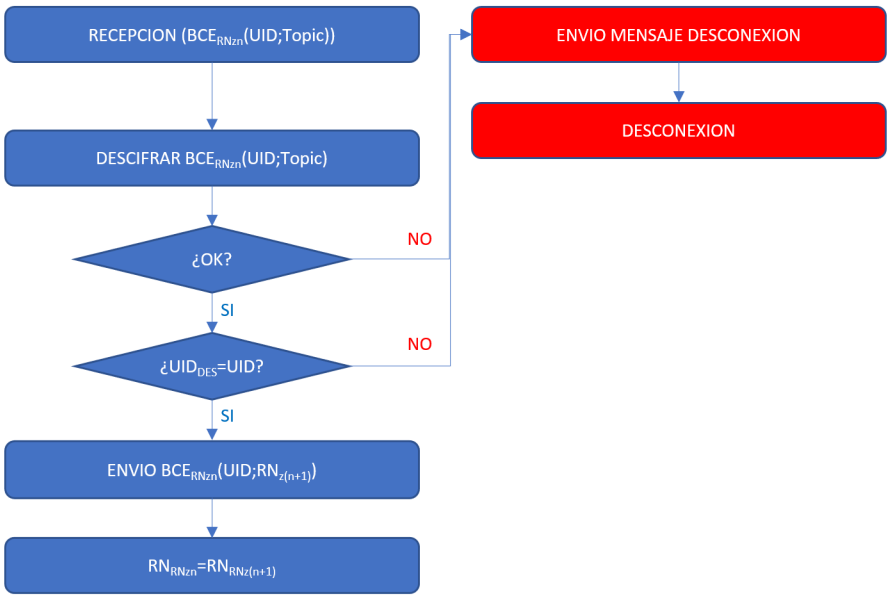


Figura 3.16: Comprobación por parte broker del mensaje de suscripción enviado por el cliente suscriptor.

Si la comprobación ha sido satisfactoria el broker envía un mensaje de respuesta, concatenando el UID del cliente suscriptor con un nuevo número aleatorio generado por su sistema criptográfico al que llamaremos $RN_{z(n+1)}$, todo ello cifrado con el algoritmo de cifrado de bloques, utilizando como clave el número aleatorio RN_{zn} una vez realizado el cifrado sustituye el número aleatorio RN_{zn} por el nuevo número aleatorio generado $RN_{z(n+1)}$.

El cliente suscriptor, una vez recibido este mensaje de respuesta, descifra el contenido del mensaje con el algoritmo de cifrado de bloques seleccionado, una vez descifrado compara su UID con el UID recibido en el mensaje. Si el proceso ha sido finalizado correctamente el cliente reemplaza su RN_{zn} con el número aleatorio recibido $RN_{z(n+1)}$ (3.17). Si la comparación no ha sido satisfactoria el cliente envía un mensaje de desconexión al broker.

$$Broker \xrightarrow{BCE_{RNzn}(UID;RN_{z(n+1)})} Cliente_x$$

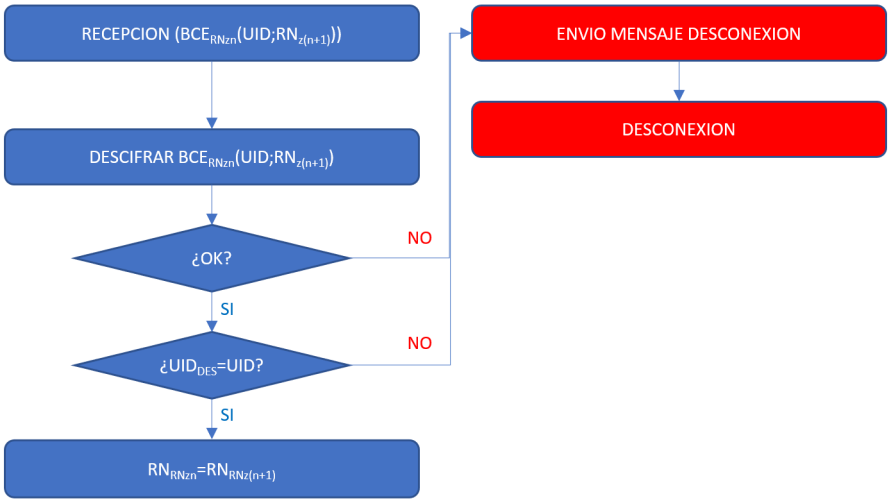


Figura 3.17: Comprobación por parte cliente del mensaje respuesta a la suscripción enviado por el broker.

Capítulo 4

MQTT-SCACAUTH: Detalle de las adecuaciones del protocolo

En el presente capitulo se desarrollan las adecuaciones del protocolo MQTT para generar comunicaciones seguras. Detallando las acciones realizadas en cada uno de los mensajes que intervienen en el intercambio seguro, tanto para la autenticación mutua del broker con el cliente y viceversa, como para el cifrado de datos en las comunicaciones. Este desarrollo fue presentado en el artículo [50], del cual el autor de esta tesis es autor principal. A partir de este artículo se ha continuado con la investigación para la realización de esta tesis, añadiendo algunas mejoras al esquema propuesto en dicho artículo. SCACAUTH es el acrónimo de *Smart Card Asymmetric Cryptography Authentication*.

4.1. Premisas

Se propone, siguiendo el estándar MQTT 5.0 [s1], la inclusión de los datos necesarios para realizar tanto la autenticación mutua entre el broker y los clientes del sistema, así como el cifrado de los datos intercambiados (*payloads* y *topics*), siguiendo el esquema de seguridad MQTT-SCACAUTH propuesto en la sección 3.3 del presente documento.

Para esta implementación partiremos de una serie de premisas iniciales, de obligatorio cumplimiento:

- Se definen los números aleatorios utilizados en el intercambio de mensajes entre el broker y el cliente y viceversa ($RN_?$), con la misma longitud que la longitud de la clave utilizada por el algoritmo de cifrado de bloques seleccionado en el esquema de seguridad. [SCACAUTH-PRE-1]
- Se define el identificador único de los clientes (UID), como una cadena de 8 caracteres, codificada en formato UTF-8. Los caracteres válidos para esta definición serán *"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"*, según la declaración normativa obligatoria [MQTT-3.1.3-5] del estándar MQTT 5.0. [SCACAUTH-PRE-2]
- Todos los mensajes de publicación enviados por el cliente o el broker deben ser obligatoriamente realizados con un nivel de calidad de servicio (QoS) igual a dos, asegurando que cada mensaje intercambiado llegue a su destinatario una y una única vez. [SCACAUTH-PRE-3]
- No será posible el envío de un mensaje de publicación por parte de un participante, hasta que haya finalizado todo el proceso de intercambio de mensajes definidos para QoS 2 de cualquier mensaje anterior, del mismo tipo, enviado al mismo interlocutor. De este modo nos aseguramos de que no se recibe un mensaje antes de recibir los correspondientes números aleatorios que actuarán como claves para su correcto descifrado. [SCACAUTH-PRE-4][2]

[2]La definición del protocolo MQTT si lo permite, utilizando para distinguir las respuestas a diferentes mensajes enviados el elemento $Message\ ID$ enviado como parte del mensaje.

- No será posible el envío de un mensaje de suscripción (SUBSCRIBE o UNSUBSCRIBE) por parte de un cliente, hasta que haya recibido la respuesta de cualquier mensaje de suscripción enviado con anterioridad por este mismo cliente. De este modo nos aseguramos que no se recibe un mensaje antes de recibir el correspondiente número aleatorio que actuará como clave para su descifrado. [SCACAUTH-PRE-5][2].
- Se limita el tamaño de los datos a cifrar con criptografía simétrica a 256 bytes, lo que limita el tamaño máximo de los *payload* y *topic* a cifrar a 246 bytes, ya que se acompañan del UID del cliente de 8 bytes y de su longitud en 2 bytes. [SCACAUTH-PRE-6]

4.2. Autenticación

En el estándar MQTT 3.1.1 [s2], la autenticación del cliente por parte del broker se debía realizar en un solo mensaje y su respuesta. El mensaje CONNECT (código 1) y su respuesta CONNACK (código 2). Esto dificultaba poder realizar una autenticación mutua, ya que no había un posible mensaje por parte del cliente al broker, durante el proceso de autenticación, para poder enviar el resultado de la autenticación del broker en el cliente.

El estándar MQTT 5.0 [s1], incluye un sistema de autenticación mejorada que permite un intercambio de mensajes en el proceso de autenticación, para este intercambio se incluye un nuevo tipo de mensaje, el mensaje AUTH (código 15), que se utiliza como respuesta al mensaje CONNECT por parte del broker y para los siguientes intercambios de mensajes, en ambas direcciones, necesarios para la autenticación. El estándar no define qué tipo de autenticación se debe realizar, pero nos da las herramientas necesarias para poder realizar los intercambios de mensajes necesarios para realizar nuestra propia autenticación. Esta autenticación ampliada está definida en la sección 4.12 del estándar.

En la presente sección se define la utilización del estándar MQTT 5.0 para cumplir con las especificaciones dadas para la autenticación mutua MQTT-SCACAUTH definida en la sección 3.3.1 del presente documento, sin incumplir ninguna especificación de dicho estándar.

4.2.1. Mensaje CONNECT $C_x \Rightarrow B$

El intento de autenticación debe iniciarse por parte del cliente, abriendo una conexión vía socket TCP/IP con el broker.

Obligatoriamente el primer mensaje intercambiado con el broker por esta conexión debe ser un mensaje CONNECT enviado por parte del cliente al broker (fig. 4.1[3]) según la declaración normativa del estándar [MQTT-3.1.0-1].

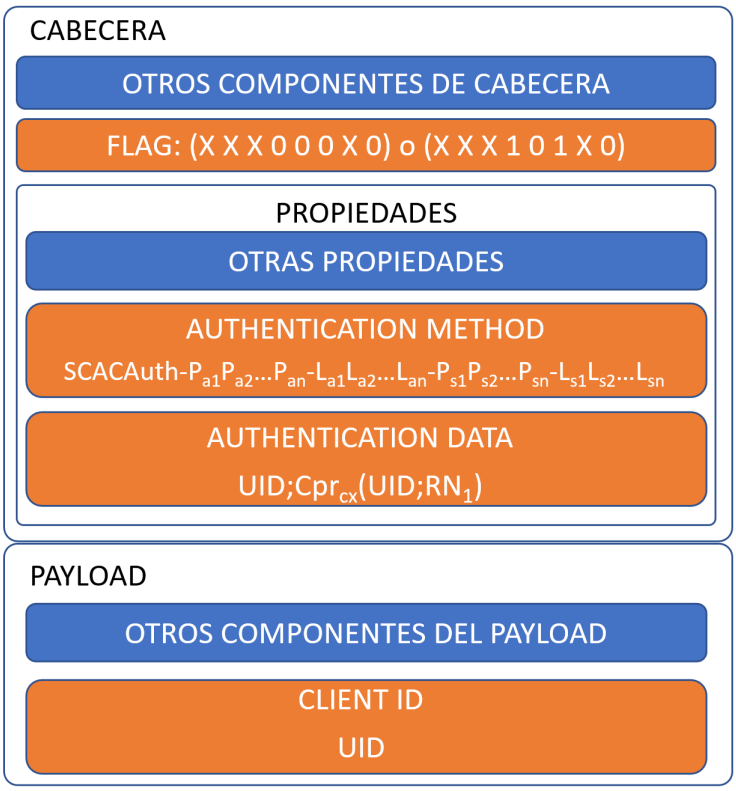


Figura 4.1: Mensaje CONNECT integrando el esquema MQTT-SCACAUTH.

[3]En las figuras que muestren la composición de los mensajes, los datos sobre fondo azul son datos estándar y los datos sobre fondo naranja son datos específicos del esquema propuesto en este documento.

Los componentes específicamente rellenados para cumplir con el esquema de seguridad propuesto se deben rellenar según las especificaciones descritas a continuación.

4.2.1.1. FLAG BYTE

El byte de flag de la cabecera del mensaje CONNECT, byte 8 de esta cabecera, se define como podemos ver en la tabla 4.1 en el estándar.

Tabla 4.1: Flag mensaje CONNECT

Bit	7	6	5	4	3	2	1	0
Flags	User Name	Pass.	Will Retain	Will QoS		Will	Clean Start	Reserv.
byte 8	X	X	X	X		X	X	0

Se rellenan todos los flags como si de una conexión estándar se tratara, excepto los bits 3 y 4 que se deberán rellenar como 0 1 respectivamente en el caso de que se indique la presencia de mensaje automático de desconexión (bit 2 Will Flag). Para asegurar que este se deba enviar con calidad de servicio QoS 2, para cumplir [SCACAUTH-PRE-3] indicada en la sección 4.1 del presente documento.

En el caso de que no se active el Will Flag, los bits del flag Will QoS deberán rellenarse con valor 0.

4.2.1.2. AUTHENTICATION METHOD

Es obligatoria la transmisión de la propiedad *Authentication Method* (código de identificación de propiedad 21) en la cabecera del mensaje CONNECT, definido en la sección 3.1.2.11.9 del estándar. Esta propiedad debe estar codificada como una cadena de caracteres codificados en formato UTF-8.

En esta propiedad, como indica el estándar, se codifica el nombre del método de autenticación con la que el cliente solicita conectarse con el broker. Este nombre se formatea como una expresión regular, separando sus componentes por medio de un guion ('-' U+002D) este carácter no puede ser utilizado en ninguno de sus componentes. Los componentes de la propiedad son los siguientes:

1. $SCACAuth$: Este primer componente es fijo e indica al broker que el método de autenticación utilizado es SCACAUTH (*Smart Card Asymmetric Cryptography Authentication*).

2. $P_{a1}P_{a2}...P_{an}$: En este campo se indica el protocolo criptográfico asimétrico utilizado para la autenticación así como su esquema de relleno ("RSANOPAD", "RSAPKCS1", "ECC", etc.).

3. $L_{a1}L_{a2}...L_{an}$: En este campo se indica la longitud en bits de las claves asimétricas utilizadas para la autenticación ("512", "1024", "2048", etc.).

4. $P_{s1}P_{s2}...P_{sn}$: En este campo se indica el protocolo simétrico de cifrado de bloques utilizado para el cifrado de mensajes de publicación o suscripción, tanto en los envíos de cliente a broker como en los de broker a cliente en el caso de publicación, como los enviados por el cliente en el caso de suscripción ("AES", "DES", "3DES", etc.).

5. $L_{s1}L_{s2}...L_{sn}$: En este campo se indica la longitud en bits de las clave simétrica utilizadas para el cifrado de mensajes de publicación o suscripción tanto en los envíos de cliente a broker como en los de broker a cliente en el caso de

publicación como los enviados por el cliente en el caso de suscripción ("64", "128", "256", etc.).

Por ejemplo, si quisiéramos realizar la autenticación con el algoritmo de criptografía asimétrica RSA [51] con un esquema de relleno según el estándar PKCS1 [s6], con una longitud de clave de 2048 bits y para el cifrado de datos el algoritmo de cifrado por bloques AES [52] con una longitud de clave de 128 bits, el método de autenticación a enviar en la propiedad seria:

"SCACAuth-RSAPKCS1-2048-AES-128"

Para enviar una propiedad en formato de cadena de caracteres UTF-8 debemos codificarla según el formato de codificación descrito en la sección 1.5.4 de la especificación del protocolo (tabla 4.2), codificando en los dos primeros bytes la longitud de la cadena a enviar, codificando el byte más significativo (MSB) en el primer byte y el menos significativo (LSB) en el segundo, formato conocido como *Big Endian*. Después de estos dos bytes y a partir del tercero se codificará la cadena de caracteres en formato UTF-8.

Tabla 4.2: Codificación cadena de caracteres UTF-8 en el protocolo MQTT

Bit	7	6	5	4	3	2	1	0
byte 1	Longitud de cadena MSB							
byte 2	Longitud de cadena MLB							
byte 3 ...	Cadena codificada en caracteres UTF-8							

4.2.1.3. AUTHENTICATION DATA

Es obligatoria la transmisión de la propiedad *Authentication Data* (código de identificación de propiedad 22) en la cabecera del mensaje CONNECT, definido en la sección 3.1.2.11.10 del estándar. Esta propiedad debe estar codificada como un array de datos binarios, este tipo de arrays está definido en la sección 1.5.6 de la especificación del protocolo, donde se define que en sus dos primeros bytes se codifica

la longitud del array en formato *Big Endian* y a partir del tercer byte se transmitirá el propio array.

En esta propiedad deberemos codificar $UID; Cpr_{cx}(UID; RN_1)$ según se describe en la sección 3.3.1.1 del presento documento.

El resto de los componentes de la cabecera del mensaje CONNECT se compone siguiendo las definiciones estándar del protocolo.

4.2.1.4. PAYLOAD

En el *payload* del mensaje CONNECT es obligatoria la inclusión en primer lugar, del $CLIENT\ ID$, en el caso de la autenticación MQTT-SCACAUTH este valor es igual al UID del cliente que esta solicitando la conexión con el broker. Tal y como define la especificación del protocolo, este valor deberá estar codificado como una cadena de caracteres UTF-8 (especificación del protocolo sección 1.5.4).

El resto de componentes del payload del mensaje CONNECT se compone siguiendo las definiciones estándar del protocolo.

4.2.2. Mensaje AUTH $B \Rightarrow C_x$

Una vez recibido el mensaje CONNECT por parte del cliente que quiere autenticarse, el paso que realiza el broker es comprobar el dato de $AUTHENTICATION\ METHOD$ para asegurar que el método de autenticación es MQTT-SCACAUTH así como que las configuraciones implícitas en el valor de este campo son soportadas por el broker.

- Algoritmo de criptografía asimétrica.
- Esquema de relleno.
- Longitud de claves asimétricas.

- Algoritmo de criptografía simétrica.
- Longitud de clave simétrica.

En el caso de que alguna de los configuraciones no sea compatible con el broker, este envía un mensaje CONNACK cuyo valor *Reason Code* sea igual a 0x8C, *Bad authentication method*. Este es un mensaje estándar del protocolo (fig. 4.2), una vez enviado este mensaje el broker cierra la conexión con el cliente.

Figura 4.2: Mensaje CONNACK reason code 0x8C.

Si todos las configuraciones son compatibles con el broker, este debe proceder a la comprobación del *AUTHENTICATION DATA*, según los pasos descritos en la fig. 3.6. Si alguno de los pasos de la comprobación resulta negativo, el broker envía un mensaje CONNACK cuyo valor *Reason Code* es igual a 0x87, *Not Authorized*. Este mensaje es un mensaje estándar del protocolo fig.4.3.

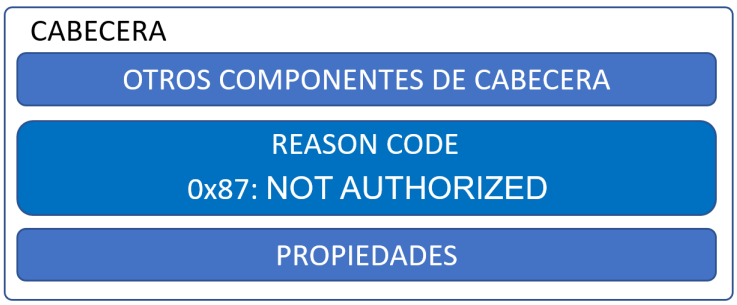


Figura 4.3: Mensaje CONNACK reason code 0x87.

En el caso de que todas las comprobaciones hayan sido correctas, el broker envía un mensaje AUTH, descrito en la sección 3.15 de la especificación, con *Reason Code* igual a 0x18 *Continue Authentication* (fig. 4.4). Con las siguientes propiedades rellenas como se describe en los puntos siguientes.

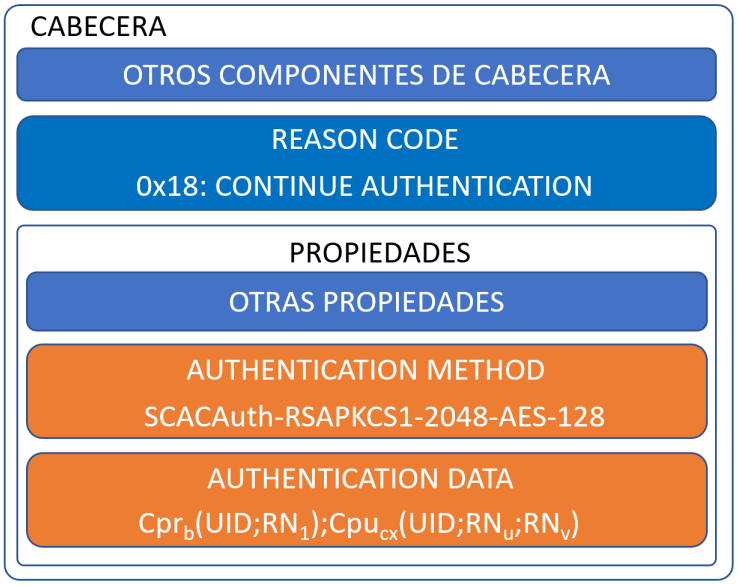


Figura 4.4: Mensaje AUTH broker a cliente.

4.2.2.1. AUTHENTICATION METHOD

En la propiedad *AUTHENTICATION METHOD* del mensaje, el broker incluye el mismo método de autenticación que ha sido enviado en el mensaje CONNECT por parte del cliente, con la misma codificación, como se describe en la sección 3.15.2.2.2 de la especificación del protocolo.

4.2.2.2. AUTHENTICATION DATA

Será obligatorio la transmisión de la propiedad *Authentication Data* (código de identificación de propiedad 22) en la cabecera del mensaje AUTH, definido en

la sección 3.15.2.2.3 del estándar. Esta propiedad deberá estar codificada como un array de datos binarios, este tipo de arrays está definido en la sección 1.5.6 de la especificación del protocolo.

En esta propiedad codificamos $Cpr_b(UID; RN_1); Cpu_{cx}(UID; RN_u; RN_v)$ según se describe en la sección 3.3.1.2 del presente documento.

En este caso, al enviar dos cifrados en el mismo campo, debemos enviar la longitud de cada una de las partes implícita en los datos que enviamos a la Smart Card criptográfica para poder dividir el mensaje en sus dos componentes y poder así descifrarlos de manera correcta. Para facilitar esta división entre las partes, incluiremos dos bytes delante de cada una de las partes, con su longitud en formato MSB-LSB (*Big Endian*).

El resto de los componentes de la cabecera del mensaje AUTH se componen siguiendo las definiciones estándar del protocolo. Este mensaje no envía *payload*.

4.2.3. Mensaje AUTH $C_x \Rightarrow B$

Cuando el cliente recibe el mensaje AUTH, enviado por el broker, el cliente realiza las comprobaciones pertinentes para este paso.

- Comprobar que el $AUTHENTICATION\ METHOD$ enviado en este mensaje es el mismo enviado por él mismo en el mensaje CONNECT.
- Realizar las comprobaciones pertinentes de los datos recibidos en la propiedad $AUTHENTICATION\ DATA$ según se muestra en la fig. 3.7.

Si alguna de las dos comprobaciones no es satisfactoria el cliente deberá enviar un mensaje DISCONNECT al broker con una *Reason Code* igual a 0x80 $Unspecified\ Error$ (fig.4.5). Se debe enviar este error debido a que la *Reason Code* 0x87 $Not\ Authorized$ está reservada únicamente para ser enviada por el broker, tabla 3.10 de la especificación.

En el caso de que la comprobación sea correcta, el cliente envía un mensaje AUTH, descrito en la sección 3.15 de la especificación, con *Reason Code* igual a

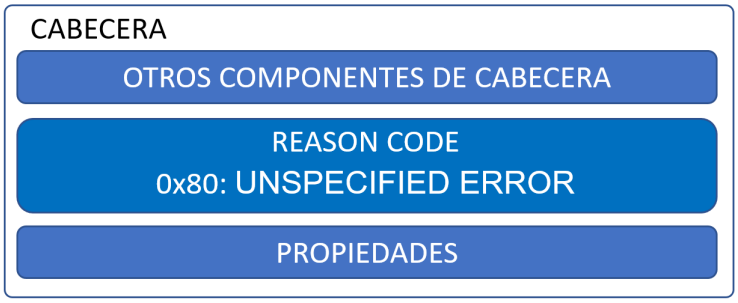


Figura 4.5: Mensaje DISCONNECT reason code 0x80.

0x18 *Continue Authentication* (fig. 4.6). Con las siguientes propiedades rellenas como se describe en los puntos siguientes.

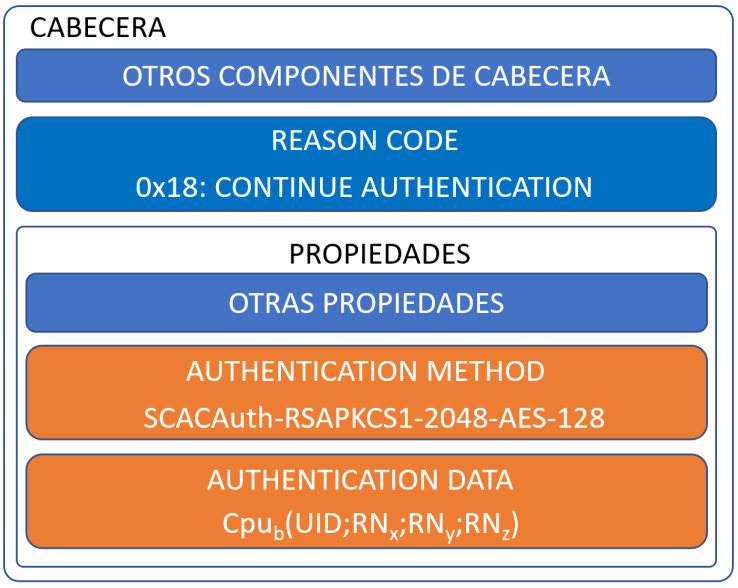


Figura 4.6: Mensaje AUTH cliente a broker.

4.2.3.1. AUTHENTICATION METHOD

En la propiedad del mensaje *AUTHENTICATION METHOD*, el cliente incluye el mismo método de autenticación que ha enviado en el inicio de la autenticación mutua con el mensaje CONNECT, con la misma codificación, como se describe en la sección 3.15.2.2.2 de la especificación del protocolo.

4.2.3.2. AUTHENTICATION DATA

Es obligatoria la transmisión de la propiedad *Authentication Data* (código de identificación de propiedad 22) en la cabecera del mensaje AUTH, definido en la sección 3.15.2.2.3 del estándar. Esta propiedad se codifica como un array de datos binarios. Este tipo de arrays está definido en la sección 1.5.6 de la especificación del protocolo.

En esta propiedad deberemos codificar $Cpu_b(RN_v; RN_x; RN_y; RN_z)$ según se describe en la sección 3.3.1.3 del presente documento.

El resto de los componentes de la cabecera del mensaje AUTH se rellena siguiendo las definiciones estándar del protocolo, este mensaje no envía *payload*.

4.2.4. Mensaje CONNACK $B \Rightarrow C_x$

Una vez el broker reciba el mensaje AUTH, enviado por el cliente que ha iniciado la autenticación mutua, realiza las siguientes comprobaciones:

- Comprobar que el *AUTHENTICATION METHOD* enviado en este mensaje es el mismo enviado por el cliente en el mensaje CONNECT que inicio la autenticación mutua.
- Realizar las comprobaciones pertinentes de los datos recibidos en la propiedad *AUTHENTICATION DATA* según se muestra en la fig. 3.8.

Si ambas comprobaciones son correctas el broker envía un mensaje CONNACK con *Reason Code* 0x00 *Success* (fig. 4.7), dando así por finalizada la autenticación mutua, y habilitando al cliente para que realice las publicaciones y suscripciones que desee.

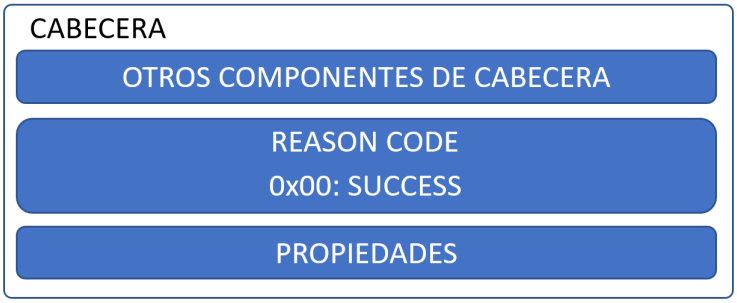


Figura 4.7: Mensaje CONNACK reason code 0x00.

Si alguna de las dos comprobaciones no resulta satisfactoria, el broker envía un mensaje CONNACK con *Reason Code* 0x87 *Not Authorized* fig. 4.3, y corta la conexión con el cliente.

4.3. Publicaciones

Una vez la autenticación mutua haya finalizado de manera satisfactoria, se pueden iniciar las publicaciones tanto desde el cliente al broker como del broker al cliente.

En cumplimiento de [SCACAUTH-PRE-3], todas estas publicaciones deben realizarse con una calidad de servicio QoS 2 (fig. 4.8).

En los procesos de publicación se envían datos útiles *payload* asociados a un *topic*. Ambos campos, en cumplimiento de [SCACAUTH-PRE-7], deben tener una longitud máxima de 246 bytes.

Las publicaciones pueden ser desde un cliente al broker, con el objeto de distribuir los datos publicados a los clientes que se hayan suscrito previamente al *topic* en el que se realiza la publicación. En este caso el cliente actúa como publicador, enviando

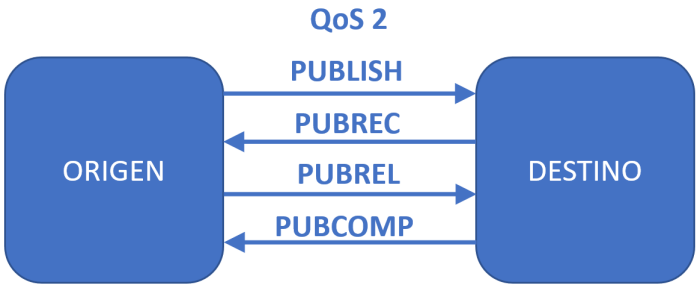


Figura 4.8: Intercambio de mensajes QoS=2.

una publicación al broker para que este la distribuya a los clientes suscritos al *topic* asociado a la publicación.

Para esta distribución se realizan las publicaciones desde el broker a todos aquellos clientes que previamente se hayan suscrito al *topic* al que se asocia la publicación. En este caso, los clientes destino actúan como suscriptores, recibiendo la información publicada por el cliente origen. La información está asociada a un *topic* al que los clientes se han suscrito en el broker. Este intercambio de información se realiza por medio de la intermediación del broker, actuando este como publicador.

Para cada secuencia de mensajes de publicación intercambiados entre el broker y un cliente, se utiliza un par de números aleatorios que únicamente se aplican en un intercambio de mensajes entre un cliente concreto y el broker, cambiando para cada intercambio de un mismo cliente y diferentes para cada uno de los clientes. Estos números aleatorios también serán diferentes dependiendo de si la publicación la realiza el broker al cliente o bien el cliente al broker.

Cuando se realiza un intercambio de mensajes de publicación donde un cliente actúa como publicador, se utilizan los números aleatorios RN_u y RN_v y en el intercambio de mensajes se envían los números aleatorios $RN_{(u+1)}$ y $RN_{(v+1)}$ por parte del broker al cliente, que serán los números aleatorios utilizados en el siguiente intercambio de mensajes de publicación donde este cliente actúe como publicador.

Cuando se realiza un intercambio de mensajes de publicación donde el cliente actúa como suscriptor y es el broker quien publica el mensaje enviándolo al cliente, se utilizan los números aleatorios RN_x y RN_y y en el intercambio de mensajes se envían por parte del cliente los números aleatorios $RN_{(x+1)}$ y $RN_{(y+1)}$ al broker, que

serán los números aleatorios utilizados en el siguiente intercambio de mensajes de publicación desde el broker a este cliente.

En el primer intercambio de publicación en cada dirección entre el broker y un cliente se utilizan, como par de números aleatorios para la cifrado simétrico de los componentes de la publicación, los números aleatorios correspondientes generados e intercambiados en el proceso de autenticación mutua descrito en el punto 4.2 del presente documento.

4.3.1. PUBLISH

En el mensaje PUBLISH enviado se cifra tanto el *topic* como el *payload* enviados con el algoritmo de cifrado por bloques seleccionado en el proceso de autenticación (fig. 4.9).

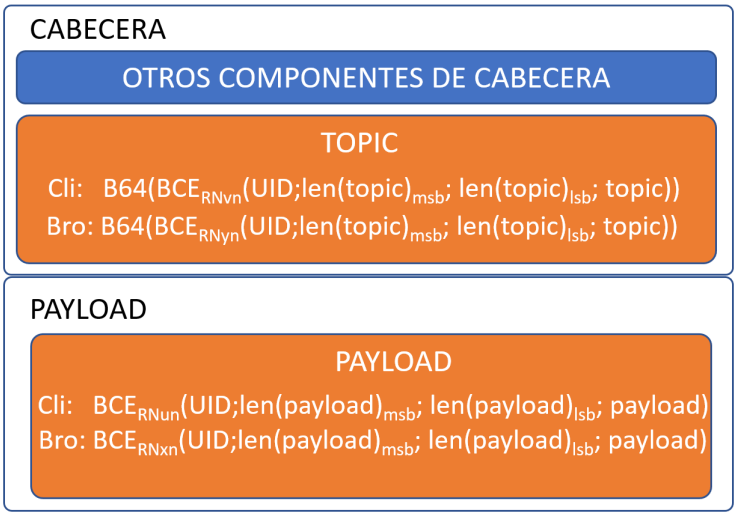


Figura 4.9: Mensaje PUBLISH.

4.3.1.1. TOPIC

Para el cifrado del *topic* se utiliza como clave de cifrado el número aleatorio RN_{yn} en el caso de una publicación del broker al cliente y el número aleatorio RN_{vn} en el caso de una publicación desde un cliente al broker.

El *topic* enviado se compone de tres elementos concatenados y cifrados. La construcción del elemento a cifrar se realiza de la siguiente forma:

1. UID, en primer lugar, se colocará el UID del cliente que envía o recibe el mensaje, de longitud fija 8 bytes según [SCACAUTH-PRE-2].
2. Longitud del *topic* mediante dos bytes en formato *Big Endian* (MSB-LSB).
3. El *topic* que se hubiera enviado en texto plano en una aplicación estándar del protocolo.

Es necesario el envío de la longitud del *topic* implícito en el mensaje debido a que al realizar un cifrado de bloques, el tamaño del elemento a cifrar (y por lo tanto el resultado del descifrado) debe ser divisible por el tamaño de la clave utilizada en bytes. Y así podemos diferenciar, con cualquier algoritmo de cifrado de bloques utilizado, en el resultado del descifrado, el propio *topic* con los posibles elementos de relleno utilizados

No nos será posible enviar directamente el resultado del cifrado como *topic* en el mensaje PUBLISH ya que, y según la sección 3.3.2.1 de la especificación del protocolo, debe ser una cadena de caracteres en codificación UTF-8, y el resultado del cifrado no asegura el cumplimiento de esta especificación.

Para poder enviar el *topic* cifrado codificaremos el resultado del cifrado según la codificación $Base64$ con conjunto de caracteres perteneciente al conjunto de caracteres UTF-8.

Así pues, el *topic* enviado, dependiendo de quién es el publicador del mensaje se compone de la siguiente manera:

$Broker \Rightarrow Cliente_n$: $B64(BCE_{RNy}(UID; MSB_{topic}; LSB_{topic}; topic))$

$Cliente_n \Rightarrow Broker$: $B64(BCE_{RNv}(UID; MSB_{topic}; LSB_{topic}; topic))$

4.3.1.2. PAYLOAD

Para el cifrado del *payload* se utiliza como clave de cifrado el número aleatorio RN_{xn} en el caso de una publicación del broker a un cliente y el número aleatorio RN_{un} en el caso de una publicación desde un cliente al broker.

El *payload* enviado se compone de tres elementos concatenados y cifrados. La construcción del elemento a cifrar se realiza de la siguiente forma:

1. UID, en primer lugar se coloca el UID del cliente al que se envía el mensaje, de longitud fija 8 bytes según [SCACAUTH-PRE-2].
2. Longitud del *payload* mediante dos bytes en formato *Big Endian* (MSB-LSB).
3. El *payload* que se hubiera enviado sin cifrar en una aplicación estándar del protocolo.

Se debe enviar la longitud del *payload* por los mismos motivos descritos anteriormente para el envío del *topic*.

En el caso del *payload* cifrado no es necesaria su transformación para su envío, ya que según el punto 3.3.3 de la especificación del protocolo, el protocolo es agnóstico al formato del contenido del *payload*.

Así pues, el *payload* enviado en el mensaje será:

$Broker \Rightarrow Cliente_n$: $BCE_{RNx}(UID; MSB_{paylaod}; LSB_{paylaod}; paylaod)$

$Cliente_n \Rightarrow Broker$: $BCE_{RNv}(UID; MSB_{paylaod}; LSB_{paylaod}; paylaod)$

El resto de los componentes del mensaje PUBLISH y teniendo en cuenta que debe ser enviado con QoS 2, se rellenan como si de un mensaje estándar se tratara.

4.3.2. PUBREC

Como mensaje de respuesta por parte del receptor del mensaje PUBLISH se debe enviar un mensaje PUBREC con el mismo packet ID recibido en el mensaje PUBLISH, según la sección 3.5 de la especificación del protocolo.

En este mensaje se envían los nuevos números aleatorios $RN_{x(n+1)}$ y $RN_{y(n+1)}$, para el caso de una publicación desde el broker a un cliente y los números aleatorios $RN_{u(n+1)}$ y $RN_{v(n+1)}$ en el caso de una publicación desde un cliente al broker. Estos nuevos números aleatorios serán los que utilizaremos para el siguiente intercambio de mensajes de una publicación entre los mismos interlocutores y en la misma dirección (fig. 4.10).

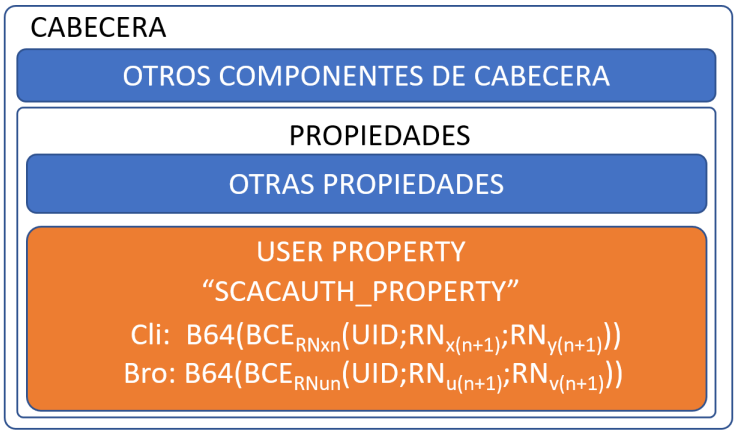


Figura 4.10: Mensaje PUBREC.

Para transmitir estos datos usamos la propiedad *User Property* (sección 3.5.2.2.3 de la especificación del protocolo) del mensaje PUBREC.

Esta propiedad debe ser codificada como un par de cadenas de caracteres UTF-8 (sección 1.5.7 de la especificación del protocolo).

Como nombre de la propiedad, que debe ser codificada en la primera de las dos cadenas UTF-8 del par, enviaremos la cadena de caracteres "SCACAUTH_PROPERTY".

En la segunda cadena del par enviaremos la concatenación de los siguientes ele-

mentos:

1. UID del cliente que actúa como publicador o suscriptor en la publicación.
2. $RN_{x(n+1)}$ para el caso de una publicación del broker a un cliente, o $RN_{u(n+1)}$ para el caso de una publicación de un cliente al broker. Número aleatorio que se utilizará para el cifrado del siguiente *payload* enviado entre los mismos interlocutores y en la misma dirección.
3. $RN_{y(n+1)}$ para el caso de una publicación del broker a un cliente, o $RN_{v(n+1)}$ para el caso de una publicación de un cliente al broker. Número aleatorio que se utilizará para el cifrado del siguiente *topic* enviado por los mismos interlocutores y en la misma dirección.

Estos tres componentes concatenados son cifrados con el algoritmo de cifrado de bloques seleccionado utilizando como clave el número aleatorio RN_{xn}, en el caso de un envío desde el broker a un cliente, o RN_{un} en el caso de un envío desde un cliente hasta el broker, correspondiente a este intercambio de mensajes.

No nos será posible enviar directamente el resultado del cifrado como $User\ Property$ en el mensaje PUBREC ya que la segunda parte del par de cadenas de caracteres UTF-8, deberá ser codificado únicamente con caracteres UTF-8.

Para poder enviar la $User\ Property$ cifrado codificaremos el resultado del cifrado según la codificación $Base64$ con conjunto de caracteres perteneciente al conjunto de caracteres UTF-8.

Así pues, la $User\ Property$ enviada en el mensaje será:

$Broker \Rightarrow Cliente_n\ B64(BCE_{RNx}(UID; RN_{x(n+1)}; RN_{y(n+1)}))$

$Cliente_n \Rightarrow Broker\ B64(BCE_{RNu}(UID; RN_{u(n+1)}; RN_{v(n+1)}))$

4.3.3. PUBREL

Este mensaje que debe ser enviado como respuesta al mensaje PUBREC, se codifica como si de una comunicación estándar MQTT v5.0 se tratara.

4.3.4. PUBCOMP

Este mensaje, que debe ser enviado como respuesta al mensaje PUBREL, se codifica como si de una comunicación estándar MQTT v5.0 se tratara.

4.4. Suscripciones

Para que un cliente reciba las publicaciones relacionadas con un determinado *topic* debe suscribirse a dicho *topic*, para realizar esta suscripción envía un mensaje SUBSCRIBE (código 8) que se responde por parte del broker con un mensaje SUBACK (código 9). Para cancelar una suscripción ya realizada el cliente envía un mensaje UNSUBSCRIBE (código 10) que es respondido por el broker con un mensaje UNSUBACK (código 11).

En el esquema de seguridad MQTT-SCACAUTH los *topics* transmitidos en este mensaje se envían cifrados para evitar que puedan ser desvelados por un atacante.

4.4.1. Crear Suscripción

Para realizar una suscripción, el cliente que desea realizar dicha suscripción envía un mensaje SUBSCRIBE y espera la respuesta por parte del broker con el mensaje SUBACK, según las secciones 3.8 y 3.9 de la especificación. Estos mensajes, adecuados al esquema de seguridad MQTT-SCACAUTH, se describen en los siguientes puntos.

4.4.1.1. SUBSCRIBE

En el mensaje SUBSCRIBE el cliente envía el/los *topic/s* a los que se quiere suscribir. Estos *topics* son cifrados con el número aleatorio RN_{zn}. Para el primer intercambio de mensajes de suscripción o cancelación de suscripción se utiliza como RN_{zn} el número aleatorio RN_z obtenido en el proceso de autenticación mutua.

Todos los *topics* que son enviados en un mensaje SUBSCRIBE concreto serán cifrados con el mismo número aleatorio (fig. 4.11).

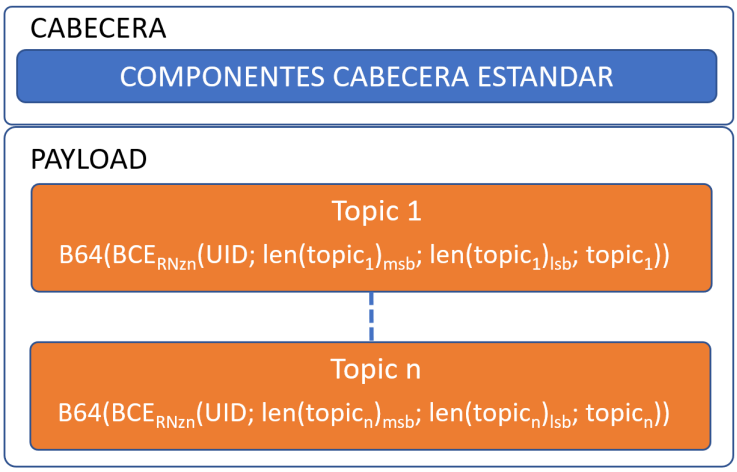


Figura 4.11: Mensaje SUBSCRIBE / UNSUBSCRIBE.

Según la especificación del protocolo, todos los *topics* que se envíen en este mensaje deben ser incluidos en el *payload* del mensaje con el formato mostrado en la tabla 4.3 (figura 3-20 de la especificación del protocolo).

En el campo *topic* (byte 3..N) introducimos el UID del cliente que solicita la suscripción concatenado con el *topic* que queramos enviar precedido de su longitud en formato *big endian*, cifrado con el algoritmo de cifrado por bloques seleccionado con la clave RN_{zn}. Según la especificación (sección 3.8.3) debemos enviar el *topic* como una cadena de caracteres UTF-8, de modo que transformaremos el resultado del cifrado a codificación *Base*64 para poder enviarlo.

Tabla 4.3: Codificación *payload* de los mensajes SUBSCRIBE/UNSUBSCRIBE

Desc.	7	6	5	4	3	2	1	0
Topic								
Byte 1	MSB Longitud topic							
Byte 2	LSB Longitud topic							
Byte 3..N	Topic Filter							
Opciones de la suscripción								
	Reserved		Retain Handling		RAP	NL	QoS	
Byte N+1	0	0	X	X	X	X	1	0

Así cada *topic* enviado en el mensaje será definido por:

$B64(BCE_{RNzn}(UID; MSB_{topic}; LSB_{topic}; topic))$

En las opciones de suscripción enviamos, en todos los mensajes, el QoS máximo de envío de mensajes igual a 2 para cumplir con la premisa inicial [SCACAUTH-PRE-3].

El resto de las opciones se configuran como si de un mensaje estándar se tratara.

La longitud del *topic* (byte 1 y 2) es calculada después de realizar todo el proceso de cifrado y codificación en Base 64.

Tanto la cabecera fija como la cabecera variable de este mensaje se codifica como si de un mensaje estándar del protocolo se tratara.

4.4.1.2. SUBACK

Como mensaje de respuesta por parte del broker a la recepción de un mensaje SUBSCRIBE se envía un mensaje SUBACK con el mismo packet ID recibido en el mensaje SUBSCRIBE, según la sección 3.8.4 de la especificación del protocolo.

En este mensaje se envía, por parte del broker, el mensaje con el nuevo número aleatorios $RN_{z(n+1)}$ que utilizaremos para el siguiente intercambio de mensajes de suscripción o cancelación de suscripción.

Para transmitir estos datos utilizamos la propiedad *User Property* (sección 3.9.2.1.2 de la especificación del protocolo) del mensaje SUBACK.

Esta propiedad debe ser codificada como un par de cadenas de caracteres UTF-8 (sección 1.5.7 de la especificación del protocolo).

Como nombre de la propiedad, que debe ser codificada en la primera de las dos cadenas UTF-8 del par, enviamos la cadena de caracteres "SCACAUTH_PROPERTY".

En la segunda cadena del par enviamos la concatenación de los siguientes elementos:

1. UID del cliente.
2. $RN_{z(n+1)}$ número aleatorio que se utilizará para el cifrado de el/los $topic/s$ en el siguiente mensaje de suscripción o cancelación de suscripción.

Estos dos componentes concatenados son cifrados con el algoritmo de cifrado de bloques seleccionado utilizando como clave el número aleatorio RN_{zn} correspondiente a este intercambio de mensajes (fig. 4.12)..

No nos será posible enviar directamente el resultado del cifrado como *User Property* en el mensaje SUBACK ya que la segunda parte del par de cadenas de caracteres UTF-8, debe ser codificado únicamente con caracteres UTF-8.

Para poder enviar la *User Property* cifrado codificamos el resultado del cifrado según la codificación $Base64$ con conjunto de caracteres perteneciente al conjunto de caracteres UTF-8.

Así pues, la *User Property* enviada en el mensaje será:

$B64(BCE_{RNz}(UID; RN_{z(n+1)}))$

En el *payload* de este mensaje se debe codificar, según la sección 3.9.3 de la especificación, un byte por cada uno de los *topics* enviados en el mensaje SUBSCRIBE con su correspondiente *Reason Code*, siendo únicamente valida como respuesta correcta en nuestro caso 0x02 *Granted QoS* 2.

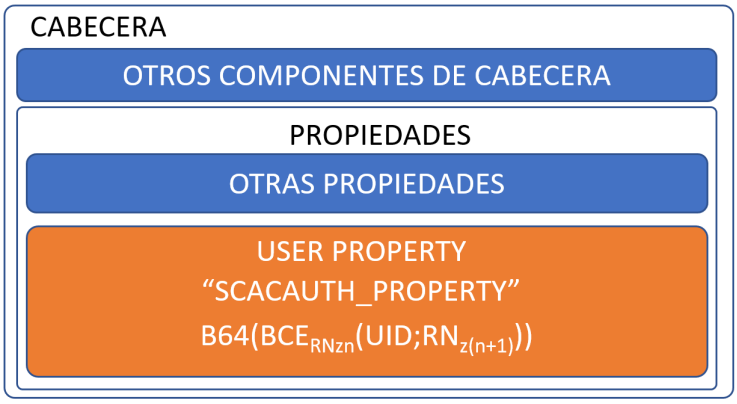


Figura 4.12: Mensaje SUBACK / UNSUBACK.

4.4.2. Cancelar Suscripción

Para realizar la cancelación de una suscripción, el cliente que desea realizar dicha cancelación envía un mensaje UNSUBSCRIBE y debe esperar la respuesta por parte del broker con el mensaje UNSUBACK, según las secciones 3.10 y 3.11 de la especificación. Estos mensajes, adecuados al esquema de seguridad MQTT-SCACAUTH, se describen en los siguientes puntos.

4.4.2.1. UNSUBSCRIBE

En el mensaje UNSUBSCRIBE el cliente envía el/los *topic/s* que a los que quiere dejar de estar suscrito. Estos *topics*, al igual que en una suscripción, son cifrados con el número aleatorio RN_{zn}.

Todos los *topics* enviados un mensaje UNSUBSCRIBE son cifrados con el mismo número aleatorio.

Según la especificación del protocolo, todos los *topics* que se envíen en este mensaje deberán ser incluidos en el *payload* del mensaje con formato de cadena de caracteres UTF-8 uno a continuación del otro.

Los *topics* que se envían en este mensaje para la cancelación de la suscripción son enviados con el mismo cifrado y codificación que de si un mensaje SUBSCRIBE se tratase (punto 4.4.1.1). En este caso no añadiremos detrás de cada uno de los *toipc* el byte de definición de propiedades de la suscripción.

El resto de las opciones se codifican como si de un mensaje estándar se tratara.

Tanto la cabecera fija como la cabecera variable de este mensaje se codifican como si de un mensaje estándar del protocolo se tratara (fig. 4.11).

4.4.2.2. UNSUBACK

Como mensaje de respuesta por parte del broker a la recepción de un mensaje UNSUBSCRIBE se envía un mensaje UNSUBACK con el mismo packet ID recibido en el mensaje UNSUBSCRIBE, según la sección 3.10.4 de la especificación del protocolo.

En este mensaje se envía, por parte del broker, el mensaje con el nuevo número aleatorios $RN_{z(n+1)}$ que utilizaremos para el siguiente intercambio de mensajes de suscripción o cancelación de suscripción (fig. 4.12)..

Para transmitir estos datos utilizamos la propiedad $User\ Property$ (sección 3.11.2.1.2 de la especificación del protocolo), con la misma codificación a la utilizada en el mensaje SUBACK (punto 4.4.1.2).

Así pues la $User\ Property$ enviada en el mensaje será:

$B64(BCE_{RNz}(UID; RN_{z(n+1)}))$

En el *payload* de este mensaje se debe codificar, según la sección 3.11.3 de la especificación, un byte por cada uno de los *topics* enviados en el mensaje UNSUBSCRIBE con su correspondiente *Reason Code*.

Capítulo 5

Integración y Prototipos con MQTT-SCACAUTH

En este capítulo se describen los prototipos realizados para la experimentación del esquema de seguridad definido en esta tesis. Realizándose para ello cuatro prototipos de cliente, para intentar cubrir una amplia gama de dispositivos que puedan utilizar este esquema, un prototipo del broker y un prototipo para el almacén de claves. También se desarrolla la implementación de las funciones criptográficas tanto de los clientes como del broker en una Smart Card criptográfica.

Todo el código desarrollado para estos prototipos está disponible para su libre utilización en el repositorio de código GitHub creado para esta tesis [w1].

5.1. Smart Card Criptográfica

Para la utilización de los algoritmos criptográficos, tanto simétricos como asimétricos en los prototipos desarrollados, se dotará a dichos prototipos de una Smart Card Criptográfica. Cualquier dispositivo con capacidades criptográficas podría servirnos para este fin. En este momento, según el resultado de las investigaciones realizadas, la opción más adecuada es la utilización de Smart Card Criptográficas.

Estos dispositivos son ampliamente utilizados en la actualidad: tarjetas bancarias, de identificación personal, de acceso físico o virtual, para micropagos, como módulos de acceso seguro (SAM por sus siglas en inglés), etc... debido a su bajo coste y altas funcionalidades. Estos dispositivos se ajustan perfectamente a las funcionalidades necesarias para la consecución de los objetivos de esta tesis.

Las Smart Card están bajo la estandarización internacional ISO/IEC 7816 que en sus quince partes define las características físicas, posición y uso de contactos, propiedades eléctricas y de comunicación, así como otros aspectos de este tipo de dispositivos.

Tanto en la parte tres de dicha norma [s7] como en el documento ETSI TS 102 221 [53], del instituto europeo de las telecomunicaciones (ETSI por sus siglas en inglés), se describen los protocolos de comunicación utilizables con este tipo de dispositivos. Después del estudio de estos documentos se han generado las funciones de comunicación entre los controladores que gestionan los prototipos y estos dispositivos.

La comunicación de las Smart Card con los dispositivos externos a ellas se realiza por medio de ocho contactos, que están definidos en la parte dos de la norma ISO/IEC 7816 [s8], distribuidos como podemos ver en la fig. 5.1.

- C1-VCC: Pin de alimentación de la Smart Card.
- C2-RST: Pin para realizar el reset del μprocesador de la Smart Card, necesario después de su alimentación y antes de iniciar la comunicación con esta.
- C3-CLK: Las Smart Card no disponen de un oscilador interno, así que por este pin debe ser proporcionado un tren de pulsos, con una frecuencia adecuada

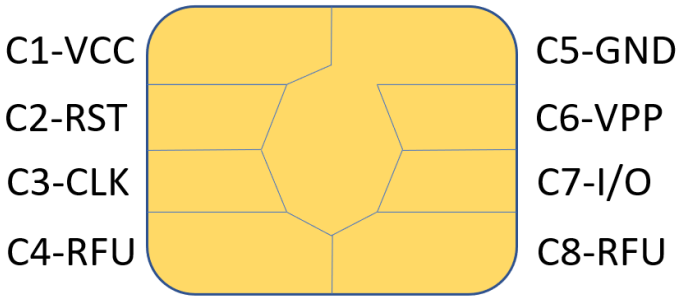


Figura 5.1: Disposición contactos Smart Card.

para el correcto funcionamiento de la tarjeta, usualmente en el rango de 1 Mhz hasta 5 Mhz.

- C4-RFU: este contacto está reservado para un futuro uso, en la actualidad en algunas tarjetas se utiliza para comunicación nativa USB [s9].
- C5-GND: contacto de masa de referencia para las distintas tensiones con las que trabaja la Smart Card.
- C6-VPP: este contacto se utiliza para dar a la Smart Card una tensión auxiliar (generalmente mayor que VCC) para la escritura de memorias EPROM internas, de ser necesario. La mayoría de las tarjetas actuales no utilizan este contacto.
- C7-I/O: Este contacto se comporta como entrada o salida de la comunicación UART que la tarjeta utiliza para recibir las instrucciones y enviar las respuestas. Esta comunicación siempre se configura a 9600 baudios después de un reset y posteriormente se puede configurar mediante comandos adecuados, dependiendo de la frecuencia introducida por el contacto C3-CLK. Usualmente con un oscilador de 4MHz esta comunicación se puede configurar hasta 115200 baudios.
- C8-RFU: este contacto está reservado para un futuro uso, en la actualidad en algunas tarjetas se utiliza para comunicación nativa USB[s9].

Las primeras Smart Card que aparecieron en el mercado no tenían un framework común para el desarrollo de aplicaciones sobre ellas. Los fabricantes de estos

dispositivos incluían un sistema operativo propietario de cada fabricante de Smart Card (SCOS por sus siglas en inglés). Para desarrollar aplicaciones basadas en estas tarjetas era necesario un conocimiento profundo del SCOS que el fabricante implementaba en la Smart Card. Estas tarjetas solo podían ejecutar una única aplicación que se ejecutaba sobre este sistema operativo. El primer intento de estandarizar un sistema operativo para las Smart Card de diferentes fabricantes vino de la mano de un consorcio de compañías denominado MAOSCO, este sistema operativo fue denominado MultOS y presentado en 1997 [w2]. En la actualidad este sistema operativo está presente tanto en Smart Card como en dispositivos IoT embebidos [54].

En 1999 se presentó la versión 2.1 de la máquina virtual Java Card Virtual Machine (JCVM por sus siglas en inglés), específicamente diseñada para su ejecución en Smart Card. No siendo un sistema operativo para su ejecución sobre Smart Card como MultOS, sino una máquina virtual que se ejecuta sobre los sistemas operativos de cada fabricante de Smart Card. Sobre esta máquina virtual se ejecutan las aplicaciones, denominadas applets, que dan las funcionalidades deseadas a la Smart Card. Pudiéndose cargar más de un applet a la vez en una Smart Card.

Esta máquina virtual está actualmente mantenida y actualizada por Oracle como el resto de máquinas virtuales basadas en Java. La última especificación de JCVM es su versión 3.1, cuya última actualización fue presentada en febrero de 2021[s10]. Akram *et al.* [55] realizan un pormenorizado estudio de las distintas etapas por las que han evolucionado los sistemas embebidos en las Smart Card.

Para el desarrollo de los prototipos de esta tesis se han utilizado Smart Card con JCVM. El desarrollo sobre JCVM facilita la utilización de esta aplicación en diferentes tarjetas de diferentes fabricantes. Las últimas versiones de JCVM permiten el manejo de las funciones criptográficas implementadas en las Smart Card.

En concreto se ha optado por una tarjeta criptográfica implementada en base al μprocesador P60D144 de la familia SmartMX2 de la firma NXP [d1]. Este μprocesador ha sido analizado por nCipher Security Limited para el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST por sus siglas en inglés)[56], certificando que implementa defensas sobre los principales ataques contra las Smart Card ([57]-[59]):

- Ataques físicos (tamper detection).

- Ataques de inducción de fallos.
- Ataques de canal lateral (SPA/DPA).
- Análisis temporales.
- Frecuencias fuera de rango.
- Direcciones o instrucciones no permitidas.

En concreto se ha utilizado la Smart Card J3H145-DI, esta tarjeta cumple las funcionalidades necesarias para la realización de estos prototipos, y es de fácil acceso en diferentes plataformas de venta de Smart Card.

Esta Smart Card cuenta con la versión 3.0.4 Classic de la JCVM. La comunicación con las Smart Card se realiza mediante comandos APDU (su estructura se describe en el punto D.3), estando disponible la utilización del protocolo T=0 y T=1 [53] [s7]. En todos los prototipos realizados en esta tesis se utiliza el protocolo T=1 (descrito en punto D.4), siendo este protocolo más seguro ya que dispone de control de errores de comunicación. El protocolo T=1 es el protocolo recomendado por el fabricante de estas Smart Card.

5.2. Applet MQTT-SCACAUTH para Clientes

Para el desarrollo de este applet se ha utilizado el entorno de desarrollo Eclipse Oxygen, con el plug-in de desarrollo de applet JavaCard proporcionado por Oracle para este entorno [w3].

Se ha desarrollado un único applet con todas las funciones necesarias para el cumplimiento de las especificaciones del esquema de seguridad MQTT-SCACAUTH del lado del cliente.

Se define el byte CLA de los comandos APDU aceptados por este applet con el valor 0x80.

En todas los comandos de este applet los bytes P1 y P2 del APDU no se utilizan por lo que deberán codificarse con el valor 0x00.

En la utilización de todas las instrucciones de este applet será necesaria la lectura, por parte del dispositivo que se conecte con el applet, de todos los datos disponibles en la respuesta, por lo que no se incluirá en los comandos APDU el byte L_e.

En todas las funciones realizadas para este prototipo se capturan las excepciones producidas por las funciones criptográficas y de formato de APDU (CryptoException y APDUException), cuya aparición devuelve bytes de estatus de respuesta SW1=0xF0 y SW1=0xF1 respectivamente y SW2 igual al número de excepción capturada.

Para excepciones no capturadas se devuelve en el estatus de respuesta los valores estándar de respuesta [s7], al igual que la devolución de la finalización OK de las funciones será el valor estándar del estatus de respuesta (SW1=0x90, SW2=0x00).

En algunas funciones se añaden estatus de respuesta específicos que son definidos en las descripciones de las funciones correspondientes (punto D.5).

Según la estandarización de los mensajes APDU, se configura este applet para que permita los mensajes extendidos (de más de 256 bytes, como datos de entrada o de respuesta de una instrucción).

La longitud máxima de los mensajes, definida para cada tipo de Smart Card y denominado "Information Field Size for the Card" (IFSC por sus siglas en inglés), se configura como una constante de este applet.

En el caso del prototipo realizado con las tarjetas seleccionadas este valor es el máximo por estándar, 254 bytes. Los comandos y respuestas APDU más largos de este valor son debidamente tratadas por el propio applet, dividiendo los mensajes de respuesta y componiendo las diferentes partes de los comandos recibidos.

Se configuran los parámetros en los que se van a desarrollar los procesos criptográficos en este applet mediante constantes que definen:

- Algoritmo de criptografía asimétrica: para el desarrollo de estos prototipos se utiliza el algoritmo RSA, modelo de relleno PKCS1 y longitud de clave 2048 bits.

- Algoritmo de criptografía simétrica: se utiliza en estos prototipos el algoritmo AES con modelo de encadenamiento CBC y longitud de clave de 128 bits.

Se han realizado 14 funciones que son llamadas mediante 17 códigos de instrucción expuestos para su utilización desde los dispositivos externos que interactúen con la Smart Card (byte INS del comando APDU). Tres instrucciones que devuelven el cifrado simétrico de su parámetro de entrada invocan a la misma función, parametrizando la clave a utilizar, así como dos funciones de descifrado simétrico invocan a otra función también parametrizando la clave a utilizar.

Las instrucciones programadas en este applet son las siguientes:

- Instrucciones para la inicialización de la Smart Card del cliente:
  - CREATE_PAIR(INS 0x20), llama a la función create_pair que genera el par de claves, pública y privada, del cliente. Como respuesta devuelve el módulo y el exponente de la clave pública generada, con el objetivo de poder almacenarla en el almacén de claves del sistema.
  - PUT_PUBLIC_KEY_BROKER (INS 0x22), llama a la función put_public_key_broker que almacena en la memoria de la Smart Card la clave pública del broker.
  - PUT_UID (INS 0x23), llama a la función put_uid que almacena en la memoria de la Smart Card el UID del cliente.
  - INICIALICE_CIPHER (INS 0x2A), llama a la función inicialice_cipher que inicializa todas las funciones criptográficas, tanto simétricas como asimétricas, a utilizar por la Smart Card.
- Instrucciones para la autenticación mutua entre el cliente y el broker:
  - CREATE_AUTH_STEP1 (INS 0x25), llama a la función create_auth_step1 que devuelve los datos a enviar en el primer paso de la autenticación mutua y almacena el número aleatorio RN_1 generado.
  - CHECK_AUTH_STEP2 (INS 0x26), llama a la función check_auth_step2 que comprueba los datos recibidos en el segundo paso de la autenticación mutua y almacena los números aleatorios RN_{un} y RN_{vn} descifrados.

- CREATE_AUTH_STEP3 (INS 0x27), llama a la función create_auth_step3 que devuelve los datos a enviar en el tercer paso de la autenticación mutua y almacena los números aleatorios RN_{xn}, RN_{yn} y RN_{zn} generados.

- Instrucciones para el intercambio de mensajes entre el cliente y el broker:
  - CREATE_PAYLOAD (INS 0x28), CREATE_TOPIC (INS 0x38) y CREATE_TOPIC_SUB (INS 0x39) llaman a la función create_enc que devuelve el cifrado simétrico de la concatenación del UID, almacenado en la Smart Card, con los datos recibidos en la llamada a la instrucción. Utilizando como clave el número aleatorio correspondiente almacenado en la Smart Card (RN_{un}, RN_{vn} y RN_{zn} respectivamente).
  - CREATE_PUBREC (INS 0x31), llama a la función create_pubrec que devuelve los datos a enviar en el siguiente mensaje PUBREC y almacena los nuevos números aleatorios generados y enviados como los nuevos RN_{xn} y RN_{yn}.
  - CHECK_PUBREC (INS 0x29), llama a la función check_pubrec que comprueba los datos recibidos como $USER\ PROERTY$ en un mensaje PUBREC recibido y almacena los números aleatorios descifrados como los nuevos RN_{un} y RN_{vn}
  - READ_PAYLOAD (INS 0x48) y READ_TOPIC (INS 0x58), llama a la función read_enc que devuelve el descifrado de los datos recibidos en la llamada a la instrucción, una vez comprobada su validez, con el algoritmo simétrico configurado. Utilizando como clave el número aleatorio correspondiente almacenado en la Smart Card (RN_{xn} y RN_{yn} respectivamente).
  - CHECK_SUB_UNSUB_ACK (INS 0x2C), llama a la función check_sub_unsub_ack que comprueba los datos recibidos como $USER\ PROERTY$ en un mensaje SUBACK o UNSUBACK recibido y almacena el número aleatorio descifrado como RN_{zn}.

- Instrucciones únicamente con fines de testear el applet durante su desarrollo, estas funciones no serían necesarias en un desarrollo final:
  - GET_PUBLIC_KEY (INS 0x21), llama a la función get_public_key que devuelve el módulo y el exponente de la clave pública almacenada en la Smart Card.
  - DAME_MEMORIA (INS 0x2B), llama a la función dame_memoria que devuelve la memoria disponible (persistente y volátil) en la Smart Card.

Estas funciones son descritas en detalle en el punto D.5 del anexo dedicado a las funciones y estructuras de la Smart Card.

5.2.1. Memoria Utilizada

La tarjeta seleccionada tiene una zona de memoria persistente EEPROM de 144 Kb y una memoria no persistente RAM de 8,125 Kb.

Se ha realizado un reparto de memoria priorizando la utilización de la memoria RAM en todos los objetos que no sea imprescindible su persistencia. Debido a la mucha mayor durabilidad en ciclos de escritura y velocidad de la memoria RAM con respecto a la EEPROM. Una vez cargado el applet en la tarjeta e inicializadas todas las zonas de memoria necesarias para el correcto funcionamiento de todas las funciones del applet, quedan libres 614 bytes en la memoria RAM (7706 bytes utilizados) y 99176 bytes de memoria EEPROM (48280 bytes utilizados).

5.3. Applet MQTT-SCACAUTH para Broker

Como sistema para realizar las operaciones criptográficas necesarias en el broker, en este prototipo se ha utilizado el mismo modelo de Smart Card criptográfica utilizada para el cliente. La mejor opción para este desarrollo hubiera sido un HSM, pero debido a su alto coste no ha sido posible utilizar este tipo de dispositivo para la realización de este prototipo.

En el caso de haber utilizado un HSM, se debería haber seleccionado un dispositivo con capacidad de integrar scripts propios, con el fin de desarrollar el esquema de seguridad MQTT-SCACAUTH en el propio HSM. Después de una búsqueda entre los diferentes fabricantes de HSM comerciales, uno de los posibles candidatos para este desarrollo es el HSM de propósito general de la firma Utimaco [w4], desarrollando los scripts necesarios con su SDK [w5].

Otra posible opción para la ejecución de las funciones criptográficas del broker es

la utilización de sistemas HSM ofrecidos por las diferentes plataformas de despliege de aplicaciones en la nube, AWS CloudHSM [w6], GCP Cloud HSM [w7], etc...

En el prototipo realizado, todas las funciones necesarias para el desarrollo del esquema de seguridad MQTT-SCACAUTH en el broker se han implementado en un applet desarrollado con el mismo entorno de desarrollo utilizado en el desarrollo del applet del cliente.

Las excepciones controladas y constantes utilizadas en este applet son las mismas que las utilizadas en el applet del cliente. Al igual que el applet del cliente, el applet desarrollado para el broker se configura para que permita el intercambio de mensajes extendidos.

Esta Smart Card actua además como almacén de claves, donde se almacenan las diferentes claves públicas de los clientes que se puedan asociar al sistema.

Se define el byte CLA de los comandos APDU aceptados por este applet con el valor 0x80.

El byte P1 se utiliza en la instrucción INIT_CLIENTES para determinar la cantidad de clientes máxima a tratar en la Smart Card, con un máximo de 64 clientes. En el resto de las funciones debe ser codificado como 0x00. El byte P2 no se utiliza en ninguna función de este applet por lo que siempre debe ser codificado como 0x00.

En la utilización de todas las instrucciones de este applet será necesaria la lectura, por parte del dispositivo que se conecte con el applet, de todos los datos disponibles en la respuesta por lo que no se incluirá en los comandos APDU el byte L_e.

Se han realizado 14 funciones que son llamadas mediante 16 códigos de instrucción expuestos para su utilización desde los dispositivos externos que interactúen con la Smart Card (byte INS del comando APDU). Dos instrucciones que devuelven el cifrado simétrico de su parámetro de entrada invocan a la misma función, parametrizando la clave a utilizar, así como tres funciones de descifrado simétrico invocan a otra función también parametrizando la clave a utilizar.

Las instrucciones programadas en este applet son las siguientes:

- Instrucciones para la inicialización de la Smart Card del broker:

  - INIT_CLIENTES (INS 0x4F), llama a la función init_clientes que inicializa la memoria necesaria para gestionar los datos asociados a los clientes.
  - CREATE_PAIR_BROKER(INS 0x20), llama a la función create_pair_broker que genera el par de claves, pública y privada, correspondientes al broker.

- Instrucciones para inicializar un cliente:
  - GET_PUBLIC_KEY (INS 0x21), llama a la función get_public_key que devuelve el módulo y el exponente de la clave pública del broker con el objetivo de cargarla en las Smart Card de los clientes que se asocien a este broker.
  - CREATE_CLIENT (INS 0x40), llama a la función create_client, esta función recibe el módulo y el exponente de la clave pública del cliente a dar de alta en el broker, así como su UID. Estos datos son almacenados en la zona de memoria inicializada para este fin.
  - DELETE_CLIENT (INS 0x60), llama a la función delete_client, esta función recibe el UID del cliente que debe ser borrado del almacén de claves de la Smart Card y lo borra.

- Instrucciones para la autenticación mutua entre el cliente y el broker:
  - CHECK_AUTH_STEP1 (INS 0x41), llama a la función check_auth_step1 que comprueba los datos recibidos en el primer paso de la autenticación mutua y almacena el número aleatorio RN_1 descifrado, asociado al cliente emisor.
  - CREATE_AUTH_STEP2 (INS 0x42), llama a la función create_auth_step2 que devuelve los datos a enviar en el segundo paso de la autenticación mutua y almacena los números aleatorios RN_{un} y RN_{vn} generados, asociados al cliente receptor.
  - CHECK_AUTH_STEP3 (INS 0x43), llama a la función check_auth_step3 que comprueba los datos recibidos en el tercer paso de la autenticación mutua y almacena los números aleatorios RN_{xn}, RN_{yn} y RN_{zn} descifrados, asociados al cliente emisor.

- Instrucciones para el intercambio de mensajes entre el cliente y el broker:

- CREATE_PAYLOAD_BROKER (INS 0x44) y CREATE_TOPIC_BROKER (INS 0x45) llaman a la función create_enc que devuelve el cifrado simétrico de la concatenación del UID, asociado en la Smart Card con el cliente destinatario, con los datos recibidos en la llamada a la instrucción. Utilizando como clave el número aleatorio correspondiente almacenado en la Smart Card (RN_{xn} y RN_{yn}, asociados al cliente destinatario).
- READ_PAYLOAD_BROKER (INS 0x46), READ_TOPIC_BROKER (INS 0x47) y READ_TOPIC_BROKER_SUB (INS 0x48), llama a la función read_enc que devuelve el descifrado de los datos recibidos en la llamada a la instrucción, una vez comprobada su validez, con el algoritmo simétrico configurado. Utilizando como clave el número aleatorio correspondiente almacenado en la Smart Card (RN_{un}, RN_{vn} y RN_{zn}, asociados al cliente emisor).
- CHECK_PUBREC_BROKER (INS 0x49), llama a la función check_pubrec_broker que comprueba los datos recibidos como $USER\ PROERTY$ en un mensaje PUBREC recibido y almacena los números aleatorios descifrados como los nuevos RN_{xn} y RN_{yn}, asociados al cliente emisor.
- CREATE_PUBREC_BROKER (INS 0x4A), llama a la función create_pubrec_broker que devuelve los datos a enviar en el siguiente mensaje PUBREC y almacena los nuevos números aleatorios generados y enviados como los nuevos RN_{un} y RN_{vn} asociados al cliente receptor.
- CREATE_SUB_UNSUB_ACK (INS 0x29), llama a la función create_sub_unsub_ack que devuelve los datos a enviar en el siguiente mensaje SUBACK o UNSUBACK y almacena el nuevo número aleatorio generado y enviado como el nuevo RN_{zn} asociado al cliente receptor.

- Instrucciones únicamente con fines de testear el applet durante su desarrollo, estas funciones no serían necesarias en un desarrollo final:

  - DAME_MEMORIA_BROKER (INS 0x2B), llama a la función dame_memoria_broker que devuelve la memoria disponible (persistente y volátil) en la Smart Card.

Estas funciones son descritas en detalle en el punto D.6 del anexo dedicado a las funciones y estructuras de la Smart Card.

5.3.1. Memoria Utilizada

Una vez cargado el applet en la tarjeta, inicializadas las zonas de memoria necesarias para la ejecución del applet y el almacén de claves con 64 clientes inicializados, quedan libres 99 bytes en la memoria RAM (8221 bytes utilizados) y 48096 bytes de memoria EEPROM (99360 bytes utilizados).

En este applet los números aleatorios asociados a cada cliente deben ser almacenados en memoria no volátil debido a que las limitaciones de memoria RAM de la Smart Card impiden su almacenamiento en dicha memoria.

5.4. Librerías Acceso a los Applets Mediante la API PCSC-Lite

Se ha realizado una librería para cada applet. Estas librerías proporcionan el interfaz de llamada a las instrucciones definidas en cada uno de los applets. Para la comunicación con la Smart Card estas librerías utilizan la API PCSC-Lite [w8].

La API PC/SC fue desarrollada por el grupo de trabajo PC/SC Workgroup [w9] para su ejecución sobre plataformas con sistema operativo Windows.

Gracias al grupo de trabajo "Movimiento para el uso de Smart Card en entorno Linux" (MUSCLE por sus siglas en inglés) se ha desarrollado una implementación de esta API para plataformas sobre sistema operativo Linux. Este desarrollo se denomina PCSC-Lite.

Las librerías desarrolladas se utilizan en todos los prototipos desarrollados sobre una plataforma Linux. En el resto de prototipos (μprocesador y PLC) se desarrolla software específico para esta comunicación.

Tanto para la compilación como para la utilización de estas librerías es necesaria la instalación de la API PCSC Lite en el sistema de destino. A continuación, se muestra los paquetes a instalar en una distribución de Linux basada en Debian. Es

posible también descargar el código de la API en la página oficial del proyecto [w8] y compilarla en nuestro sistema.

```
sudo apt-get install libusb-dev libusb++
sudo apt-get install libccid
sudo apt-get install pcscd
sudo apt-get install libpcsclite1
sudo apt-get install libpcsclite-dev
```

Para la comprobación del funcionamiento del lector es también recomendable, al menos en fase de desarrollo, la instalación del siguiente paquete para poder ejecutar el comando pcsc_scan.

```
sudo apt-get install pcsc-tools
```

Una vez descargados estos paquetes, podemos comprobar el funcionamiento del lector conectado al prototipo, ejecutando la instrucción "pcsc_scan". Con el lector conectado y una tarjeta insertada podremos ver la respuesta al reset de la Smart Card (ATR por sus siglas en inglés) y la descripción de los valores dados (fig. 5.2).

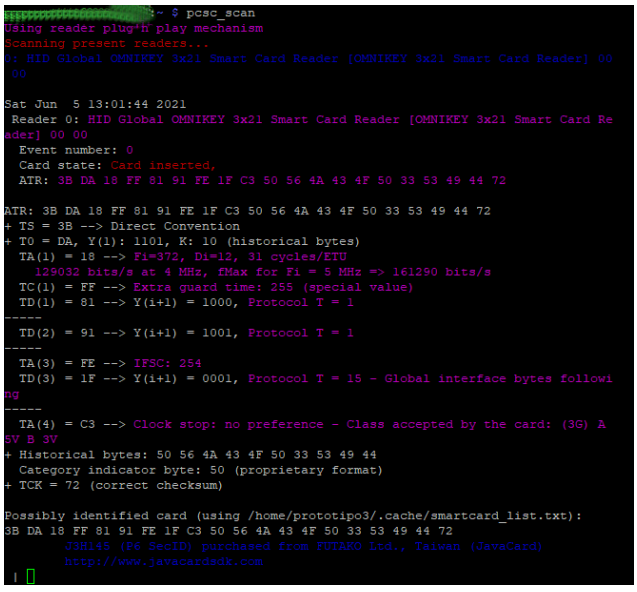


Figura 5.2: PCSC_SCAN lectura ATR.

5.4.1. Librería SCACAuth-Applet-Cliente

El desarrollo de esta librería se ha realizado en lenguaje C. Proporciona funciones para la invocación de todas las instrucciones desarrolladas en el applet MQTT-SCACAUTH para clientes (punto 5.2).

Esta librería la componen la pareja de ficheros "SCACAuth_Applet_Cliente.h" y el fichero "SCACAuth_Applet_Cliente.c" y exporta las siguientes funciones:

- create_pair
- pub_broker_public_key
- get_public_key
- put_uid
- InitCiphers
- create_auth_step1
- check_auth_step2
- create_auth_step3
- create_payload
- create_topic
- create_topic_sub
- read_payload
- read_topic
- check_pubrec
- create_pubrec
- check_sub_unsub_ack

- dame_memoria

Estas funciones invocan a la instrucción correspondiente en la Smart Card y espera a la recepción de la respuesta de cada instrucción. Para cumplir este objetivo todas estas funciones construyen el mensaje APDU correspondiente a enviar a la Smart Card y utilizan la función auxiliar "enviar_APDU" para enviar el APDU construido mediante la función "SCardTransmit" de la API PCSC Lite. Esta función envía el APDU y espera la respuesta de la Smart Card la cual es enviada como respuesta de la función que la ha llamado.

Las funciones "create_payload", "create_topic" y "create_topic_sub" utilizan la función auxiliar "create_enc" pasándole como parámetro en valor INS para la llamada a la instrucción correspondiente en la Smart Card.

Las funciones "read_payload" y "read_topic" utilizan la función auxiliar "read_enc" pasándole como parámetro el valor INS correspondiente para la ejecución de las instrucciones en la Smart Card.

5.4.1.1. Memoria Utilizada

La ejecución del fichero de compilación definido da como resultado los siguientes tres ficheros:

- SCACAuth_Applet_Cliente.o, objeto compilado cuyo tamaño es de 13144 bytes compilado para procesador ARM y 15800 bytes compilado para procesador x64.
- libSCACAuth_Applet_Cliente.so, librería dinámica cuyo tamaño es de 16824 bytes compilado para procesador ARM y 25608 bytes compilado para procesador x64.
- libSCACAuth_Applet_Cliente.a, librería estática cuyo tamaño es de 13782 bytes compilado para procesador ARM y 16438 bytes compilado para procesador x64.

5.4.2. Librería SCACAuth-Applet-Broker

El desarrollo de esta librería se ha realizado en lenguaje C. Proporciona funciones para la invocación de todas las instrucciones desarrolladas en el applet MQTT-SCACAUTH para broker (punto 5.3).

Esta librería la componen la pareja de ficheros "SCACAuth_Applet_Broker.h" y el fichero "SCACAuth_Applet_Broker.c" y exporta la siguientes funciones:

- init_clientes
- create_pair_broker
- get_public_key
- create_client
- delete_client
- check_auth_step1
- create_auth_step2
- check_auth_step3
- create_payload_broker
- create_topic_broker
- read_payload_broker
- read_topic_broker
- read_topic_broker_sub
- check_pubrec_broker
- check_pubrec
- create_sub_unsub_ack

- dame_memoria_broker

En esta librería, al igual que en la de llamada a las funciones del applet del cliente, también se ha generado una función auxiliar "enviar_APDU" llamada por todas las funciones para el envío de los comandos APDU y la recepción de su respuesta mediante la función "SCardTransmit" de la API PCSC Lite. En esta función se colocan unos bloqueos por medio de la función "pthread_mutex_lock" de la librería "phthread" para impedir que se ejecute una instrucción en el applet de la tarjeta antes de la finalización de una instrucción previamente enviada. En el broker puede darse el caso de un intento concurrente de acceso a la Smart Card, cuando se estén realizando intercambios de mensajes con varios clientes distintos al mismo tiempo.

De la misma manera que la librería para el applet del cliente se ha creado una función auxiliar "create_enc" que utilizan las funciones "create_payload_boker" y "create_topic_broker". Así como otra función auxiliar "read_enc" que es utilizada por las funciones "read_payload_broker", "read_topic_broker" y "read_topic_broker_sub".

5.4.2.1. Memoria Utilizada

La ejecución del fichero de compilación definido da como resultado los siguientes tres ficheros:

- SCACAuth_Applet_Broker.o, objeto compilado cuyo tamaño es de 16216 bytes compilado para procesador ARM y 18792 bytes compilado para procesador x64.
- libSCACAuth_Applet_Broker.so, librería dinámica cuyo tamaño es de 21324 bytes compilado para procesador ARM y 26008 bytes compilado para procesador x64.
- libSCACAuth_Applet_Broker.a, librería estática cuyo tamaño es de 16990 bytes compilado para procesador ARM y 19566 bytes compilado para procesador x64.

5.5. Librería SCACAuth-Cliente

Esta librería utiliza la librería estática "SCACAuth_Applet_Cliente.a" para ejecutar las instrucciones de la Smart Card.

En el código de esta librería se ha creado una pareja de ficheros ".c" y ".h" para cada uno de los tipos de mensaje del protocolo MQTT. En estos ficheros se desarrollan las funciones de envío y recepción de estos mensajes utilizados por un cliente y las funciones auxiliares necesarias para el tratamiento de estos mensajes.

- "message_auth.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_auth" y "send_auth" para su uso interno en la librería.
- "message_connack.c/.h": Este mensaje solo puede ser recibido por el cliente. Se crea la función "receive_connack" para su uso interno en la librería.
- "message_connect.c/.h": Este mensaje solamente puede ser enviado por el cliente. Se crea la función "send_connect" para su uso interno en la librería.
- "message_disconnect.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_disconnect" y "send_disconnect" para su uso interno en la librería.
- "message_pingreq.c/.h": Este mensaje solo es enviado por el cliente, antes de cumplir el tiempo configurado como keepalive. Se crea la función "send_pingreq" para su uso interno en la librería.
- "message_pubcomp.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_pubcomp" y "send_pubcomp" para su uso interno en la librería.
- "message_publish.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_publish" y "send_publish" para su uso interno en la librería.
- "message_pubrec.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_pubrec" y "send_pubrec" para su uso interno en la librería.

- "message_pubrel.c/.h": Este mensaje puede ser enviado y recibido por el cliente. Se crean las funciones "receive_pubrel" y "send_pubrel" para su uso interno en la librería.
- "message_subscribe.c/.h": Este mensaje solo puede ser enviado por el cliente. Se crea la función "send_subscribe" para su uso interno en la librería.
- "message_suback.c/.h": Este mensaje solo puede ser recibido por el cliente. Se crea la función "receive_suback" para su uso interno en la librería.
- "message_unsubscribe.c/.h": Este mensaje solo puede ser enviado por el cliente. Se crea la función "send_unsubscribe" para su uso interno en la librería.
- "message_unsuback.c/.h": Este mensaje solo puede ser recibido por el cliente. Se crea la función "receive_unsuback" para su uso interno en la librería.

Esta librería exporta una serie de funciones para su utilización en el programa cliente que utilice esta librería. Estas funciones realizan el intercambio completo de mensajes para las diferentes funciones utilizadas en un intercambio de datos con el protocolo MQTT bajo el esquema de seguridad MQTT-SCACAUTH, estas funciones se ejecutan en un bucle de envío y recepción de mensajes.

En la pareja de ficheros "conectar_SCard.c/.h" se crea la función para el uso externo a la librería "conectar_SCard" que realiza la conexión con la Smart Card. Y en la pareja de ficheros "connect_SCACAuth.c/.h" se crea la función para el uso externo a la librería, "connect_SCACAuth" que realiza todo el proceso de autenticación mutua entre el cliente y el broker.

El resto de las funciones exportadas para su uso externo a la librería se han desarrollado en la pareja de ficheros "loop.c/.h".

Para inicializar el bucle donde se controlan estas funciones se crea la función "loop_start". En esta función se inicializa el hilo de recepción de datos de la librería, ejecutando en un hilo separado la función "loop_recepcion", que se encarga de recibir los datos desde el broker.

Una vez llamada a la función de inicialización se debe llamar a una de las dos siguientes funciones:

- "loop_forever": esta función entra en un bucle de ejecución infinito, el hilo de llamada no podrá ejecutar ninguna otra funcionalidad, únicamente recibirá los eventos generados por la librería.
- "loop_once": esta función realiza una ejecución del bucle de recepción y envío, esta función esta preparada para ser ejecutada dentro de un bucle del programa principal.

A ambas funciones se les pasan seis punteros a funciones que actúan como receptores de eventos dentro del programa principal, son ejecutadas ante los siguientes eventos:

- "on_receive": esta función se ejecuta en el momento de finalizar correctamente un proceso de recepción de un mensaje PUBLISH, por parte del cliente, enviado por el broker y todo el proceso siguiente de mensajes en el esquema MQTT-SCACAUTH con un resultado correcto.
- "on_subscribe": esta función se ejecuta en el momento de finalizar correctamente un proceso de envío de un mensaje SUBSCRIBE por parte del cliente al broker y todo el proceso siguiente de mensajes en el esquema MQTT-SCACAUTH con resultado correcto.
- "on_unsubscribe": esta función se ejecuta en el momento de finalizar correctamente un proceso de envío de un mensaje UNSUBSCRIBE por parte del cliente al broker y todo el proceso siguiente de mensajes en el esquema MQTT-SCACAUTH con resultado correcto.
- "on_publish": esta función se ejecuta en el momento de finalizar correctamente un proceso de envío de un mensaje PUBLISH por parte del cliente al broker y todo el proceso siguiente de mensajes en el esquema MQTT-SCACAUTH con resultado correcto.
- "on_error": esta función se ejecuta ante cualquier defecto generado en cualquier intercambio de mensajes iniciado bien por el broker o bien por el cliente.
- "on_reconnect": ante cualquier problema de comunicación en el que sea necesario reconectar con el servidor y volver a ejecutar el proceso de autenticación mutua se ejecuta una reconexión de forma automática por parte de la librería,

si este proceso de reconexión se realiza de manera correcta, se ejecuta esta función en el programa del cliente.

Para finalizar la ejecución del bucle de intercambio y la finalización de la conexión se exporta para su uso por parte del programa del cliente la función "loop_exit", que desconecta la conexión entre el cliente y el broker y finaliza la ejecución del bucle.

Para realizar las funcionalidades iniciadas por parte del cliente se exporta en esta librería las siguientes tres funciones. En su lanzamiento crean un nuevo hilo para su ejecución, el intercambio de mensajes para su finalización se realiza por medio del bucle de ejecución y el resultado de su finalización se informa al programa del cliente por medio de las funciones de evento configuradas en el bucle.

- "subscribe_async": Esta función inicia el proceso de suscripción a un *topic* por parte del cliente. Si se finaliza correctamente la suscripción se informa al programa de usuario mediante la ejecución de la función configurada como "on_subscribe", si finaliza incorrectamente el intercambio de mensajes se informa al programa mediante la ejecución de la función configurada como "on_error".
- "unsubscribe_async": Esta función inicia el proceso de finalizar la suscripción a un *topic* por parte del cliente. Si se finaliza correctamente el proceso UNSUBSCRIBE se informa al programa de usuario mediante la ejecución de la función configurada como "on_unsubscribe", si finaliza incorrectamente el intercambio de mensajes se informa al programa mediante la ejecución de la función configurada como "on_error".
- "publish_async": Esta función inicia el proceso de publicación de un *payload* asociado a un *topic*. Si se finaliza correctamente la publicación se informa al programa de usuario mediante la ejecución de la función configurada como "on_publish", si finaliza incorrectamente el intercambio de mensajes se informa al programa mediante la ejecución de la función configurada como "on_error".

Con el objetivo de comprobar la interoperabilidad de esta librería en diferentes lenguajes de programación, se ha generado un módulo Python a partir de esta librería. Para la generación de este módulo se ha creado un warpper para las funciones a exportar en este módulo en el fichero "scacauth_client_warpper_py.c". Con este fin

se ha utilizado la librería para construir este tipo de módulos cuyo fichero de cabecera es "Python.h".

Con la ayuda de esta librería se ha generado una librería dinámica "SCACAuth_Cliente_Python.so", con el formato adecuado para poder ser importada en programas desarrollados con el lenguaje de programación Python. Esta librería se compila incluyendo en su compilación el objeto "SCACAuth_Applet_Cliente.o" para empaquetar todo lo necesario en un solo módulo Python.

5.5.1. Memoria Utilizada

La ejecución del fichero de compilación definido da como resultado los siguientes dos ficheros:

- SCACAuth_Cliente.o, objeto compilado cuyo tamaño es de 58652 bytes compilado para procesador ARM y 84600 bytes compilado para procesador x64.
- libSCACAuth_Cliente.so, librería dinámica cuyo tamaño es de 67000 bytes compilado para procesador ARM y 80584 bytes compilado para procesador x64.
- libSCACAuth_Cliente.a, librería estática cuyo tamaño es de 61018 bytes compilado para procesador ARM y 86966 bytes compilado para procesador x64.
- SCACAuth_Cliente_Python.so, librería dinámica cuyo tamaño es de 81948 bytes compilado para procesador ARM y 99112 bytes compilado para procesador x64.

5.6. Aplicación para Generar Smart Card de Broker y Clientes

Con el fin de inicializar las Smart Card implicadas en las comunicaciones se ha desarrollado una aplicación, en lenguaje C. Esta aplicación genera las claves de criptografía asimétrica de las Smart Card tanto del broker como de los clientes y rellena el almacén de claves de la Smart Card del broker con las claves generadas para los clientes.

Para el completo y correcto uso de las funcionalidades de esta aplicación se debe ejecutar en un dispositivo basado el Linux y con dos lectores de Smart Card conectados, uno para el trabajo con la Smart Card del broker y el otro para el trabajo con las diferentes Smart Card de los clientes a inicializar y dar de alta.

Para el intercambio de mensajes con las Smart Card, esta aplicación hace uso de las librerías estáticas "SCACAuth_Applet_Broker.a" y "SCACauth_Applet_Cliente.a".

5.6.1. Funciones

La aplicación realiza diferentes funciones dependiendo de los parámetros de entrada que se le pasen en su llamada. Antes de cada ejecución la aplicación consulta al usuario que lector se debe utilizar para cada Smart Card (fig. 5.3).

```
Seleccione el lector para el cliente:

[0] ACS ACR38U-CCID 00 00
[1] HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21 Smart Card Reader] 01 00
1

Seleccione el lector para el broker:

[0] ACS ACR38U-CCID 00 00
[1] HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21 Smart Card Reader] 01 00
```

Figura 5.3: Selección Lectores

Es posible ejecutar múltiples funciones en una sola ejecución pasando los parámetros de cada función en el orden que se deseen ejecutar, la aplicación ejecuta las funciones en el orden de recepción de los parámetros.

Una vez finalizada la ejecución de cada función se informa al usuario del resultado de la misma.

Inicialización de memoria de clientes en la Smart Card del broker

Para ejecutar esta función se deben pasar dos parámetros a la aplicación "-I" para designar esta función y un segundo parámetro con el número de clientes máximo a inicializar en la tarjeta. Esta función ejecuta la instrucción INIT_CLIENTES para inicializar las áreas de memoria del almacén de claves en la Smart Card del broker.

Para la ejecución de esta función únicamente es necesario un lector.

Inicialización par de claves asimétricas en broker

Para ejecutar esta función se debe pasar un único parámetro "-B". Ejecuta la instrucción CREATE_PAIR_BROKER para generar el par de claves de criptografía asimétrica en la Smart Card del broker.

Para la ejecución de esta función únicamente es necesario un lector.

Inicialización cliente

Esta función se ejecuta pasando como parámetros a la aplicación "-C" para indicar esta instrucción y como un segundo parámetro el UID del cliente a inicializar.

Ejecuta las siguientes instrucciones:

- PUT_UID (cliente).
- CREATE_PAIR (cliente). Como resultado obtiene el módulo y el exponente de la clave pública del cliente.
- CREATE_CLIENTE (broker). Llamada con el resultado de la instrucción anterior.
- GET_BROKER_PULBIC_KEY (broker). Como resultado se obtiene el módulo y el exponente de la clave pública del broker.

- PUT_BROKER_PUBLIC_KEY (cliente). Llamada con el resultado de la instrucción anterior.
- INIT_CIPHERS (cliente).

Estas funciones inicializan la Smart Card del cliente y almacenan en la Smart Card del broker la clave pública del cliente.

Para la ejecución de esta función son necesarios dos lectores.

Borrar cliente

Para ejecutar esta función se deben pasar dos parámetros, "-D" para indicar esta función y como segundo parámetro el UID del cliente que queremos borrar. Ejecuta la instrucción DELETE_CLIENT para borrar el cliente del almacén de claves de la Smart Card del broker.

Para la ejecución de esta función únicamente es necesario un lector.

Probar cliente

Para ejecutar esta función se deben pasar dos parámetros, "-P" para indicar esta función y como segundo parámetro el UID del cliente que queremos probar.

Esta función realiza una serie de pruebas de las instrucciones de la Smart Card del cliente contra la Smart Card del broker. Probando todas las funcionalidades de ambas tarjetas.

Para la ejecución de esta función son necesarios dos lectores.

5.7. Prototipo Basado en μprocesador

5.7.1. Hardware

Para este prototipo se ha utilizado una placa comercial Sistem On Chip (SoC por sus siglas en inglés), MKR1000. Este dispositivo integra en una sola tarjeta un μprocesador ATSAMD21G18A y un interfaz WiFi WinC1500 802.11 b/g/n a 2,4Ghz. Así como los diferentes elementos necesarios para su integración y desarrollo (fig. 5.4).

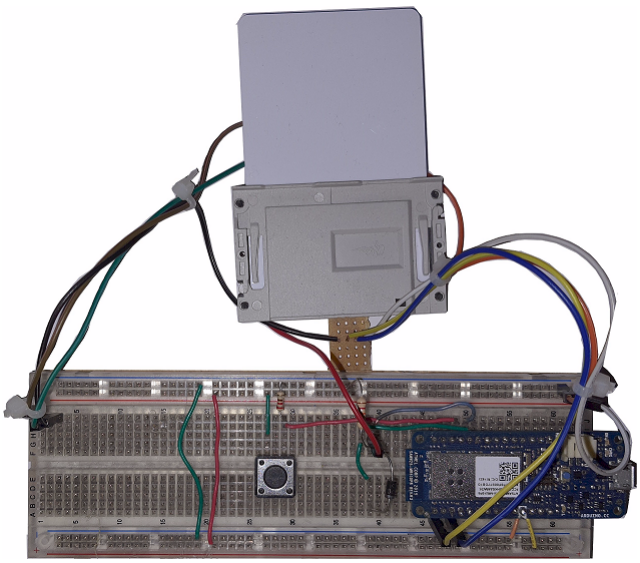


Figura 5.4: Prototipo μprocesador (esquema en fig. C.1).

Para su fácil prototipado este SoC integra el cargador para plataformas de código abierto Arduino, facilitando así la realización del prototipo y su sencilla traslación a cualquier SoC basado en esta plataforma de código abierto.

Este dispositivo ha sido utilizado en diversas investigaciones relacionadas con las comunicaciones en el ámbito del IoT [60]-[62].

En este prototipo se ha realizado la interconexión entre la Smart Card y el μprocesador a través de su puerto UART. Para realizar la adaptación del puerto UART del μprocesador, a dos hilos, y el puerto UART (C7-I/O) de la Smart Card a un hilo se ha colocado un diodo y una resistencia de pull-up (esquema en C.1). Por otro lado, para enviar la señal de reloj a la Smart Card (C3-CLK) se ha generado una señal de reloj desde el propio μprocesador a través de una de sus salidas PWM.

5.7.2. Software

El software desarrollado se ha realizado con el IDE "Visual Studio 2019 Community", IDE de libre distribución de la firma Microsoft y el plug-in "Aruino IDE for Visual Studio" desarrollado para la realización de software destinado a placas de desarrollo Arduino en este IDE.

La comunicación mediante la tarjeta WiFi integrada se realiza con la librería "WiFi101" de libre distribución adecuada para el control de la tarjeta WiFi integrada en el SoC.

Todo el código ha sido desarrollado en el lenguaje de programación C++, pudiéndose trasladar la funcionalidad de este prototipo a cualquier dispositivo que tenga un compilador adecuado de este lenguaje.

En este prototipo se ha integrado el software necesario para la autenticación del cliente con el broker mediante el sistema de autenticación MQTT-SCACAUTH descrito en esta tesis. Así como el envío de mensajes ante el cambio de nivel en una de las entradas digitales que proporciona el SoC.

Cuando la entrada configurada para el envío de mensajes cambia a nivel alto, se envía un mensaje al *topic* "micro/pulsador_micro" con el *payload* "ha sido pulsado". Cuando la misma señal cambia a nivel bajo, se envía un mensaje al mismo *topic* con el *payload* "ha sido liberado". Realizando todo el intercambio de mensajes descrito en esta tesis.

El fichero principal del programa, "mqttSCACAuth.ino", contiene las dos funciones estándar para las aplicaciones desarrolladas en esta plataforma.

La función *Setup* únicamente se ejecuta una vez cuando el dispositivo se inicia.

En esta función se realizan las tareas de configuración del dispositivo, configurando las I/O utilizadas para la comunicación UART con la Smart Card (RX/TX), una señal que servirá como señal RTS de la comunicación y la I/O utilizada como entrada para el control de los mensajes a enviar. También se configura la WiFi del SoC con la SSID y las claves utilizadas en la comunicación.

Se configura la señal de reloj generada por el μprocesador a 3.67 Mhz para la recepción del ATR de la Smart Card a 9600 baudios, que es la velocidad de comunicación a la que la Smart Card envía el ATR.

Una vez realizadas estas configuraciones se envía un reset a la Smart Card mediante el pin configurado como señal RTS. Una vez recibido el ATR se envía un mensaje de selección de protocolo y parámetros (PPS por sus siglas en inglés) (punto D.2) a la Smart Card para cambiar el protocolo de intercambio de APDUs con la Smart Card a T=1 así como la velocidad de comunicación a 161290 baudios, velocidad máxima de comunicación para la frecuencia máxima de reloj admitida por la Smart Card según su ATR (punto D.1). Una vez recibida la respuesta a este mensaje se cambia la frecuencia de la señal de reloj generada a 5 Mhz.

Al finalizar el proceso de configuración con el mensaje PPS, se envía el primer mensaje en protocolo T=1. Este primer mensaje se trata de un mensaje S-Block de configuración de la longitud máxima del campo INF. Se configura al máximo de 254 bytes. Por defecto, las Smart Card se inicializan con este valor a 32 bytes.

Por último, en la función *Setup* se realizan todas las reservas de memoria de las variables que se van a utilizar durante el funcionamiento del prototipo.

Una vez finalizada la función *Setup* se comenzará a ejecutar en un bucle infinito la función *loop* del fichero principal.

En esta función primero se comprueba la conexión correcta a la WiFi. Si la conexión es correcta y no se ha autenticado la comunicación con el broker se lanza el proceso de autenticación con este.

Una vez el proceso de autenticación ha sido finalizado correctamente, se comprueba si en este ciclo ha habido un cambio en el nivel de la señal de la I/O configurada. En el caso de haber detectado un cambio de nivel se realiza el proceso de envío del

mensaje correspondiente.

En el caso de perder la conexión con el broker, en el siguiente ciclo de la función *loop* se volverá a intentar la conexión y autenticación del cliente en el broker.

Como se puede observar en la fig. 5.5, se ha estructurado el código generado para su fácil reutilización en futuros prototipos o proyectos.

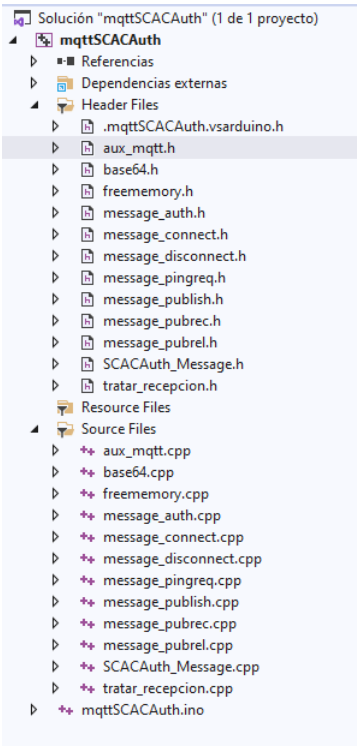


Figura 5.5: Ficheros prototipo μprocesador.

Cada uno de los tipos de mensajes intercambiados con el broker para el cumplimiento de los objetivos de este prototipo ha sido encapsulado en una pareja de ficheros (.h y .cpp).

message_connect.h / .cpp

En estos ficheros se desarrolla la función "send_connect". Se encarga de construir y enviar el mensaje CONNECT al broker, con los datos de autenticación obtenidos

de la Smart Card mediante la instrucción CREATE_AUTH_STEP1, que se le pasan como parámetro a esta función.

message_auth.h / .cpp

En estos ficheros se desarrollan dos funciones "rcv_auth" y "send_auth".

La función "rcv_auth" encargada de comprobar el mensaje AUTH enviado por el broker al cliente y recuperar los datos de autenticación en un parámetro pasado por referencia. Este parámetro posteriormente será comprobado mediante la instrucción CHECK_AUTH_STEP2 de la Smart Card.

Por otro lado, define la función "send_auth" encargada de construir y enviar el mensaje AUTH del cliente al broker, con los datos de autenticación creados con la instrucción CREATE_AUTH_STEP3 de la Smart Card y que se le pasan como parámetro a esta función.

message_pingreq.h / .cpp

En estos ficheros se desarrolla la función "send_pingreq", que se encarga de construir y enviar el mensaje PINGREQ al broker.

message_publish.h / .cpp

Estos ficheros desarrollan una única función, "send_publish", encargada de construir y enviar el mensaje PUBLISH que el cliente envía al broker. Para la construcción de este mensaje se deben pasar como parámetro a esta función el *topic* y el *payload* a enviar. Estos parámetros son construidos mediante las instrucciones CREATE_TOPIC y CREATE_PAYLOAD de la Smart Card.

message_pubrec.h / .cpp

Contienen una única función "rcv_pubrec" encargada de comprobar que la recepción del mensaje PUBREC enviado desde el broker al cliente es correcta. Extrae la propiedad de usuario que será recuperada por la función en un parámetro pasado por referencia y será posteriormente comprobada con la instrucción CHECK_PUBREC de la Smart Card.

message_pubrel.h / .cpp

Estos ficheros contienen la definición de la función "send_pubrel", que se encarga de construir y enviar el mensaje PUBREL al broker.

Además de los ficheros donde se encuentran las funciones relacionadas con la mensajería MQTT, se desarrollan las funciones auxiliares necesarias para el correcto funcionamiento del prototipo en los ficheros descritos a continuación.

aux_mqtt.h / .cpp

En estos ficheros se definen las funciones de codificación y descodificación de datos en los formatos utilizados en los mensajes MQTT, enteros en 2 bytes, 4 bytes, en longitud variable, utf8, etc...

base64.h / .cpp

En estos ficheros se definen las funciones de codificación en formato Base64 ("base64_encode") de array de bytes y descodificación ("base64_decode") en array de bytes.

Esta codificación es utilizada para el intercambio de los cifrados correspondientes en aquellos campos de los mensajes MQTT que solo admiten cadenas de caracteres codificados en formato UTF-8.

freememory.h / .cpp

En estos ficheros se define la función "freeMemory", utilizada únicamente con fines de análisis del código generado para poder saber la cantidad de memoria RAM utilizada por el programa cargado en el dispositivo.

tratar_recepcion.h / .cpp

Estos ficheros contienen la función "tratar_recepcion" que se encarga de la lectura de los mensajes recibidos por el cliente. Lee la cabecera fija del mensaje, obteniendo así el tipo de mensaje recibido y la longitud que sigue a esta cabecera y posteriormente lee el resto del mensaje recibido.

Esta función devuelve en sendos parámetros por referencia el tipo de mensaje leído, así como el mensaje leído a partir de la cabecera fija.

SCACAuth_Message.h / .cpp

Estos ficheros contienen el desarrollo de todas las funciones que se utilizan para el envío de instrucciones a la Smart Card y su posterior recepción de respuesta:

- "create_auth_step1"
- "check_auth_step2"
- "create_auth_step3"
- "create_payload"
- "create_topic"
- "check_pubrec"

En estas funciones se construye el APDU de cada mensaje y utilizando la función auxiliar, "SCTransaccionT1", también definida en este fichero, se envía el mensaje y se espera su respuesta.

La función "SCTransaccionT1" se encarga del intercambio de mensajes necesario para transmitir cada instrucción y recibir su respuesta, utilizando el protocolo T=1.

Para ello, divide el mensaje APDU en los diferentes mensajes necesarios según su longitud y la longitud máxima de mensajes aceptados por la Smart Card (IFSC), espera la petición del siguiente mensaje y repite el proceso hasta que el APDU ha sido enviado en su totalidad.

En el envío de estos mensajes se ayuda de dos funciones auxiliares, la primera de ellas para la creación del byte final de los mensajes T=1, mediante la función "LRC". Esta función calcula la función XOR de todos los bytes enviados en cada mensaje a modo de control de errores de la transmisión necesario en el protocolo T=1. Por otro lado, para el control del bit de secuencia de transmisión (incluido en el byte PCB de la cabecera de cada mensaje mensaje) se utiliza la función auxiliar "cambioPCB_envio".

Una vez finalizada la transmisión completa del APDU, esta función se encarga de esperar la respuesta completa por parte de la Smart Card, gestionando las diferentes partes de la respuesta recibida, rellenando un array de bytes con la respuesta

completa y pidiendo en cada recepción parcial el envío de la siguiente en el momento de estar lista para su tratamiento.

Utilizando la función auxiliar "LRC" comprueba en cada recepción parcial que el LRC enviado en el último byte del mensaje es correcto, comprobando también que la recepción es la correspondiente a la petición realizada mediante el bit de secuencia del byte PCB recibido.

Una vez finalizado todo el proceso la función, devuelve la respuesta completa de la Smart Card en un parámetro por referencia para su análisis posterior.

En estos ficheros se define así mismo la función "SCCambioMaxIF" para el envío del mensaje S-Block de cambio de la longitud máxima del campo INF en los mensajes del protocolo T=1.

5.7.3. Memoria Utilizada

El μprocesador utilizado para este prototipo tiene una memoria Flash de 256Kb y una memoria SRAM de 32Kb. Para calcular el espacio en la memoria Flash utilizado por las funciones programadas para este prototipo, y diferenciar este espacio del espacio ocupado por la función de control de la tarjeta WiFi y del cargador de Arduino, se ha realizado un programa únicamente habilitando la conectividad WiFi.

Este programa que únicamente habilita la comunicación WiFi ocupa, según el compilador, 41804 bytes (fig. 5.6).

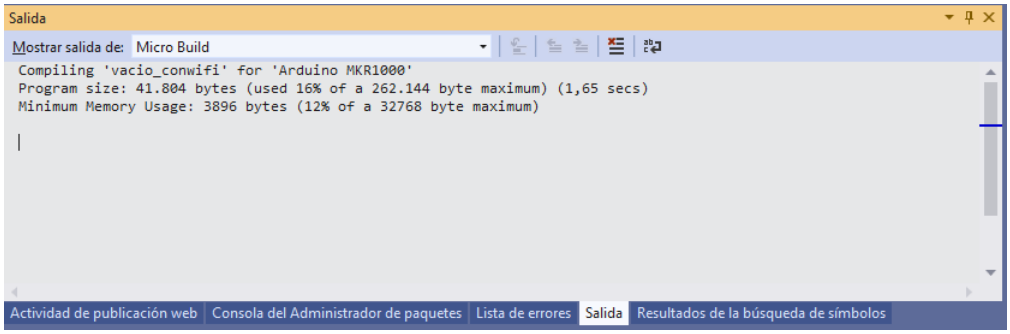


Figura 5.6: Memoria usada programa conexión WiFi μprocesador.

El programa del prototipo completo ocupa, según el compilador, 48980 bytes (fig. 5.7), de lo que deducimos que el tamaño ocupado en la memoria Flash del μprocesador por las funciones específicas de este prototipo es de 7176 bytes.

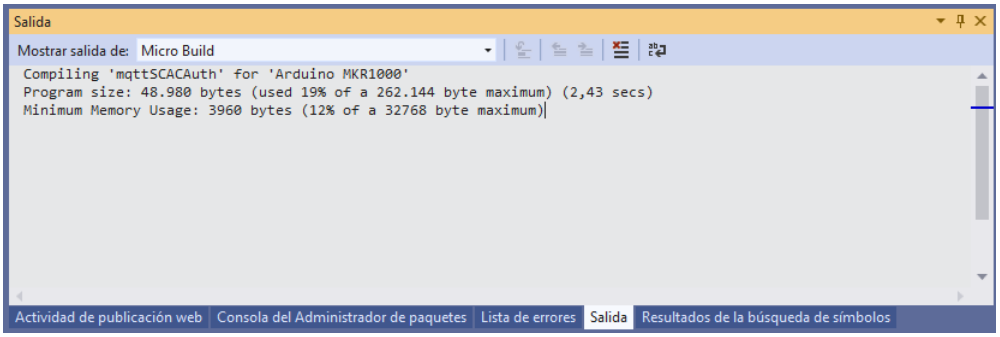


Figura 5.7: Memoria usada prototipo completo μprocesador.

Para la comprobación de la memoria SRAM utilizada en el programa se utiliza la función "freeMemory" y se envía por el puerto USB, utilizado para la comunicación con el entorno de desarrollo, el valor de la memoria libre en diferentes puntos del bucle principal del programa. Pudiendo así comprobar que no se reserva memoria de manera recurrente en la ejecución cíclica del programa, reservando toda la memoria necesaria durante la inicialización del μprocesador. El valor de memoria SRAM libre una vez reservada toda la memoria necesaria es de 25467 bytes, por lo que la memoria SRAM utilizada para todo el proceso es 7301 bytes.

5.8. Prototipo Basado en PLC

5.8.1. Hardware

Para la implementación de este prototipo (fig. 5.8) se ha utilizado como controlador un PLC S7-1212C (6ES7212-1HE40-0XB0) [d2] de la firma Siemens. Este PLC fue lanzado a la venta en 2014 y tiene integrado un interfaz ethernet que se utiliza, en este prototipo, para la conexión con el broker. Se selecciona este dispositivo como representativo de los PLCs de bajos recursos utilizados en la industria y por su disponibilidad.

Figura 5.8: Prototipo PLC.

La conexión entre el PLC y la Smart Card se realizará mediante comunicación RS-232. Se dotará al PLC de una tarjeta de comunicaciones RS-232 serie CM1241 (6ES7 241-1AH32-0XB0) [d3]. Esta tarjeta se conectará a un circuito comercial basado en el interfaz *Phoenix* para conexión de Smart Card vía RS-232 (fig. 5.9).

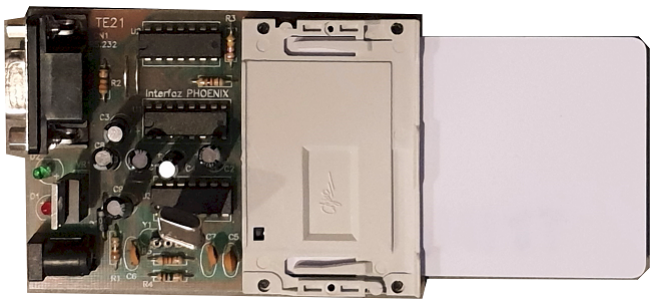


Figura 5.9: Interfaz Smart Card RS-232 (esquema en fig. C.2).

5.8.2. Software

Todo el software desarrollado para este prototipo se ha realizado con el software específico para desarrollo de aplicaciones de este tipo de PLCs, "Step 7" en su versión v14. Este software está incluido en el paquete de software de configuración y programación de toda la gama de elementos de automatización de la firma Siemens, "Tia Portal" versión 14. Es distribuido bajo licencia de pago por Siemens. Existe una versión de evaluación de este software para su descarga desde la página del fabricante [w10].

Las funciones integradas en este PLC para realizar las comunicaciones, tanto con la Smart Card como con el broker, se han realizado en lenguaje SCL. Cualquiera de los tres lenguajes posibles para la programación de este tipo de PLCs: KOP, FUP y SCL (fig. 5.10), hubiera sido igual de valido para la realización de esta implementación. Se ha seleccionado el lenguaje SCL, que es un lenguaje de alto nivel, para que el código resultante sea más comprensible, ya que los lenguajes KOP y FUP son lenguajes basados en objetos gráficos y su utilización en la realización de este prototipo hubiera dado como resultado un código mucho menos comprensible.

Este lenguaje de programación es compilable para las gamas de PLCs de Siemens S7-300, S7-400, S7-1200 y S7-1500, el software de este prototipo sería fácilmente utilizable en cualquier PLC de estas gamas, que tuviera el hardware necesario para su conexión con una red TCP/IP y comunicación RS-232.

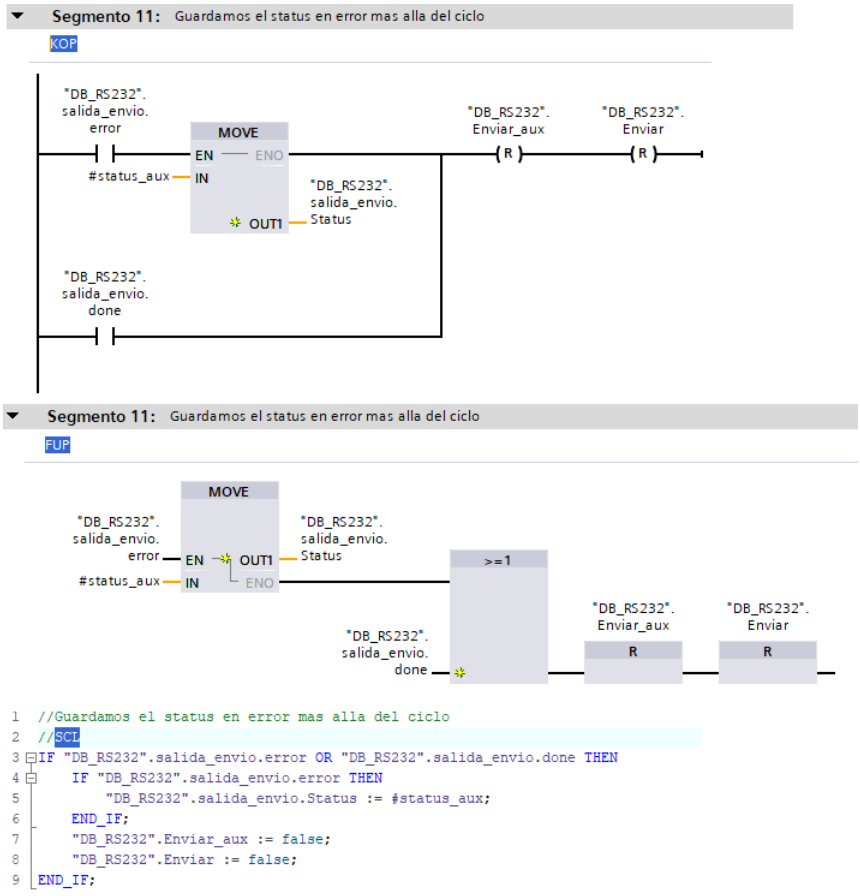


Figura 5.10: Tipos de lenguaje de programación PLC S7-1212C.

En este prototipo se ha integrado el software necesario para la autenticación del cliente con el broker mediante el sistema de autenticación MQTT-SCACAUTH descrito en esta tesis. Así como el envío de mensajes periódicos al *topic* "PLC_analogica" del valor en porcentaje de la primera entrada analógica de las dos de las que dispone el PLC.

Se ha estructurado el programa en diferentes secciones o carpetas (fig. 5.11).

La función "Main" es siempre en los PLCs de Siemens la función que genera el ciclo principal del programa, y es en esta función desde donde se llama al resto de

Figura 5.11: Estructura programa S7-1212C.

funciones del programa.

Por otro lado, la función "Startup", es por definición la función que se ejecuta en el momento de la inicialización del PLC y aquí se inicializan todas las variables utilizadas en el programa.

La función "TratamientoAnalogica", es llamada desde "Main" y realiza la conversión del valor leído en la analógica 1 del PLC en un valor porcentual.

En el objeto "DB_Config" se almacenan los valores de configuración de la comunicación del cliente, UID, *topic*, keepalive, etc...

Ciclos

Dentro de la carpeta "Ciclos" (fig. 5.12) están incluidas las funciones que gestionan los diferentes ciclos utilizados en el prototipo, "CicloCom", llamada desde la función "Main", es desde donde se gestionan los diferentes ciclos del programa y se llaman a las funciones de cada ciclo "Autentificar", "ProcesoPING" y "ProcesoPUB" que gestionan los tres ciclos principales de comunicación. Por otro lado, tenemos los cuatro bloques de datos utilizados en estas cuatro funciones.

SCard

En esta carpeta (fig. 5.13) se incluyen todas las funciones necesarias para el envío de las instrucciones a la Smart Card así como la recepción de sus correspondientes respuestas.

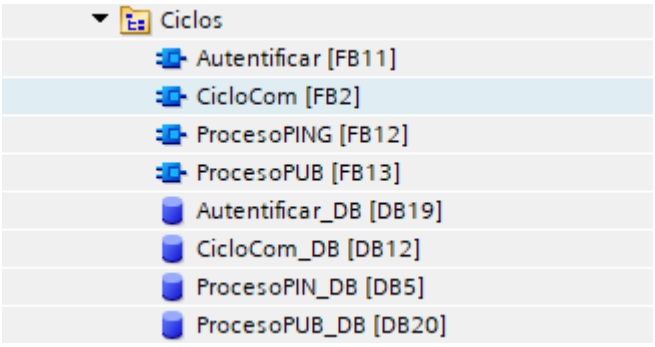


Figura 5.12: Carpeta Ciclos programa S7-1212C.

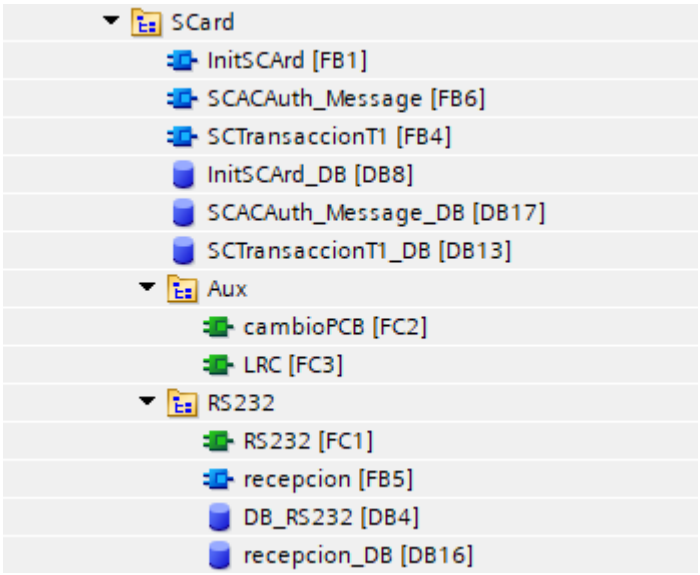


Figura 5.13: Carpeta SCard programa S7-1212C.

En la función "InitSCard" se programa el ciclo necesario para la realización de un reset a la Smart Card, así como el envío del comando PPS para su configuración a 115200 baudios y comunicación mediante el protocolo T=1. Esta función es llamada desde la función "Main".

La función "SCACAuth_Message", llamada desde la función "Main", desarrolla los diferentes mensajes APDU que debemos utilizar para el cumplimiento de los

objetivos de este prototipo:

- "Create_Auth_Step1"
- "Check_Auth_Step2"
- "Create_Auth_Step3"
- "Create_Topic"
- "Create_Payload"
- "Check_Pubrec"

Para el envío de estas instrucciones mediante el protocolo T=1 y la recepción de sus respuestas se utiliza la función "SCTransaccionT1", con el mismo funcionamiento que la función auxiliar con el mismo nombre descrita en el prototipo realizado con un μprocesador.

En esta carpeta también se incluyen los bloques de datos necesarios para la ejecución de estas funciones: "InistSCard_DB", "SCACAuth_Message_DB" y "SCTransaccionT1_DB".

La subcarpeta "Aux" contiene dos funciones auxiliares para la ejecución de las funciones de esta carpeta, "cambioPCB" se utiliza para el control del bit de secuencia del protocolo T=1. Para la creación del byte de control LRC de los mensajes APDU enviados a la Smart Card y la comprobación de este bytes recibidos como respuesta se utiliza la función "LRC".

La subcarpeta "RS232" contiene la función "RS232" necesaria para la comunicación entre la CPU del PLC y la carta de comunicación RS-232 y la configuración de dicha carta.

Para la programación de este prototipo se debe tener en cuenta que las funciones utilizadas de envío y recepción entre la CPU a la carta de comunicación RS-232 se pueden ejecutar en varios ciclos de ejecución del programa en la CPU.

Por otro lado, contiene la función "recepcion", se encarga de, una vez recibida una trama de datos desde la tarjeta de comunicación RS-232, tratarla y almacenarla en el bloque de datos correspondiente para su tratamiento en el resto del programa.

Esta subcarpeta contiene los bloques de datos necesarios para la correcta ejecución de estas funciones, "DB_RS232" y "recepcion_DB".

SCACAuthMQTT

Esta carpeta (fig. 5.14) incluye las funciones necesarias para el intercambio de datos del cliente asociado al PLC con el broker al que se asocia.

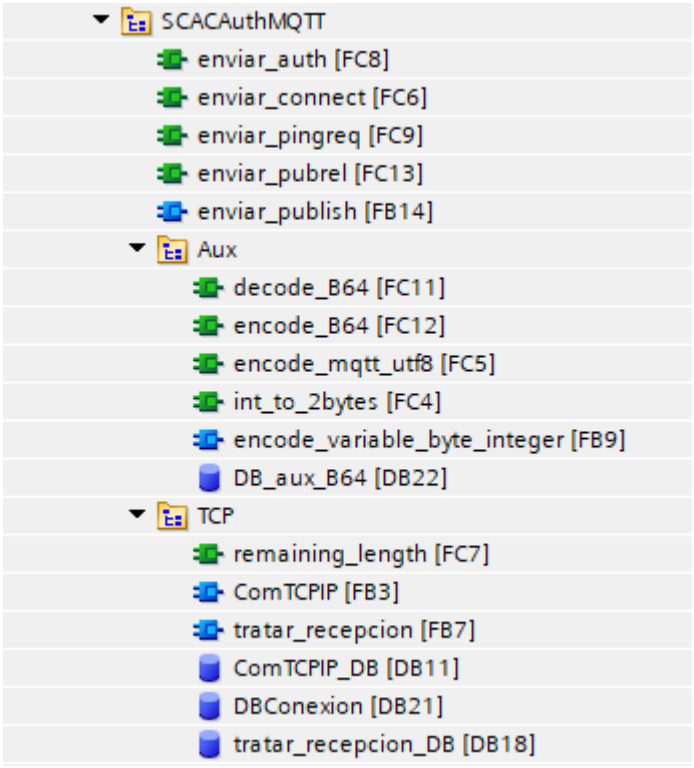


Figura 5.14: Carpeta SCACAuthMQTT programa S7-1212C.

En esta carpeta tenemos las funciones donde se construyen los mensajes a intercambiar con el broker, "enviar_auth", "enviar_connect", "enviar_pingreq", "enviar_pubrel" y "enviar_publish" construyen los correspondientes mensajes para ser enviados al broker.

Para poder realizar estos mensajes se utilizan una serie de funciones auxiliares contenidas en la subcarpeta "Aux". Para codificar los datos en los formatos

requeridos por el estándar MQTT. Utilizamos la función "encode_mqtt_utf8" para construir cadenas de caracteres UTF-8. Para componer enteros en 2 bytes se utiliza "int_to_2bytes" y para construir codificaciones de enteros de longitud variable "encode_variable_byte_integer".

Por otro lado, los mensajes cifrados, que debemos enviar en campos que el estándar MQTT define como cadenas de caracteres UTF-8, los codificamos en formato Base64. Para transformar estos datos utilizaremos las funciones "encode_B64" y "decode_B64". Las tablas de codificación necesarias para estas funciones se almacenan en el bloque de datos "DB_aux_B64".

En la subcarpeta "TCP" se incluyen las funciones necesarias para el envío y recepción de datos mediante el interfaz ethernet incluido en el hardware de la CPU.

En la función "ComTCPIP" se incluyen las funciones estándar de configuración y comunicación de datos a través del interfaz ethernet mediante TCP/IP. Estas funciones, al igual que ocurre con las funciones de comunicación RS-232, se pueden ejecutar en varios ciclos de ejecución de la CPU.

Por otro lado, contiene la función "tratar_recepcion", que se encarga de, una vez recibida una trama de datos desde el interfaz ethernet, tratarla y almacenarla en el bloque de datos correspondiente para su tratamiento en el resto del programa.

La función "ramaining_length" se encarga de leer el entero en longitud variable de la cabecera fija de los mensajes MQTT recibidos, para conocer la longitud restante que acompaña a dicha cabecera.

Esta subcarpeta también incluye el bloque de datos de configuración de la conexión ethernet (ip, puerto, tipo de conexión, interfaz....) "DBConexion" así como los bloques de datos necesarios para la ejecución de la comunicación "ComTCPIP_DB" y "tratar_recepcion_DB".

5.8.3. Memoria Utilizada

Las funciones y bloques de datos creados para la realización de la comunicación con el broker y con la Smart Card ocupan el tamaño de memoria de trabajo en el

PLC que podemos ver en las tablas 5.1 y 5.2.

Tabla 5.1: Memoria utilizada objetos PLC 1/2

Nombre	Objeto	Memoria utilizada
RS232	FC1	1570 bytes
cambioPCB	FC2	44 bytes
LRC	FC3	102 bytes
int_to_2bytes	FC4	54 bytes
encode_mqtt_utf8	FC5	184 bytes
enviar_connect	FC6	788 bytes
remaining_length	FC7	152 bytes
enviar_auth	FC8	377 bytes
enviar_pingreq	FC9	70 bytes
TratamientoAnalogico	FC10	77 bytes
decode_B64	FC11	720 bytes
encode_B64	FC12	672 bytes
enviar_pubrel	FC13	163 bytes
InitSCArd	FB1	1298 bytes
CicloCom	FB2	423 bytes
ComTCPIP	FB3	1120 bytes
SCTransaccionT1	FB4	1628 bytes
recepcion	FB5	483 bytes
SCACAuth_Message	FB6	1325 bytes
tratar_recepcion	FB7	291 bytes
encode_variable_byte_integer	FB9	156 bytes
Autentificar	FB11	1314 bytes
ProcesoPING	FB12	252 bytes
ProcesoPUB	FB13	2365 bytes
enviar_publish	FB14	732 bytes
DB_RS232	DB4	906 bytes
ProcesoPIN_DB	DB5	116 bytes
InitSCArd_DB	DB8	156 bytes
DB_Config	DB10	86 bytes
ComTCPIP_DB	DB11	1332 bytes
CicloCom_DB	DB12	164 bytes

Tabla 5.2: Memoria utilizada objetos PLC 2/2

Nombre	Objeto	Memoria utilizada
SCTransaccionT1_DB	DB13	1204 bytes
recepcion_DB	DB16	884 bytes
SCACAuth_Message_DB	DB17	1236 bytes
tratar_recepcion_DB	DB18	676 bytes
Autentificar_DB	DB19	116 bytes
ProcesoPUB_DB	DB20	548 bytes
DBConexion	DB21	22 bytes
DB_aux_B64	DB22	428 bytes

Podemos ver así que la inclusión de esta comunicación en el PLC consume 27843 bytes de memoria de trabajo de los 76800 bytes de este modelo de CPU.

5.9. Prototipo Basado en μPC

5.9.1. Hardware

Para la realización de este prototipo se ha utilizado una placa de desarrollo Raspberry Pi 3 Model B+ [d4], muy utilizada para el desarrollo de prototipos. Estas placas, en sus diferentes versiones, se han utilizado en una amplia gama de investigaciones relacionadas con las comunicaciones en el ámbito del IoT ([63]-[65]) en los últimos años.

Este dispositivo cuenta con un interfaz ethernet por el que se conecta al broker y cuatro puertos USB a uno de estos puertos se conecta el lector comercial OMNIKEY 3021 [d5](fig. 5.15) de la firma HID, para la comunicación del dispositivo con la Smart Card.

Figura 5.15: Lector RFID HID OMNIKEY 3021.

5.9.2. Software

Para este prototipo se ha utilizado como sistema operativo la distribución de Linux "Raspbian GNU/Linux 10 (buster)", específicamente desarrollada para este tipo de dispositivos.

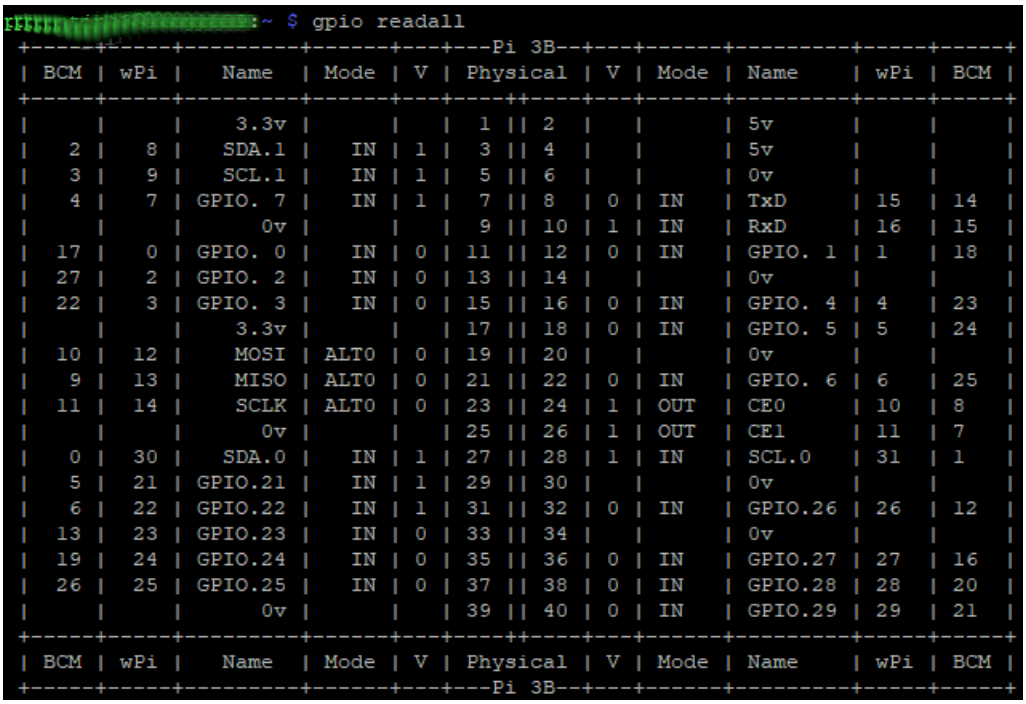
Se ha cargado a este dispositivo la librería estática "SCACAUTH_Cliente.a" (punto 5.5) y la API PCSC Lite para su correcta ejecución. Utilizando esta librería para todos los intercambios de mensajes con el broker según el esquema de seguridad MQTT-SCACAUTH.

En este prototipo se utilizan dos I/O del dispositivo para poder comprobar el correcto funcionamiento de la comunicación. Para el acceso a estas I/O se utiliza la librería "wiringpi", para descargar esta librería desde el repositorio estándar de la distribución de Linux utilizada se ejecuta la siguiente instrucción:

```
sudo apt-get install wiringpi
```

Una vez instalada esta librería se puede comprobar su funcionamiento a través de la instrucción "gpio readall". Esta instrucción nos mostrará el estado de todos los

elementos del puerto I/O del dispositivo (fig 5.16).



```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3B--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3B--+---+------+---------+-----+-----+
```

Figura 5.16: Test wiringpi.

En este prototipo ante el cambio de nivel en una de las entradas que proporciona el dispositivo Raspberry PI, envía un mensaje al *topic* "mPC_pulsador" con el *payload* "pulsado" cuando el valor de la I/O asignada cambia a valor 1 y "liberado" cuando el valor cambia a 0. También se realiza una suscripción al *topic* "micro/pulsador_micro", de tal manera que, cuando por ese *topic* se reciba el *payload* "ha sido pulsado", se active la I/O asignada y cuando se reciba "ha sido liberado" se desactive dicha I/O. Se ha conectado a esta I/O un led para que se encienda o apague cuando se reciba el *payload* correspondiente. Podemos ver este esquema de conexiones en la fig. C.3.

5.9.3. Memoria Utilizada

El programa resultante de la compilación de este prototipo, "mqttSCACAuth", tiene un tamaño de 83716 bytes.

5.10. Prototipo Basado en μPC J1939

En este prototipo se pretende demostrar la viabilidad del envío de datos mediante el esquema de seguridad MQTT-SCACAUTH desde un camión, utilizado en la logística del sistema. Las electrónicas de control (ECU por sus siglas en inglés) de los diferentes dispositivos integrados en los vehículos de transporte de mercancías se comunican entre sí mediante el protocolo J1939 en una amplia mayoría de los vehículos comercializados en Europa. Los fabricantes de vehículos pesados más importantes (Hyundai, MAN AG, Volvo, Renault, Scania, Jhon Deere, etc...) siguen esta norma en la actualidad.

El protocolo J1939 corresponde a una de las normas de la Sociedad de Ingenieros Automotrices (SAE por sus siglas en inglés). Este protocolo se transmite sobre el protocolo CAN extendido, con 29 bits en el campo CAN ID. El campo CAN ID se descompone en diferentes componentes para la identificación del mensaje J1939 (fig. 5.17)[s11].

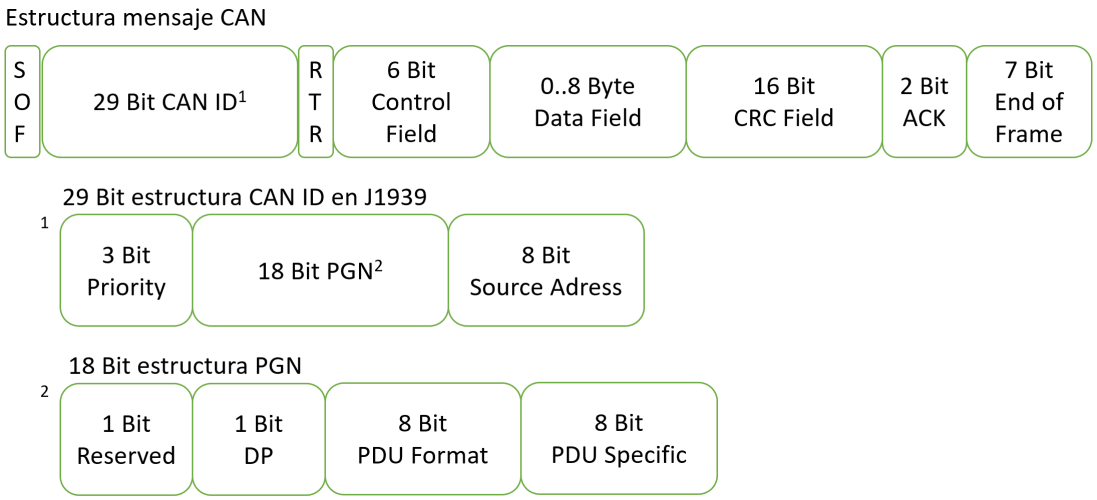


Figura 5.17: Estructura CAN ID en J1939.

El protocolo J1939 es estudiado ampliamente en la literatura científica publicada, [66]-[69].

Esta norma define una serie de números de grupo de parámetros (PGN por sus siglas en inglés) que deben estar presentes en los vehículos y deja un espacio en las definiciones de los códigos PGN para la transmisión de datos específicos del fabricante (tabla 5.3)[s12].

Tabla 5.3: Distribución PGN

Rango PGNs	Num.PGNs	SAE o Fab.	Comunicación
000000 - 00EE00	239	SAE	PDU1=Peer to Peer
00EF00	1	Fab.	PDU1=Peer to Peer
00F000 - 00FEFF	3840	SAE	PDU2=Broadcast
00FF00 - 00FFFF	256	Fab.	PDU2=Broadcast
010000 - 01EE00	239	SAE	PDU1=Peer to Peer
01EF00	1	Fab.	PDU1=Peer to Peer
01F000 - 01FEFF	3840	SAE	PDU2=Broadcast
01FF00 - 01FFFF	256	Fab.	PDU2=Broadcast

No todos los PGNs reservados para el estándar SAE están definidos, en los tramos reservados existen espacios sin definir.

Cada uno de los PGNs puede contener uno o varios parámetros (SPN por sus siglas en inglés) que se transmitirán en el mensaje asociado a dicho PGN. En la definición del protocolo se describe que parámetros se transmiten en cada PGN, si son transmitidos cíclicamente o ante un cambio, la ratio de transmisión en el caso de ser cíclicos, y la posición y codificación de los diferentes SPN que se transmiten en cada uno de los PGNs.

El protocolo J1939 define también mensajes especiales que se utilizan para enviar los defectos de todas las ECU del sistema [s13]. Estos mensajes de defecto (DM por sus siglas en inglés) se envían ante un cambio de la lista de mensajes de defecto presentes en las ECU. Tanto ante la aparición de los defectos como ante la desaparición de estos.

Cada mensaje DM puede transmitir uno o varios códigos de diagnóstico de errores (DTC por sus siglas en inglés), en el caso de transmitir más de un DTC el mensaje DM es transmitido en múltiples mensajes CAN para poder transmitir una longitud mayor de datos de los que se pueden transmitir en un mensaje CAN, para ello se utiliza la codificación multimensaje definida en el protocolo J1939.

5.10.1. Hardware

Para la realización de este prototipo se ha partido el mismo hardware que para el prototipo basado en μPC (punto 5.9), añadiendo a ese hardware la tarjeta de comunicación CAN "PICAN2 CAN-Bus Board for Raspberry Pi", fabricada por la firma SK PANG ELECTRONICS. Esta tarjeta proporciona dos interfaces CANopen a la placa de evaluación Raspberry Pi 3 Model B+.

Para poder simular el envío de datos desde las ECU de un vehículo al bus CAN del prototipo se utilizan dos tarjetas de simulación de ECU. Estas tarjetas se denominan "JCOM.J1939.USB-B" de la firma Copperhill Technologies (fig. 5.18). Nos permiten enviar cualquier comando J1939 desde la dirección configurada, con el PGN que seleccionemos debidamente formateado. Para el control de la simulación se utiliza el software de libre distribución, proporcionado por el fabricante de la tarjeta, "J1939 COM Monitor Pro". Se realiza la conexión entre la tarjeta y el ordenador donde se ejecute el software de control mediante un puerto USB.

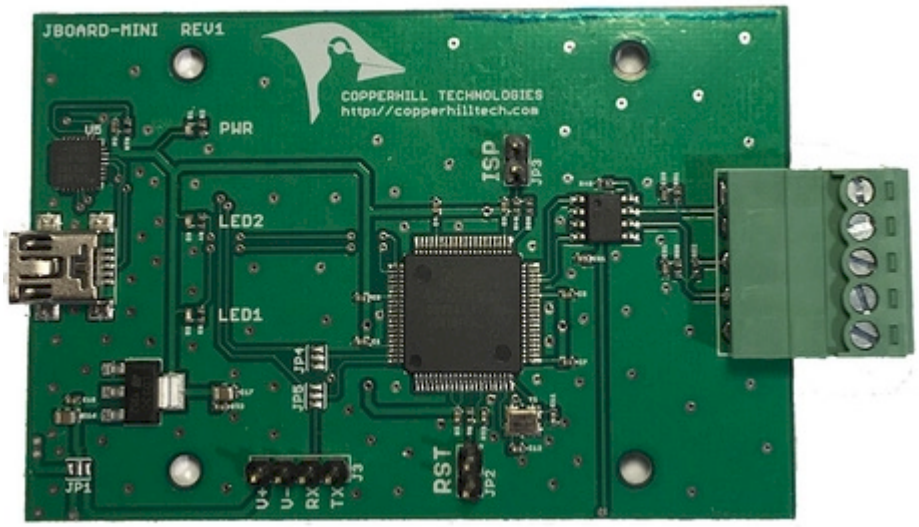


Figura 5.18: Placa de simulacion ECU J1939.

Para completar el prototipo, se conecta un dispositivo GPS a un puerto USB del prototipo, cuya referencia es VK-162, desarrollado por la firma FTVOGUE, con el objetivo de simular el envío de la geoposición del vehículo.

Todos estos datos serán enviados al servidor a través de un dongle USB 3G E3372 de la firma Huawei, dotando así de una interfaz WAN móvil al prototipo.

5.10.2. Software

En este prototipo, a partir del interfaz CAN proporcionado por la tarjeta "PiCAN2", se leen diversos PGNs estándar definidos como datos periódicos a enviar (anexo E). Estos valores se envían al broker de manera cíclica. Este envío se realiza al *topic* "/J1939_periodico/idvehiculo/", como *payload* se envía una cadena con formato JSON con todos los parámetros de envío periódico configurados y el timestamp de recogida de los datos.

Se lee el mensaje DM1 y se envía la aparición y desaparición de los defectos informados por las ECU. Los DTC transmitidos en estos mensajes identifican el

mensaje de error por medio de la dirección de envío de dicho mensaje (ECU donde se ha producido) y dos valores contenidos en el mensaje, el SPN y el indicador de modo de fallo (FMI por sus siglas en inglés). Se envía cada DTC, en su aparición y desaparición o reseteo, al *topic* "/J1939_def/idvehiculo/", como *payload* se envía una cadena JSON con la dirección de la ECU, SPN, FMI, un booleano que indica aparición o desaparición y un timestamp con el instante del apunte.

Las geoposiciones leídas se envían al *topic* "/geopos/idvehiculo/", en su *payload* se envía una cadena con formato JSON con la geoposición a enviar y la velocidad asociada.

Por último, cada vez que se produce una conexión a la red 3G se envía al servidor un mensaje al *topic* "/ips/idvehiculo/" y en su *payload* se envía una cadena JSON con la ip y el timestamp del instante de la conexión.

El software de este prototipo se ha desarrollado completamente en el lenguaje de programación Python, para ello se ha utilizado el módulo Python "SCACAuth_Cliente_Python" desarrollado en esta tesis (punto 5.5).

Este software se divide en cuatro servicios, que se interconectaran mediante una base de datos Sqlite3 configurada en el prototipo.

Esta base de datos tiene las siguientes tablas:

- geopos: en esta tabla se almacenan los valores de geoposición y velocidad a enviar hasta su envío.
- J1939_onchange: en esta tabla se almacenan los defectos leídos en los mensajes DM. Se crea un apunte en el momento de su aparición y otro en el momento de su desaparición. Los datos permanecen en esta tabla hasta el momento de su envío.
- J1939_periodico: En esta tabla se almacenan los datos cíclicos leídos de los diferentes mensajes J1939. Se realiza un apunte cada sesenta segundos con los valores a enviar. Los datos permanecen en esta tabla hasta el momento de su envío.
- ips: Cada vez que se produzca una conexión del sistema a la red 3G se almacena la ip asignada al interfaz 3G hasta su envío.

Los cuatro servicios que se han creado para este prototipo son los siguientes:

- servGPS: lee los datos obtenidos del dispositivo GPS y si ha habido un desplazamiento mayor de 50 metros desde el último apunte, realiza un nuevo apunte en la tabla geopos, con la geoposición y la velocidad leída.
- servJ1939: este servicio lee todos los mensajes que se intercambian por el bus CAN con formato J1939. Selecciona los mensajes configurados y trata los datos transmitidos en estos mensajes. Cada minuto realiza la media de todos los valores leídos por los mensajes configurados y crea un apunte en la tabla J1939_periodico. Si recibe algún mensaje DM trata dicho mensaje y realiza el apunte correspondiente en la tabla J1939_onchange. Para realizar la lectura de los mensajes en J1939 que viajen por el bus CAN se ha utilizado el módulo Python "can-j1939" de libre distribución.
- serv3G: este servicio inicia la comunicación 3G a través del dongle 3G instalado en el sistema, configura el firewall para únicamente permitir las comunicaciones utilizadas en el prototipo y se mantiene vigilante de la conexión. Una vez realizada la conexión correctamente crea un apunte en la tabla ips con la ip asignada. Si se detecta que la conexión 3G se ha caído (por ejemplo, por falta de cobertura), realiza intentos de reconexión cada cierto tiempo configurable. Para la configuración del dongle 3G se ha utilizado el programa de libre distribución "wvdial" y para gestionar la seguridad de las conexiones se ha utilizado la herramienta "iptables".
- servMQTT: Este servicio se conecta al broker MQTT asignado mediante el módulo Python "SCACAuth_Cliente_Python.so". Cada 60 segundos comprueba si existen apuntes sin enviar en la base de datos del prototipo. Si existe algún apunte en alguna tabla, envía este apunte al broker asociando los datos al *topic* correspondiente dependiendo de en qué tabla está ese apunte y del identificador de vehículo configurable. Una vez recibido el evento que confirma que la publicación ha finalizado correctamente, borra el apunte correspondiente de la base de datos.

5.10.3. Memoria Utilizada

En la siguiente tabla 5.4 podemos ver el tamaño de todos los módulos desarrollados para este prototipo.

Tabla 5.4: Memoria prototipo μPC J1939

Módulo	Tamaño (bytes)
Módulos servGPS	
aux_GPS.pyc	2404
servGPS.pyc	1319
Total	3723
Módulos servJ1939	
clases_datos.pyc	955
esc_msg.pyc	1331
servj1939.pyc	3679
Total	6005
Módulos serv3G	
ArrancarUSB.pyc	1047
const3G.pyc	2219
ControlDongle.pyc	4048
serv3G.pyc	2063
Total	9377
Módulos servMQTT	
servMQTT.pyc	4190
Total	4190

5.11. Prototipo Broker

Para la realización del broker que cumpla con el esquema de seguridad MQTT-SCACAUTH se ha partido del broker de código abierto, bajo licencia EPL/EDL, "mosquitto" [70], en su versión 2.0.4.

El broker "mosquitto" es parte de la fundación Eclipse dentro de su proyecto "Open Source for IoT" [w11]. El desarrollo de "mosquitto" fue realizado por Roger Light y su código está accesible en el repositorio de código GitHub para su libre acceso [w12].

Se ha añadido, en el fichero de configuración de la compilación del proyecto "config.mk", una nueva variable denominada WITH_SCACAUTH para poder realizar la compilación de este proyecto con o sin el esquema de seguridad MQTT-SCACAUTH. Esta variable cambia las variables de compilación preexistentes para añadir la librería "libSCACAuth_Applet_Broker.a" a la compilación de los diferentes módulos que componen el proyecto. Además se incluye la macro de preprocesado homónima, utilizada para seleccionar los segmentos de código a compilar cuando deseamos compilar el proyecto con el esquema de seguridad MQTT-SCACAUTH o sin él.

```
ifeq ($(WITH_SCACAUTH),yes)
        SCACAUTH_CARD:= -L../ -lSCACAuth_Applet_Broker
            -I/usr/include/PCSC -lpcsclite -DWITH_SCACAUTH
        APP_CPPFLAGS:=$(APP_CPPFLAGS) $(SCACAUTH_CARD)
        BROKER_CPPFLAGS:=$(BROKER_CPPFLAGS) $(SCACAUTH_CARD)
        CLIENT_CPPFLAGS:=$(CLIENT_CPPFLAGS) $(SCACAUTH_CARD)
        LIB_CPPFLAGS:=$(LIB_CPPFLAGS) $(SCACAUTH_CARD)
        LIB_CLAGS:=$(LIB_CFLAGS) $(SCACAUTH_CARD)
        CLIENT_LDFLAGS:=$(CLIENT_LDFLAGS) $(SCACAUTH_CARD)
        BROKER_LDADD:=$(BROKER_LDADD) $(SCACAUTH_CARD)
endif
```

En el mismo fichero de configuración se modifica la variable "VERSION" del valor "2.0.4" al valor "SCACAUTH1.0".

Todas las modificaciones del código de este proyecto se han encapsulado con la macro de preprocesado WITH_SCACAUTH para compilarlo únicamente cuando en el fichero de configuración de la compilación se active la compilación con MQTT-SCACAUTH.

```
#ifdef WITH_SCACAUTH
    ...nuevo...
#else
    ...preexistente...
#endif
```

En el fichero de configuración "mosquitto.conf", donde se configura el broker, se ha añadido una nueva variable, denominada "scard_reader", para designar el lector a utilizar en las comunicaciones con la Smart Card del broker.

Para la utilización de la librería de acceso al applet del broker, en todos los módulos del programa, se incluye en el fichero de cabecera "mosquitto.h" la llamada al fichero "SCACAuth_Broker.h".

Se han realizado las modificaciones necesarias en la función "config__read_file_core" del fichero "conf.c" para la lectura de esta nueva propiedad. Esta propiedad se almacena en la propiedad "scard_reader" de la estructura modificada "mosquitto__config" definida en el fichero "mosquitto_broker_internal.h".

En el fichero "mosquitto.c", donde se encuentra la función "main" del programa, se ha insertado el código necesario para la inicialización del lector que se utiliza en el resto del programa para la comunicación con la Smart Card. Si la propiedad "scard_reader" no está definida en el fichero de configuración, en el arranque del programa se consulta el lector a utilizar para la comunicación con la Smart Card.

Las funciones "mosquitto_main_loop" del fichero "loop.c", "mux_epoll__handle" del fichero "mux_epoll.c", "mux_poll__handle" del fichero "mux_poll.c", y "mux__add_in" definida en los ficheros "mux.c / .h", se han modificado para poder definir los nuevos clientes que se conectan al broker con los datos de la Smart Card utilizada.

Los datos relacionados con cada cliente activo se almacenan en una estructura "mosquitto" definida en el fichero "mosquitto_internal.h". Esta estructura se ha modificado para la inclusión de los datos necesarios para la utilización del esquema de seguridad definido en esta tesis. Estos nuevos datos son los relacionados con la Smart Card que se debe utilizar para la llamada a las funciones del broker y una variable booleana para el cumplimiento de la premisa [SCACAUTH-PRE-4], el no envío de un mensaje hasta la finalización del proceso desencadenado por el mensaje anterior.

En la función "db__message_write_inflight_out_single" del fichero "database.c" se realiza el bloqueo del envío de nuevas publicaciones hasta que se finalice el proceso desencadenado por el envío anterior.

En los ficheros "util_mosq.c /.h" se han desarrollado las funciones para la codificación y decodificación en Base 64, necesarios para la transmisión de elementos cifrados en propiedades codificadas en UTF-8.

5.11.1. Autenticación

El broker "mosquitto" está preparado para incluir el código necesario para realizar una autenticación en varios pasos definido en la versión 5.0 del protocolo MQTT [s1]. Este código se debe generar como un plug-in compilado como un módulo separado del proyecto principal. Para ello define un esquema para la realización de este plug-in cuyas estructuras y funciones están definidas en el fichero "mosquitto_plugin.h".

En la realización del plug-in para la autenticación según el esquema de seguridad MQTT-SCACAUTH se ha creado un nuevo directorio denominado "SCACAuth_plugin", a partir del directorio raíz del proyecto. En este directorio se han colocado todos los ficheros necesarios para la creación de este plug-in así como su fichero de compilación.

Los ficheros generados son los siguientes:

- "SCACAuth.c": en este fichero se desarrollan las funciones definidas para un plug-in de autenticación en varios pasos del proyecto "mosquitto", creando estas funciones para el correcto cumplimiento del esquema de seguridad MQTT-SCACAUTH.
- "Aux_SCACAuth.c / .h": en estos ficheros se definen las funciones auxiliares, estructuras y constantes que se utilizan en "SCACAuth.c".

La compilación de este plug-in da como resultado el objeto "SCACAuth.o". En el fichero "mosquitto.conf" se rellena la variable de configuración "auth_plugin" con el objeto generado.

```
auth_plugin SCACAuth_plugin/SCACAuth.o
```

5.11.2. Intercambio de Mensajes

Para el cumplimiento del esquema de seguridad MQTT-SCACAUTH en el intercambio de mensajes entre el broker y el cliente, una vez finalizada la autenticación, se han modificado 9 ficheros del proyecto inicial (fig. 5.19).

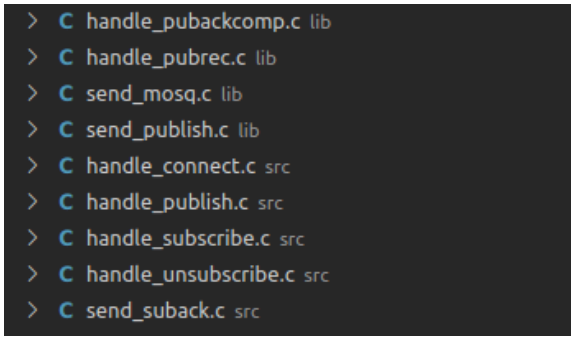


Figura 5.19: Ficheros modificados en "mosquitto" para intercambio de mensajes.

Una vez realizadas estas modificaciones se consigue el cumplimiento de todos los aspectos necesarios para la utilización del esquema de seguridad MQTT-SCACAUTH.

En los manejadores de recepción de mensajes ("handle_*.c / .h") y de envío de mensajes ("send_*.c / .h") se ha incluido el código necesario para el tratamiento de los componentes específicos del esquema de seguridad MQTT-SCACAUTH. Para la generación y comprobación de estos componentes se realizan las llamadas necesarias a las funciones expuestas por la librería "libSCACAUth_Applet_Broker.a".

- "handle_pubackcomp.c": se modifica la función "handle__pubackcomp" para liberar el envío de nuevas publicaciones [SCACAUTH-PRE-4].
- "handle_pubrec.c": se modifica la función "handle__pubrec" para la comprobación de los datos específicos del esquema de seguridad con la función "check_pubrec_broker'.
- "send_mosq.c": se modifica la función "send__pubrec" para la creación de los datos específicos del esquema de seguridad con la función "create_pubrec_broker".
- "send_publish.c": se modifica la función "send__publish" para bloquear el envío de nuevas publicaciones hasta la finalización del proceso desencadenada por esta [SCACAUTH-PRE-4]. Además se cifra el *topic* y el *payload* a enviar con las funciones "create_payload_broker" y "create_topic_broker".
- "handle_connect.c": se realiza la modificación de la función "handle__connect". En el caso de un fallo de autenticación se envía un mensaje CONNACK con código de respuesta 0x87 (NOT AUTHORIZED).

- "handle_publish.c": se modifica la función "handle__publish" para la decodificación del *payload* y el *topic* recibidos mediante las funciones "read_payload_broker" y "read_topic_broker".
- "handle_subscribe.c": se modifica la función "handle__subscribe" para el descifrado del *topic* recibido mediante la función "read_topic_broker_sub".
- "handle_unsubscribe.c": se modifica la función "handle__unsubscribe" para el descifrado del *topic* recibido con la función "read_topic_broker_sub". También en la misma función se genera el cifrado necesario para el envío del mensaje UNSUBACK mediante la función "create_sub_unsub_ack".
- "send_suback.c": se modifica la función "send__suback" para la inclusión de los datos cifrados correspondientes mediante la función "create_sub_unsub_ack".

Capítulo 6

Resultados Experimentales

En este capítulo se exponen los resultados obtenidos de los diferentes experimentos realizados. Se pretende comprobar el tiempo necesario para la ejecución de las funciones propias del esquema de seguridad MQTT-SCACAUTH. También se realiza un estudio del consumo eléctrico de las funciones criptográficas utilizadas en el esquema propuesto en esta tesis.

6.1. Tiempo de Ejecución

Todos los ficheros de datos obtenidos en esta experimentación, así como el proyecto RStudio utilizado para la realización de cálculos y generación de las gráficas, están disponibles en el repositorio de esta tesis [w1].

6.1.1. Cliente

Para la toma de datos de tiempo de ejecución de las diferentes funciones criptográficas, en el esquema de seguridad MQTT-SCACAUTH, se ha realizado una modificación del prototipo cliente generado con un μprocesador (punto 5.7). Se ha modificado el software de este prototipo para que realice un número determinado de iteraciones de todos los procesos a estudiar. Configurando con constantes el número de iteraciones a realizar para cada tipo de proceso.

En el prototipo original no se realizan los procesos de suscripción y recepción de publicación, es por ello por lo que se han añadido las funciones necesarias para la realización de las iteraciones de estos procesos en el prototipo utilizado para la medición de tiempos.

Se desea estudiar el tiempo total empleado en el envío del APDU, ejecución de la instrucción y recepción de la respuesta.

Para ello se calcula el tiempo con la ayuda de la función "micros" y se envía el dato obtenido por el puerto serie de depuración para su recogida.

La función "micros" nos devuelve el tiempo transcurrido desde el inicio de la ejecución del programa en microsegundos.

Se toma el tiempo antes y después de la ejecución de la función "SCTransaccionT1", la diferencia entre ambos es el tiempo empleado para el envío del APDU, ejecución de la instrucción y recepción de la respuesta.

```
unsigned long tiempo1 = micros();
SCTransaccionT1(conexion, mensaje,
```

```
        longitud_enviar, &longitud_respuesta, PCB, respuesta);
unsigned long tiempo2 = micros();
Serial.print(tiempo2 - tiempo1); Serial.print(";");
```

6.1.2. Broker

Con el fin de calcular el tiempo de ejecución de las funciones en la Smart Card del broker se añade una nueva variable en el fichero "config.mk", WITH_TIMECSV, en el prototipo del broker (punto 5.11). Esta variable nos da la opción de poder compilar el proyecto del broker con o sin el almacenamiento de estos tiempos en archivos csv. Esta variable introduce una nueva macro de preprocesado homónima que compila o no los fragmentos de código dedicados al almacenamiento del tiempo de ejecución.

Se utiliza la función "gettimeofday", de la librería "sys/time.h", que nos devuelve el tiempo transcurrido desde el 1 de enero de 1970, con una precisión de microsegundos. Esta función es invocada antes y después de la ejecución de la función a comprobar y se calcula su diferencia con la función "timersub" de la misma librería. Estos tiempos son almacenados en el archivo csv correspondiente.

```
#ifdef WITH_TIMECSV
    gettimeofday(&tval_before, NULL);
#endif
.....Smart Card........
#ifdef WITH_TIMECSV
    gettimeofday(&tval_after, NULL);
    timersub(&tval_after, &tval_before, &tval_result);
    diftiempo=tval_result.tv_usec;
    ....escribe en fichero correspondiente.....
#endif
```

6.1.3. Autenticación Mutua. Cifrado Asimétrico

Para el cálculo de tiempos en una autenticación mutua se realiza el siguiente proceso de llamadas a las funciones de los applets correspondientes:

- "create_auth_step1" (Smart Card cliente). Crea los datos para el primer paso de la autenticación.
- "check_auth_step1" (Smart Card broker). Comprueba los datos del primer paso de la autenticación.
- "create_auth_step2" (Smart Card broker). Crea los datos para el segundo paso de la autenticación.
- "check_auth_step2" (Smart Card cliente). Comprueba los datos del segundo paso de la autenticación.
- "create_auth_step3" (Smart Card cliente). Crea los datos para el tercer paso de la autenticación.
- "check_auth_step3" (Smart Card broker). Comprueba los datos del tercer paso de la autenticación.

Para el estudio de estos tiempos se han realizado 100 iteraciones del proceso de autenticación, los datos obtenidos se almacenan en los ficheros "autB.csv" y "autC.csv".

Podemos ver los resultados de este estudio en la siguiente figura (fig. 6.1).

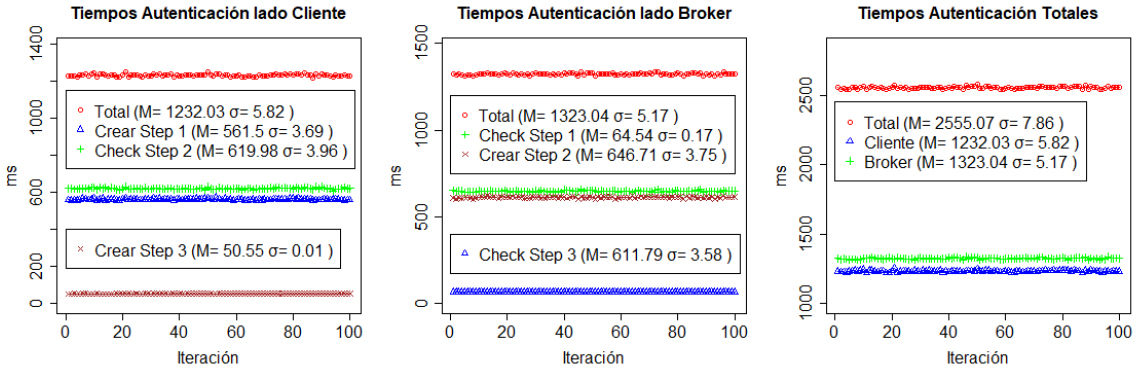


Figura 6.1: Tiempos Autenticación Mutua.

Como podemos ver, el tiempo total de la autenticación mutua es de media de 2555,07 ms, 1232,03 ms en las funciones ejecutadas en el cliente y 1323,04 ms en las ejecutadas en el broker. El tiempo total de ejecución es muy estable, con una desviación típica de 7,86 ms.

En la tabla 6.1 se muestra un resumen de los tiempos obtenidos en esta prueba.

Tabla 6.1: Resumen tiempos Autenticación Mutua

Función	Lugar de ejecución	Media (ms)	σ (ms)
Crear Step 1	Cliente	561,5	3,69
Check Step 1	Broker	64,54	0,17
Crear Step 2	Broker	646,71	3,75
Check Step 2	Cliente	619,98	3,96
Crear Step 3	Cliente	50,55	0,01
Check Step 3	Broker	611,79	3,58
	Total	2555,07	7,86

Los resultados obtenidos son los esperables teniendo en cuenta otras investigaciones con experimentación en criptografía asimétrica con Smart Card. En el artículo de Almeida *et al.* [71], se realiza una implementación mejorada del SCP10 [s14], que utiliza criptografía asimétrica, en un applet JavaCard. Los autores realizan una serie de experimentos con la misma Smart Card que la utilizada en esta tesis, J3H145. En

estas experimentaciones se comprueba que una autenticación mutua tiene un tiempo de ejecución de 2.76 segundos con el SPC10 original y de 4.11 segundos con el SPC10 mejorado.

6.1.4. Cifrado Simétrico

En el intercambio de mensajes entre el broker y un cliente para una publicación se realizan tres cifrados y sus correspondientes descifrados con el algoritmo de criptografía simétrica seleccionado, *topic*, *payload* y los datos adicionales enviados en el mensaje PUBREC.

El *topic* y el *payload* pueden tener una longitud variable de entre 11 y 256 bytes. Este tamaño se corresponde con la longitud del UID (8 bytes en cumplimiento de la premisa [SCACAUTH-PRE-2]), dos bytes que indican la longitud del componente cifrado, más de 1 a 246 bytes (en cumplimiento de la premisa [SCACAUTH-PRE-6]) del propio componente. Para hacer posible el cifrado simétrico seleccionado (AES) esta longitud se rellena hasta un múltiplo de la longitud de la clave de cifrado (16 bytes en este prototipo).

El tamaño de los datos adicionales enviados en el mensaje PUBREC siempre será el mismo ya que siempre se cifra una longitud igual al tamaño del UID (8 bytes) más dos números aleatorios de longitud igual a la clave de cifrado (16 bytes). Por lo que siempre se cifra un conjunto de datos de 40 bytes que son rellenados hasta una longitud de 48 bytes.

En el intercambio de mensajes entre el broker y un cliente para la creación de una suscripción o su anulación se realiza el cifrado y descifrado de dos componentes. El *topic* enviado en el mensaje SUBSCRIBE o UNSUBSCRIBE, con un tamaño variable de entre 1 y 246 bytes, para un cifrado de entre 11 y 256 bytes, al igual que en el *topic* enviado en una publicación. El otro componente cifrado en este intercambio de mensajes son los datos adicionales enviados en el mensaje SUBACK o UNSUBACK. Estos datos se componen del UID del cliente más un número aleatorio de longitud igual a la clave simétrica seleccionada (16 bytes). Estos 24 bytes son rellenados hasta una longitud de 32 bytes.

La longitud de los datos cifrados en el envío de mensajes afecta notablemente al

tiempo de ejecución de las funciones de cifrado y descifrado realizadas en la Smart Card. Es por ello que primero se realiza el cálculo del tiempo de cifrado y descifrado simétrico para diferentes longitudes y a continuación se hará un estudio completo de las diferentes funciones que intervienen en cada tipo de intercambio de mensajes con la longitud máxima en los campos de longitud variable.

Se realiza el cifrado y descifrado para los diferentes tamaños del componente a cifrar, teniendo en cuenta que estas longitudes tienen que ser múltiplos de la longitud de la clave de cifrado, las posibles longitudes son los múltiplos de 16 entre 16 y 256. Para realizar esta experimentación se realizan cifrados y descifrados de longitud 16, 32, 64, 128 y 256 bytes. Tanto el cifrado como el descifrado se realiza desde el cliente y desde el broker. En la figura 6.2 podemos ver los resultados de estas pruebas. En todas las pruebas se han realizado 100 iteraciones.

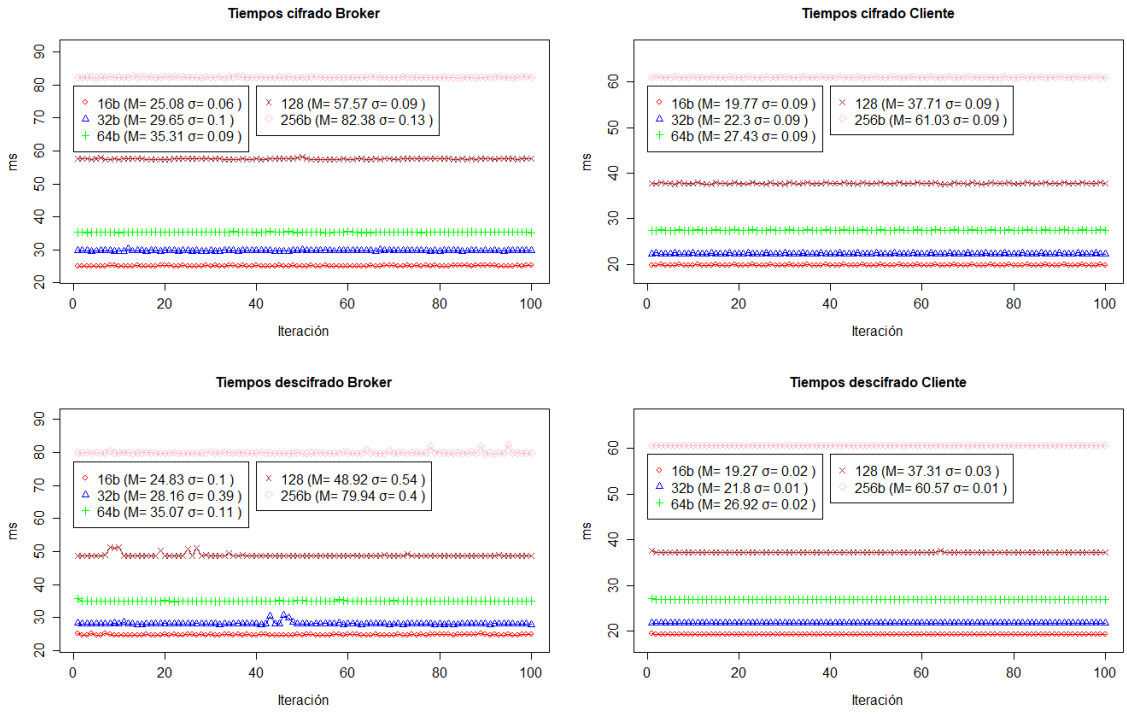


Figura 6.2: Tiempos cifrado y descifrado simétrico.

En todos los casos vemos que el tiempo de realización de una operación en la

Smart Card del cliente es sensiblemente inferior a la realizada en la Smart Card del broker. Esta diferencia está justificada porque en la Smart Card del broker los números aleatorios que se utilizan como claves de cifrado simétrico se han almacenado en memoria EEPROM, por las restricciones de memoria RAM de la Smart Card. En la Smart Card del cliente estos números se almacenan en memoria RAM, mucho más rápida. Este es el motivo de que los procesos de criptografía simétrica son más lentos en la Smart Card del broker que en la del cliente.

Los datos medios obtenidos en este experimento se pueden ver en la siguiente tabla 6.2.

Tabla 6.2: Resumen tiempos cifrado simétrico.

Operación	Broker		Cliente	
	Media (ms)	σ (ms)	Media (ms)	σ (ms)
Cifrado 16b	25,08	0,06	19,77	0,09
Cifrado 32b	29,65	0,10	22,30	0,09
Cifrado 64b	35,31	0,09	27,43	0,09
Cifrado 128b	57,57	0,09	37,71	0,09
Cifrado 256b	82,38	0,13	61,03	0,09
Descifrado 16b	24,83	0,10	19,27	0,02
Descifrado 32b	28,16	0,39	21,80	0,01
Descifrado 64b	35,07	0,11	26,92	0,02
Descifrado 128b	48,92	0,54	37,31	0,03
Descifrado 256b	79,94	0,40	60,57	0,01

6.1.5. Publicación $B \Rightarrow C_x$

En este proceso de cálculo de tiempos en una publicación desde el broker a un cliente se realiza el siguiente proceso de llamadas a las funciones de los applets correspondientes.

- "create_payload_broker" (Smart Card broker). Cifra el *payload* a enviar (246 bytes).

- "create_topic_broker" (Smart Card broker). Cifra el *topic* a enviar (246 bytes).
- "read_payload" (Smart Card cliente). Descifra el *payload* recibido y comprueba su validez (246 bytes).
- "read_topic" (Smart Card cliente). Descifra el *topic* recibido y comprueba su validez (246 bytes).
- "create_pubrec" (Smart Card cliente). Crea el cifrado para su envío en el mensaje PUBREC.
- "check_pubrec_broker" (Smart Card broker). Descifra y comprueba la validez del cifrado recibido en el mensaje PUBREC.

Se han realizado 100 iteraciones del proceso de publicación. Se suscribe el prototipo utilizado en las pruebas a un *topic* y desde una pequeña aplicación cliente generada para este fin se realizan las 100 publicaciones en ese *topic*. Los datos obtenidos se almacenan en los ficheros "pubBC_C.csv" y "pubBC_B.csv".

Podemos ver los resultados de este estudio en la siguiente figura (fig. 6.3).

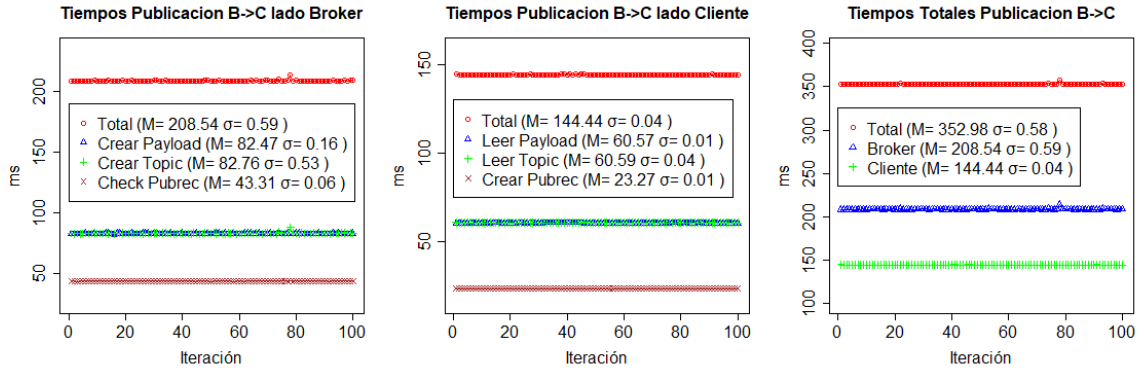


Figura 6.3: Tiempos publicación $B \Rightarrow C_x$.

Como nos muestran las gráficas de los tiempos obtenidos, la media de todos los procesos criptográficos utilizados para una publicación enviada por el broker al cliente es de 352,98 ms, es un proceso con un tiempo de ejecución muy estable (con

una desviación típica de 0,58 ms). La mayor parte del tiempo es utilizado para la realización de las funciones del broker, 144,44 ms para el cliente y 208,54 ms para el broker.

En la tabla 6.3 podemos ver un resumen de los tiempos obtenidos en esta prueba.

Tabla 6.3: Resumen tiempos Publicación $B \Rightarrow C_x$

Función	Lugar de ejecución	Media (ms)	σ (ms)
Crear *Payload*	broker	82,47	0,16
Crear *Topic*	broker	82,76	0,53
Leer *Payload*	Cliente	60,57	0,01
Leer *Topic*	Cliente	60,59	0,04
Crear PUBREC	Cliente	23,27	0,01
Check PUBREC	broker	43,31	0,06
	Total	352,98	0,58

6.1.6. Publicación $C_x \Rightarrow B$

Para el cálculo de tiempos en una publicación desde un cliente al broker se realiza el siguiente proceso de llamadas a las funciones de los applets correspondientes.

- "create_payload" (Smart Card cliente). Cifra el *payload* a enviar (246 bytes).
- "create_topic" (Smart Card cliente). Cifra el *topic* a enviar (246 bytes).
- "read_payload_broker" (Smart Card broker). Descifra el *payload* recibido y comprueba su validez (246 bytes).
- "read_topic_broker" (Smart Card broker). Descifra el *topic* recibido y comprueba su validez (246 bytes).
- "create_pubrec_broker" (Smart Card broker). Crea el cifrado para su envío en el mensaje PUBREC.

- "check_pubrec" (Smart Card cliente). Descifra y comprueba la validez del cifrado recibido en el mensaje PUBREC.

Para el estudio de estos tiempos se han realizado 100 iteraciones del proceso de autenticación, los datos obtenidos se almacenan en los ficheros "pubCB_B.csv" y "pubCB_C.csv".

Podemos ver los resultados de este estudio en la siguiente figura (fig. 6.4).

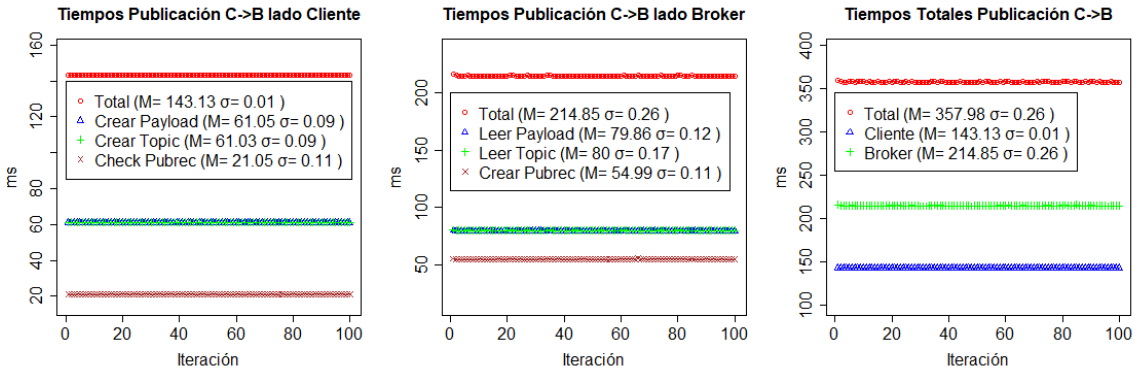


Figura 6.4: Tiempos publicación $C_x \Rightarrow B$.

Como nos muestra las gráficas de los tiempos obtenidos, la media de todos los procesos criptográficos utilizados para una publicación enviada por un cliente al broker es de 357,98 ms, es un proceso con un tiempo de ejecución muy estable (con una desviación típica de 0,26 ms). La mayor parte del tiempo es utilizado para la realización de las funciones del broker, 143,13 ms para el cliente y 214,85 ms para el broker.

En la tabla 6.4 podemos ver un resumen de los tiempos obtenidos en esta prueba.

Tabla 6.4: Resumen tiempos Publicación $C_x \Rightarrow B$

Función	Lugar de ejecución	Media (ms)	σ (ms)
Crear *Payload*	Cliente	61,05	0,09
Crear *Topic*	Cliente	61,03	0,09
Leer *Payload*	Broker	79,86	0,12
Leer *Topic*	Broker	80,00	0,17
Crear PUBREC	Broker	54,99	0,11
Check PUBREC	Cliente	21,05	0,11
	Total	357,98	0,26

6.1.7. Suscripción

Para el cálculo de tiempos en una suscripción o anulación de suscripción se realiza el siguiente proceso de llamadas a las funciones de los applets correspondientes.

- "create_topic_sub" (Smart Card cliente). Cifra el *topic* al que el cliente se suscribe o bien cancela su suscripción (246 bytes).
- "read_topic_sub" (Smart Card broker). Descifra el *topic* recibido (246 bytes).
- "create_ack_sub" (Smart Card broker). Genera el cifrado a enviar en un mensaje SUBACK o UNSUBACK.
- "check_ack_sub" (Smart Card cliente). Descifra dato cifrado recibido por un mensaje SUBACK o UNSUBACK y comprueba su validez.

Para el estudio de estos tiempos se han realizado 100 iteraciones del proceso de autenticación, los datos obtenidos se almacenan en los ficheros "subB.csv" y "subC.csv".

Podemos ver los resultados de este estudio en la siguiente figura (fig. 6.5).

Como nos muestran las gráficas de los tiempos obtenidos, la media de todos los procesos criptográficos utilizados para un proceso de suscripción es de 202,55 ms, es

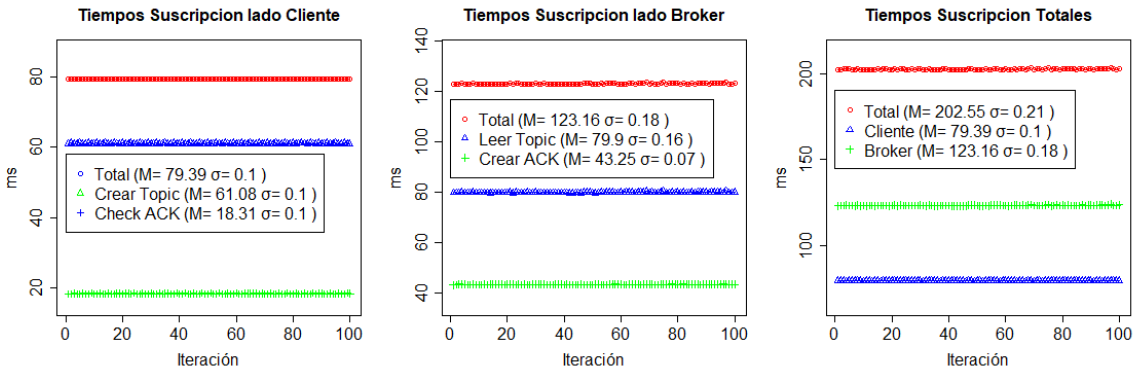


Figura 6.5: Tiempos suscripción o anulación de suscripción.

un proceso con un tiempo de ejecución muy estable (con una desviación típica de 0,21 ms). La mayor parte del tiempo es utilizado para la realización de las funciones del broker, 79,39 ms para el cliente y 123,16 para el broker.

En la tabla 6.5 podemos ver un resumen de los tiempos obtenidos en esta prueba.

Tabla 6.5: Resumen tiempos Suscripción o anulación de Suscripción

Función	Lugar de ejecución	Media (ms)	σ (ms)
Crear *Topic*	Cliente	61,08	0,01
Leer *Topic*	Broker	79,9	0,16
Crear ACK	Broker	43,25	0,07
Check ACK	Cliente	18,31	0,10
	Total	202,55	0,21

6.1.8. Tiempo de Ejecución vs Tiempo de Comunicación

Analizando con un osciloscopio la señal I/O de la Smart Card (C7) durante el proceso de envío del APDU, ejecución de la instrucción y recepción de la respuesta por parte del μprocesador, podemos observar que el tiempo de comunicación es

en algunas de las instrucciones bastante significativo con respecto al tiempo de su ejecución.

Con el objeto de poder realizar esta medición, se activa una señal de salida en el μprocesador durante el proceso de envío del APDU, ejecución y recepción de la respuesta y se mide la señal I/O durante el periodo en el que la señal esta activa.

Para realizar este análisis se ha utilizado un osciloscopio PicoScope 2204A [d6]. El resultado de esta medida lo podemos observar en la figura 6.6, la señal de disparo se ha conectado a la sonda A del osciloscopio (señal azul) y la señal I/O se ha conectado a la sonda B del osciloscopio (señal roja).

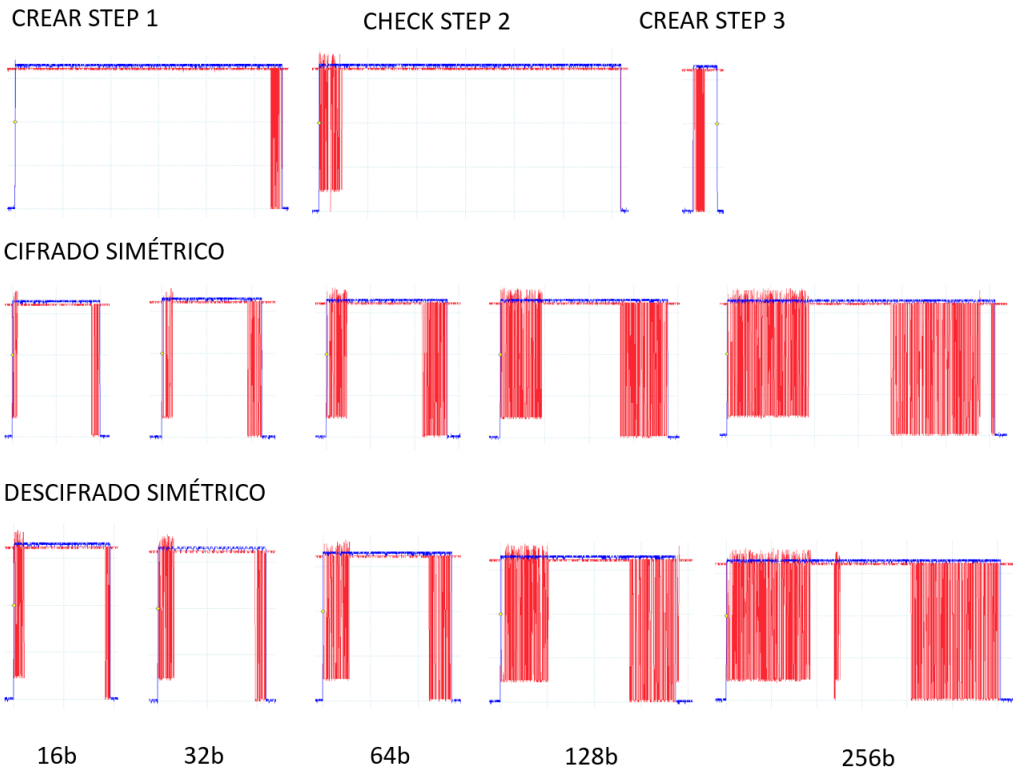


Figura 6.6: Análisis tiempos comunicación vs ejecución.

En el caso de las funciones con criptografía asimétrica podemos observar que el tiempo de comunicación es mucho menor que el de ejecución en el caso de las

instrucciones CREATE_AUTH_STEP1 y CHECK_AUTH_STEP2, no siendo así en el caso de la instrucción CREATE_AUTH_STEP3.

Por otro lado, en el caso del cifrado simétrico, podemos observar que el aumento del tiempo en las instrucciones de cifrado y descifrado simétrico correspondiente al tamaño de los elementos a tratar se justifica en gran medida por el tiempo de comunicación de estos datos, siendo menos significativo el aumento del tiempo de ejecución de la función.

Para analizar estas gráficas debemos tener en cuenta algunas consideraciones:

- La diferencia de amplitud entre la señal I/O cuando el μprocesador envía datos a cuando los recibe, se debe al diodo colocado en el TX del μprocesador (C.1) para poder realizar la conexión entre su UART (a dos hilos) y el UART de la Smart Card (a un hilo).
- En los cifrados simétricos los datos enviados y recibidos no son los mismos, ya que en el cifrado la Smart Card añade el UID y el tamaño del elemento a cifrar en dos bytes. Por ello vemos una clara diferencia en el tiempo de envío y recepción, apreciable sobre todo en los cifrados con pocos bytes.
- En las comunicaciones de más de 254 bytes vemos un espacio sin comunicación antes de finalizar la misma. Esto es debido a que supera la longitud máxima de comunicación de la Smart Card (IFSC). Por lo que en estas comunicaciones hay que realizar la segmentación de los mensajes. Este tiempo de segmentación de los mensajes es el tiempo sin comunicación que podemos observar en la gráfica. En este tiempo, en el caso de recepciones de más de 254 bytes por parte de la Smart Card, se realiza el traspaso de los datos recibidos en el primer mensaje en el buffer de recepción a las variables internas donde se trabajan durante la ejecución de la instrucción. En el caso del envío de respuestas de más de 254 bytes por parte de la Smart Card, este tiempo es el empleado para el paso de las variables de trabajo internas al buffer de salida. Este tiempo se tiene en cuenta como tiempo de ejecución de la función.

Las mediciones de tiempo de la ejecución de las instrucciones obtenidas en el proceso de medición de la potencia consumida (punto 6.2) se resumen en la tabla 6.6.

Tabla 6.6: Resumen tiempos de ejecución, sin comunicación.

Autenticación				
Instrucción			Media (ms)	σ (ms)
CREATE_AUTH_STEP1			535,82	3,897
CHECK_AUTH_STEP2			565,95	3,380
CREATE_AUTH_STEP3			30,05	0,036
Cifrado Simétrico				
Tamaño	Cifrado		Descifrado	
	Media (ms)	σ (ms)	Media (ms)	σ (ms)
16b	16,57	0,093	16,14	0,095
32b	16,71	0,090	16,28	0,098
64b	16,97	0,095	16,55	0,088
128b	17,55	0,094	17,21	0,091
256b	20,88	0,092	20,48	0,095
Creación PUBREC				
CREATE_PUBREC			18,09	0,024

6.1.9. Conclusiones

En todos los procesos analizados el tiempo de ejecución de las funciones criptográficas en el broker es mayor que en el cliente. La utilización de otro sistema criptográfico más rápido para la ejecución de estas funciones en el broker, como se recomienda en este documento, supondría una reducción drástica en el tiempo utilizado en todos los procesos.

El proceso de autenticación mutua tiene un tiempo de ejecución alto, este hecho debe tenerse en cuenta en la estrategia de comunicación de los diferentes dispositivos.

Si las comunicaciones de datos se realizan de forma periódica con una separación temporal mucho mayor que el tiempo de autenticación, se debe considerar la opción de la desconexión del cliente entre envíos. Esta desconexión debe realizarse sobre todo si el dispositivo cliente esta alimentado por una batería. En estos casos es habitual que entre los envíos el controlador pase a un modo de bajo consumo, lo que desconecta los interfaces de comunicación. En este modo de funcionamiento se puede habilitar una ventana de tiempo después del último dato enviado para la recepción

de datos en caso de ser necesario.

Por el contrario, si el dispositivo debe poder recibir datos en cualquier momento o bien el envío de los datos no está planificado, o está planificado con un tiempo no mucho mayor que el tiempo de autenticación, o bien el dato a enviar debe ser entregado con el menor retraso posible, debemos mantener la sesión abierta durante el funcionamiento del dispositivo.

6.2. Consumo Eléctrico

El análisis del consumo eléctrico derivado de la utilización de la Smart Card en un cliente, se considera un dato a tener en cuenta para la utilización de este esquema de seguridad en los dispositivos alimentados por baterías.

Para la realización de las mediciones de consumo de la Smart Card se ha utilizado el prototipo utilizado para la medición de tiempos. A este prototipo se le ha añadido una resistencia, de 4,3 Ω ± 0,1 %, entre el pin GND de la Smart Card (C5) y el pin GND del μprocesador (fig. 6.7). De modo que midiendo la caída de tensión en esta resistencia (Sonda B del osciloscopio) podremos calcular la intensidad que circula por la Smart Card. Este esquema de medición es el empleado en diferentes investigaciones previas [59], [72].

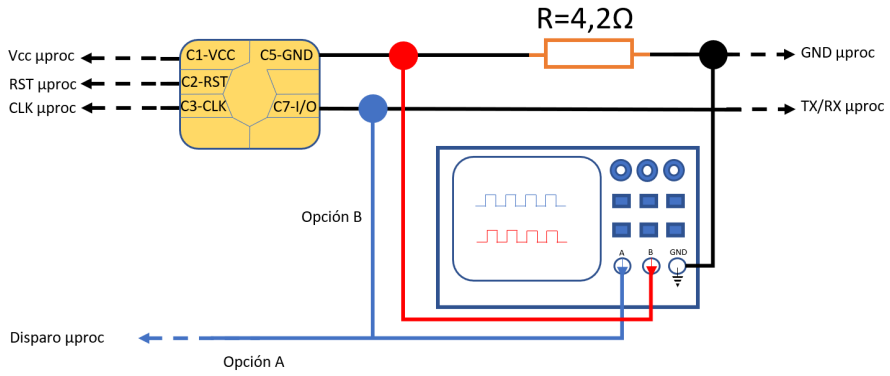


Figura 6.7: Conexionado mediciones (esquema en C.4).

El aparato de medida utilizado es el osciloscopio PicoScope 2204A [d6]. Este osciloscopio tiene una precisión en su medida de tensión continua de un ±3 % del total de escala ±200μV.

Este esquema de medición se basa en la premisa de que el total de la intensidad de entrada a la Smart Card será evacuada por el pin GND (C5). Esta consideración es aceptable excepto en el momento de la comunicación de datos a través del puerto I/O de la Smart Card (C7).

Según las especificaciones eléctricas de las Smart Card tipo B (3,3v) [53], las intensidades máximas que pueden circular por los pines RST (C2), CLK (C3) e I/O (C7) son las mostradas en la tabla 6.7.

Tabla 6.7: Intensidades por pines Smart Card según especificación

Pin	Estado	I_{max}
RST (C2)	V_{OH}	+20μA
	V_{OL}	-200μA
CLK (C3)	V_{OH}	+20μA
	V_{OL}	-200μA
I/O (C7)	V_{IH}	±20μA
	V_{IL}	+1mA
	V_{OH}	+20μA
	V_{OL}	-1mA

Se podrían considerar despreciable la intensidad del pin RST, ya que después de la ejecución del reset de la Smart Card esta señal se mantiene alta (+20μA). Y la de la señal CLK ya que al ser una señal cíclica, con un ciclo del 50 % alto y 50 % bajo, su consumo medio máximo en el caso más desfavorable será de -100μA. Pero en ningún caso se puede considerar despreciable la intensidad circulante por el pin I/O (C7) de la Smart Card durante la comunicación. Durante los periodos de no comunicación esta señal se mantiene como entrada alta (±20μA). Pero durante la comunicación, cuando la señal se encuentre en estado bajo, la intensidad de esta señal puede alcanzar valores de ±1mA, valor que no se puede considerar en ningún caso como despreciable.

Sería posible la medida de esta intensidad colocando una resistencia entre el pin I/O (C7) de la Smart Card y los pines RX y TX del μprocesador. Para ello sería necesaria la colocación de dos sondas para medir la caída de tensión en esta

resistencia. La tensión medida por estas dos sondas estaría en el rango de Vcc (3,3v), con lo que se debería colocar el rango de lectura en el osciloscopio en una escala de ±5V. Lo que llevaría a un error máximo de medida posible en cada sonda, según la especificación del osciloscopio utilizado, de 300,2 mV. Este rango de error es mucho mayor que la tensión esperada, colocando una resistencia de 4,3 Ω se espera una caída de tensión máxima de ±4,3 mV para ±1mA.

Por este motivo solo se ha podido realizar la medida de los procesos de ejecución de las instrucciones en la Smart Card, no pudiéndose medir el consumo en los procesos de intercambio de datos entre la Smart Card y el μprocesador.

Para la realización del disparo de almacenamiento de la señal de caida de tensión en R se han utilizado dos opciones.

- Opción A: En las instrucciones cuya duración obliga a la utilización de una escala de tiempo que impide la lectura correcta de los cambios en la señal I/O se utiliza una nueva señal de disparo generada por el μprocesador. Esta señal cambia a valor alto desde el envío del último dato a la Smart Card y hasta la recepción del primer byte de la respuesta.
- Opción B: En el resto de las instrucciones se utiliza como señal de disparo la señal I/O de la Smart Card. Los periodos de comunicación se desestiman en el tratamiento de datos posterior en el software RStudio.

Antes de realizar las pruebas de medición de consumo, se comprueba que la tensión de entrada a la Smart Card (C1), es en todo momento estable y con un valor medio de 3,299V y una desviación típica de 624,5μV (fig. 6.8). Consideraremos así la tensión Vcc=3,299V. Este valor será el utilizado para el cálculo de energía consumida.

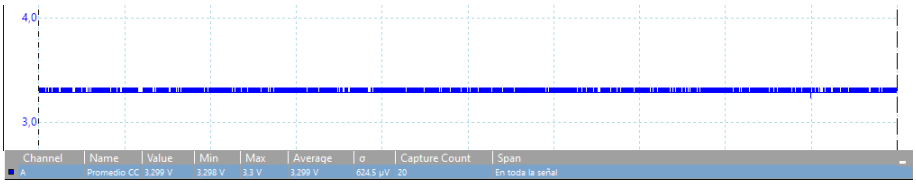


Figura 6.8: Medición C1-Vcc.

Los archivos csv con los datos obtenidos de cada medición, así como el proyecto RStudio que se ha realizado para realizar los cálculos, se pueden encontrar en el repositorio de esta tesis [w1].

Para todos los experimentos realizados se obtienen los siguientes valores:

- Tiempo mediciones (Milisegundos).
- Intensidad instantánea que circula por R en cada medición (Miliamperios). ($I = V_{sondaB}/R$).
- Intensidad Máxima (Miliamperios).
- Potencia consumida por la Smart Card en cada medida (Milivatios). ($P = (Vcc - V_{sondaB}) * I$).
- Energía consumida (Microvatios hora). ($E_w = P_{media} * tiempo\ (horas)$).

En todos los experimentos se han realizado 100 iteraciones.

6.2.1. Autenticación Mutua

En esta sección se analiza el consumo de las tres funciones ejecutadas por la Smart Card del cliente en la autenticación mutua.

CREATE_AUTH_STEP1 (fig. 6.9)

- Canal A: Señal de disparo. Gráfica azul.
- Canal B: Caída de tensión en R. Gráfica roja.
- Fondo de Escala Canal A: ±5V.
- Fondo de Escala Canal B: ±100mV.

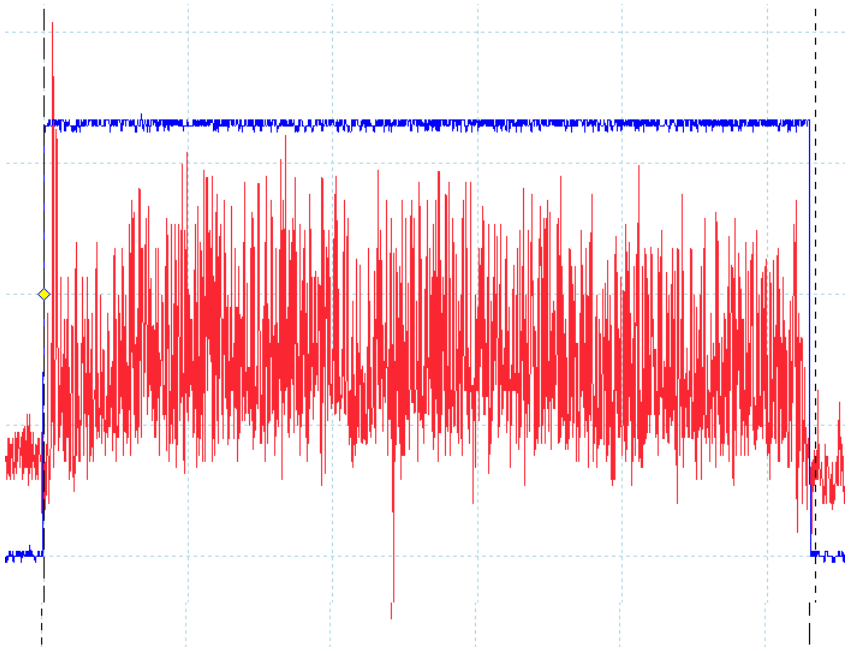


Figura 6.9: Medición CREATE_AUTH_STEP1.

El resumen de los datos de estas mediciones se describe en la tabla 6.8.

Tabla 6.8: Resumen Mediciones eléctricas CREATE_AUTH_STEP1

Medida	Max.	Min.	Media	σ	Unidades
Energía	3,355	3,184	3,277	0,037	μWh
I. Media	6,876	6,600	6,743	0,058	mA
I. Máxima	22,976	14,754	19,061	2,009	mA
Potencia Media	22,451	21,556	22,019	0,189	mW
Tiempo	547,22	527,56	535,83	3,891	ms
Muestras	1671	1611	1636,23	11,89	ud

CHECK_AUTH_STEP2 (fig. 6.10)

- Canal A: Señal de disparo. Gráfica azul.
- Canal B: Caída de tensión en R. Gráfica roja.
- Fondo de Escala Canal A: ±5V.
- Fondo de Escala Canal B: ±100mV.

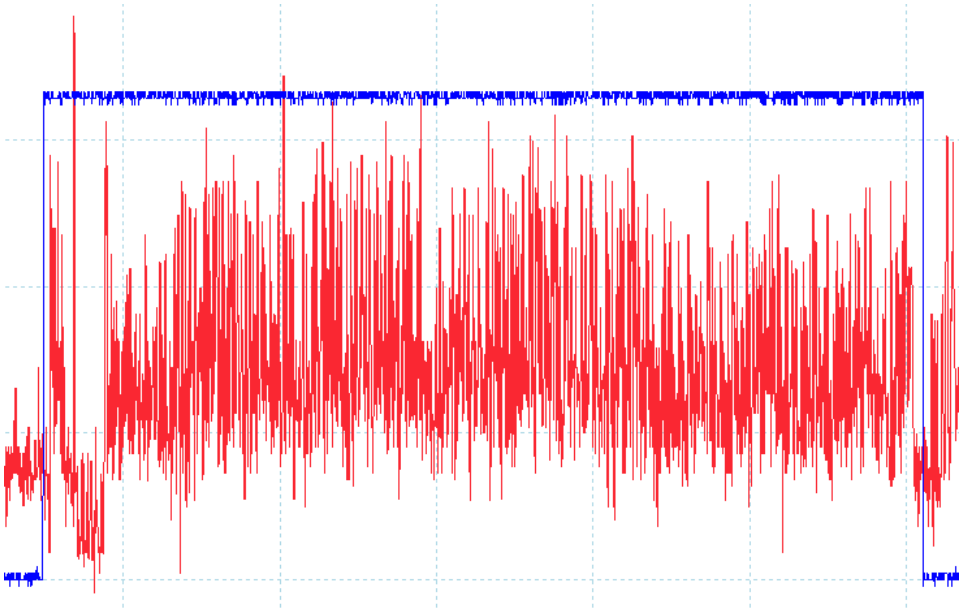


Figura 6.10: Medición CHECK_AUTH_STEP2.

Podemos ver el resumen de los datos de estas mediciones en la tabla 6.9.

Tabla 6.9: Resumen Mediciones eléctricas CHECK_AUTH_STEP2

Medida	Max.	Min.	Media	σ	Unidades
Energía	3,491	3,277	3,400	0,040	μWh
I. Media	6,814	6,412	6,624	0,068	mA
I. Máxima	22,765	15,808	19,129	1,939	mA
Potencia Media	22,245	20,944	21,629	0,222	mW
Tiempo	574,09	557,71	565,95	3,380	ms
Muestras	1753	1703	1728,15	10,31	ud

CREATE_AUTH_STEP3 (fig. 6.11)

- Canal A: Señal I/O. Gráfica azul.
- Canal B: Caída de tensión en R. Gráfica roja.
- Fondo de Escala Canal A: ±5V.
- Fondo de Escala Canal B: ±200mV.

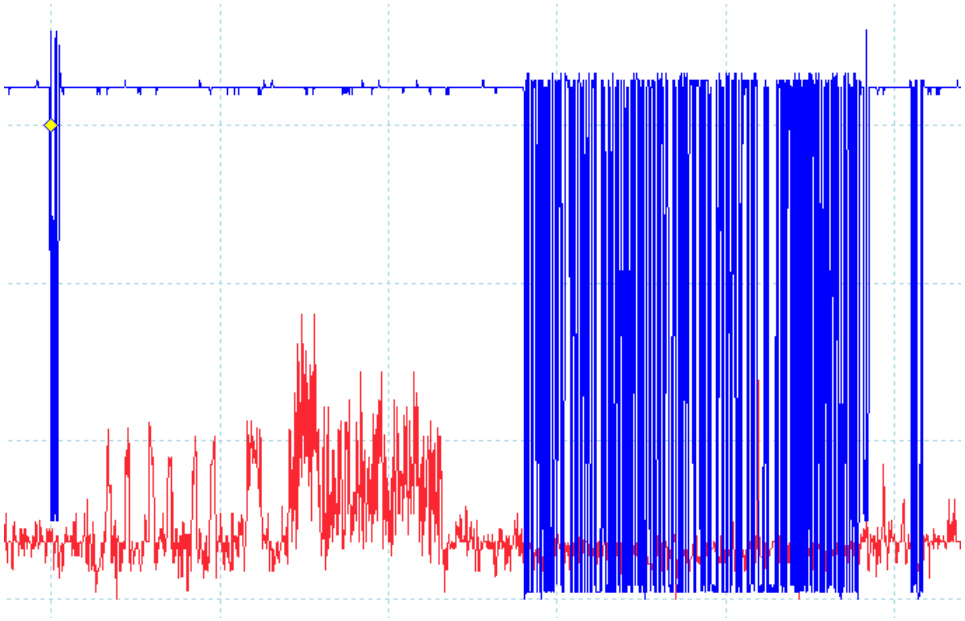


Figura 6.11: Medición CREATE_AUTH_STEP3.

Podemos ver el resumen de los datos de estas mediciones en la tabla 6.10.

Tabla 6.10: Resumen Mediciones eléctricas CREATE_AUTH_STEP3

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,134	0,126	0,130	0,0015	μWh
I. Media	4,930	4,633	4,777	0,057	mA
I. Máxima	19,280	13,831	16,112	1,016	mA
Potencia Media	16,126	15,164	15,630	0,186	mW
Tiempo	30,105	29,982	30,053	0,036	ms
Muestras	733	730	731,74	0,88	ud

6.2.2. Cifrado Simétrico

En esta sección se analiza el consumo del cifrado y descifrado simétrico en la Smart Card con las longitudes de datos a tratar de 16, 32, 64, 128 y 256 bytes.

- Canal A: Señal I/O. Gráfica azul.
- Canal B: Caída de tensión en R. Gráfica roja.
- Fondo de Escala Canal A: ±5V.
- Fondo de Escala Canal B: ±200mV.

Podemos ver la representación gráfica de las medidas con el osciloscopio para los cifrados y descifrados con criptografía simétrica con las diferentes longitudes de datos indicados fig. 6.12.

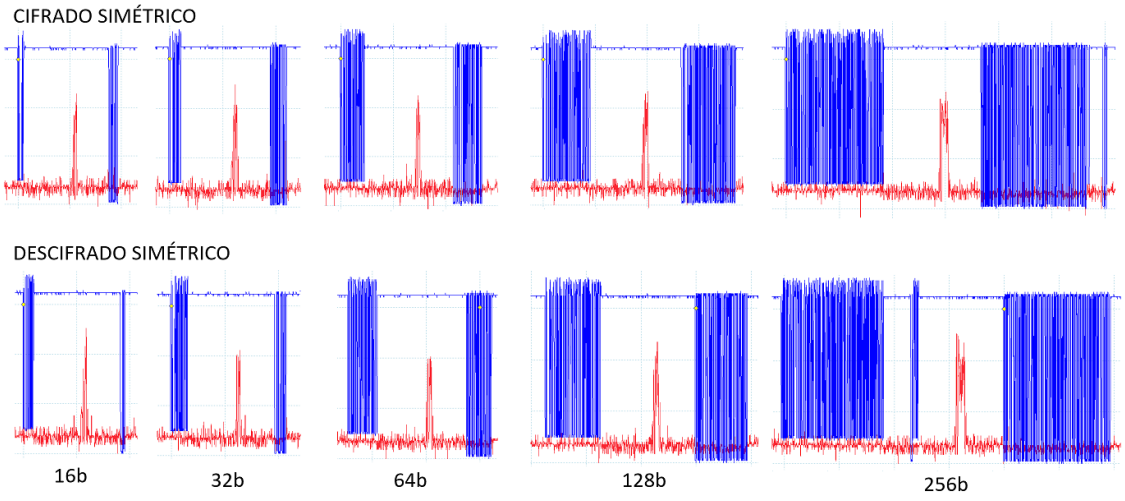


Figura 6.12: Figuras osciloscopio criptografía simétrica.

Para cada uno de los tamaños y cada una de las operaciones se han realizado 100 iteraciones y los datos obtenidos se muestran en la siguientes tablas: cifrado 16 bytes 6.11, cifrado 32 bytes 6.12, cifrado 64 bytes 6.13, cifrado 128 bytes 6.14, cifrado 256 bytes 6.15, descifrado 16 bytes 6.16, descifrado 32 bytes 6.17, descifrado 64 bytes 6.18, descifrado 128 bytes 6.19 y descifrado 256 bytes 6.20.

Tabla 6.11: Resumen mediciones eléctricas cifrado simétrico 16 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,056	0,052	0,054	0,0007	μWh
I. Media	3,657	3,496	3,573	0,038	mA
I. Máxima	28,501	18,022	22,712	1,730	mA
Potencia Media	11,980	11,460	11,710	0,123	mW
Tiempo	16,814	16,465	16,571	0,093	ms
Muestras	822	805	810,16	4,565	ud

Tabla 6.12: Resumen mediciones eléctricas cifrado simétrico 32 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,057	0,053	0,055	0,0008	μWh
I. Media	3,750	3,538	3,628	0,041	mA
I. Máxima	28,08	19,280	22,557	1,713	mA
Potencia Media	12,278	11,599	11,883	0,133	mW
Tiempo	16,936	16,609	16,710	0,090	ms
Muestras	828	812	816,92	4,403	ud

Tabla 6.13: Resumen mediciones eléctricas cifrado simétrico 64 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,059	0,055	0,057	0,0009	μWh
I. Media	3,839	3,570	3,706	0,049	mA
I. Máxima	28,501	19,281	23,123	1,965	mA
Potencia Media	12,568	11,698	12,139	0,158	mW
Tiempo	17,223	16,875	16,975	0,095	ms
Muestras	842	825	829,86	4,660	ud

Tabla 6.14: Resumen mediciones eléctricas cifrado simétrico 128 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,063	0,058	0,061	0,001	μWh
I. Media	3,968	3,675	3,812	0,051	mA
I. Máxima	28,501	20,118	23,291	1,572	mA
Potencia Media	12,985	12,035	12,476	0,167	mW
Tiempo	17,777	17,449	17,551	0,094	ms
Muestras	869	853	858,01	4,611	ud

Tabla 6.15: Resumen mediciones eléctricas cifrado simétrico 256 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,080	0,073	0,076	0,001	μWh
I. Media	4,192	3,875	4,027	0,057	mA
I. Máxima	28,501	19,699	22,268	1,526	mA
Potencia Media	13,698	12,678	13,167	0,183	mW
Tiempo	21,135	20,767	20,889	0,092	ms
Muestras	514	505	507,98	2,260	ud

Tabla 6.16: Resumen mediciones eléctricas descifrado simétrico 16 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,055	0,051	0,052	0,009	μWh
I. Media	3,698	3,489	3,574	0,045	mA
I. Máxima	28,081	19,280	22,721	1,651	mA
Potencia Media	12,112	11,436	11,713	0,146	mW
Tiempo	16,322	16,056	16,142	0,095	ms
Muestras	798	785	789,20	4,640	ud

Tabla 6.17: Resumen mediciones eléctricas descifrado simétrico 32 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,055	0,052	0,053	0,0008	μWh
I. Media	3,721	3,507	3,611	0,043	mA
I. Máxima	28,501	19,281	23,161	1,615	mA
Potencia Media	12,187	11,495	11,830	0,139	mW
Tiempo	16,589	16,200	16,281	0,098	ms
Muestras	811	792	796,00	4,807	ud

Tabla 6.18: Resumen mediciones eléctricas descifrado simétrico 64 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,058	0,054	0,055	0,0008	μWh
I. Media	3,803	3,577	3,681	0,043	mA
I. Máxima	26,824	20,118	22,960	1,538	mA
Potencia Media	12,452	11,720	12,056	0,140	mW
Tiempo	16,732	16,466	16,559	0,088	ms
Muestras	818	805	809,55	4,303	ud

Tabla 6.19: Resumen mediciones eléctricas descifrado simétrico 128 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,062	0,057	0,059	0,0009	μWh
I. Media	3,926	3,692	3,824	0,048	mA
I. Máxima	27,662	20,536	23,509	1,555	mA
Potencia Media	12,843	12,089	12,515	0,157	mW
Tiempo	17,408	17,121	17,218	0,092	ms
Muestras	851	837	841,74	4,491	ud

Tabla 6.20: Resumen mediciones eléctricas descifrado simétrico 256 bytes.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,077	0,073	0,074	0,009	μWh
I. Media	4,132	3,928	4,017	0,039	mA
I. Máxima	27,243	21,375	23,881	1,295	mA
Potencia Media	13,498	12,839	13,131	0,127	mW
Tiempo	20,685	20,377	20,487	0,095	ms
Muestras	1008	993	998,37	4,665	ud

6.2.3. Creación PUBREC (fig. 6.13)

En esta sección se analiza el consumo de la instrucción CREATE_PUBREC que, además de realizar el cifrado simétrico de los datos a enviar en la instrucción, genera dos nuevos números aleatorios que serán los nuevos números RN_{xn} y RN_{yn}.

- Canal A: Señal I/O. Gráfica azul.
- Canal B: Caída de tensión en R. Gráfica roja.
- Fondo de Escala Canal A: ±5V.
- Fondo de Escala Canal B: ±200mV.

Podemos ver el resumen de los datos de estas mediciones en la tabla 6.21.

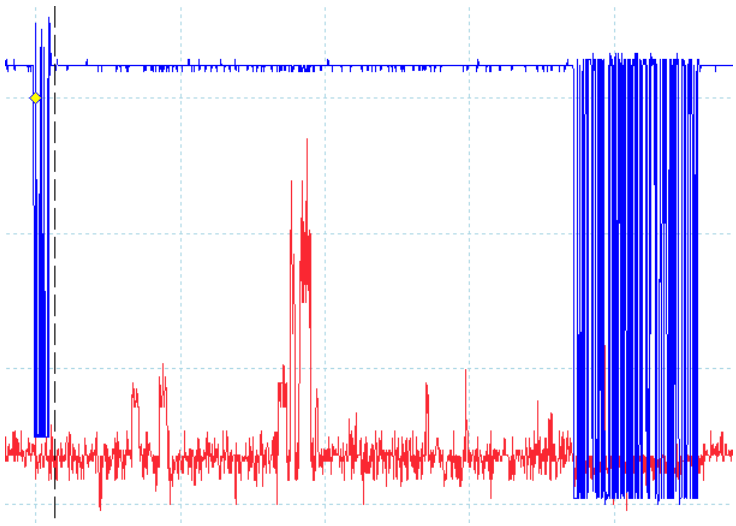


Figura 6.13: Figuras osciloscopio CREATE_PUBREC.

Tabla 6.21: Resumen mediciones eléctricas instrucción CREATE_PUBREC.

Medida	Max.	Min.	Media	σ	Unidades
Energía	0,065	0,061	0,062	0,005	μWh
I. Media	3,868	3,705	3,791	0,032	mA
I. Máxima	27,244	19,699	22,851	1,478	mA
Potencia Media	12,662	12,138	12,416	0,105	mW
Tiempo	18,142	18,042	18,09	0,024	ms
Muestras	887	882	884,31	1,169	ud

6.2.4. Consumo μprocesador a la espera de instrucción

Durante los periodos de espera de una nueva instrucción tanto μprocesador como el μprocesador criptográfico, así como otros componentes como la memoria RAM, etc... de la Smart Card permanecen activos. Esta actividad genera un consumo eléctrico que depende en gran medida de la frecuencia de reloj enviada a la Smart Card (pin C3-CLK) [73].

Con la frecuencia de 5 MHz enviada en este prototipo a la Smart Card se ha realizado la media de la intensidad circulante por la resistencia R durante 20 segundos,

tomando durante este tiempo 244500 valores de la caída de tensión en R. El valor de intensidad media es de 3,36 mA con una desviación típica de 0,021 mA.

6.2.5. Conclusiones

Este estudio de consumo eléctrico es únicamente una aproximación, debido a las limitaciones del aparato de medida utilizado para realizar la experimentación [d6]. Sin embargo, nos permite extraer algunas conclusiones validas y puede ser el inicio de una experimentación posterior en profundidad a realizar con equipos de medida más precisos.

El consumo energético más acusado durante la ejecución de instrucciones en la Smart Card se produce durante la ejecución de funciones de criptografía asimétrica con clave privada. Tanto en la instrucción CREATE_AUTH_STEP1, que se produce el cifrado con la clave privada del cliente, como en la instrucción CHECK_AUTH_STEP2 donde se produce un descifrado con la misma clave. Esto es correspondiente con otros estudios de consumo energético en Smart Card [74].

En la ejecución de criptografía simétrica podemos ver en todos los casos un pico de consumo (fig. 6.12) cuando se produce el cifrado o descifrado. El resto del tiempo de ejecución de la función, en la preparación de los datos a tratar y en su respuesta pasando estos datos de una zona de memoria a otra, el consumo es muy similar al consumo de la Smart Card acitva esperando una instrucción. Este consumo no se puede calcular debido a la imposibilidad de realizar el disparo de la medición del osciloscopio en un punto intermedio de la ejecución de la instrucción en la Smart Card y al error introducido por el dispositivo de medida.

En la ejecución de la función CREATE_PUBREC (fig. 6.13) podemos observar los dos picos generados por la creación de los nuevos números aleatorios antes del pico, con mucha más amplitud, del cifrado de los datos. Después de este pico se pueden observar dos picos mucho más pequeños, en tiempo y amplitud, que corresponden al almacenamiento de estos nuevos números como claves de cifrado.

El consumo energético derivado de la actividad de la Smart Card a la espera de una nueva instrucción es considerable, por lo que se deben seguir las recomendaciones dadas en 6.1.9 para el ahorro de energía cuando sea posible, desactivando la Smart

Card durante estos periodos. Lo más recomendable a nivel energético es quitar la alimentación de Vcc (C1) durante estos periodos. Si bien existen dos métodos para reducir el consumo sin tener que quitar la alimentación [53]. El primero es cambiar el estado de la señal RST (C2) a nivel bajo para pasar la Smart Card a Idle Mode, cuyo consumo máximo según la especificación será de 100μA a 1 Mhz. Otro modo de ahorro de energía disponible es Clock Stop Mode, alcanzado parando la señal de reloj suministrada a la Smart Card. En este modo el consumo máximo según la especificación es de 200μA.

Ambos métodos requieren de una inicialización completa de la Smart Card pero permiten alcanzar una disminución del consumo de energía sin tener que cortar Vcc. Cortar Vcc requiere que el μprocesador o el circuito de control, dispongan de una salida que pueda suministrar la corriente necesaria para el funcionamiento de la Smart Card.

La alimentación de la Smart Card, bien directa o bien a través de una salida, debe poder suministrar al menos la intensidad máxima requerida por los procesos de la Smart Card, más un margen de seguridad de al menos un 20 %. Según la experimentación dada 28,5mA +20 %.

Las gráficas obtenidas de todas las mediciones realizadas en esta sección se presentan en el Anexo F.

Capítulo 7

Conclusiones y Líneas Futuras

En este capítulo se tratan las conclusiones obtenidas en la realización de esta tesis así como las posibles líneas de investigación futuras que tomen como punto de partida este documento.

7.1. Conclusiones

En el desarrollo de esta tesis se ha presentado un esquema de seguridad para el protocolo MQTT. Se han incluido las modificaciones necesarias en el intercambio de mensajes propio del MQTT, sin realizar ninguna modificación en contra de la especificación del protocolo. Se ha aportado seguridad tanto en la autenticación de los participantes de la comunicación como para asegurar la integridad y ocultamiento de los datos intercambiados.

Este esquema de seguridad permite la utilización de este protocolo en un entorno industrial, en el que la seguridad en las comunicaciones es pieza fundamental para posibilitar su utilización.

De esta manera se ha presentado para su utilización, tanto en futuras investigaciones como en un entorno productivo real, un nuevo esquema de seguridad denominado MQTT-SCACAUTH, que permite la utilización del protocolo MQTT en el entorno industrial.

Para alcanzar los objetivos de esta tesis se han presentado tanto desde un punto de vista teórico como practico los procesos criptográficos necesarios.

La seguridad en la autenticación se ha realizado por medio de criptografía asimétrica, se ha utilizado en los prototipos de demostración el algoritmo RSA, este algoritmo es el más utilizado en la actualidad en los cifrados con claves asimétricas. Para la comprobación de la integridad de los datos intercambiados, así como para su ocultación ante posibles atacantes del sistema, se ha utilizado criptografía simétrica, mucho más rápida en su ejecución. En concreto, en los diferentes prototipos presentados en este documento se ha utilizado el algoritmo de cifrado simétrico AES.

Para la integración de este esquema de seguridad nos hemos enfrentado al hecho de que muchos de los controladores que tienen que ser comunicados en el entorno del IIoT no tienen capacidad de utilización de algoritmos criptográficos, bien por sus bajos recursos o bien por ser dispositivos heredados en la industria, que no han sido concebidos para la utilización de algoritmos de cifrado, como los PLCs. Con el fin de solventar este inconveniente se ha introducido el hardware necesario para la ejecución de los algoritmos criptográficos utilizados. Como hardware capaz de ejecutar estos algoritmos se han utilizado Smart Card criptográficas. En este documento se ha

desarrollado el software necesario a cargar en estos dispositivos, así como para su comunicación con los diferentes tipos de controladores existentes en el ámbito del IIoT.

Las Smart Card criptográficas seleccionadas integran la máquina virtual java JCVM, por lo que el software que se ha desarrollado para las Smart Card en esta tesis ha sido programado en JavaCard. De este modo es fácilmente trasladable a cualquier Smart Card que integre esta máquina virtual, sin importar su fabricante. Esto ayuda a la utilización de este esquema de seguridad en futuras investigaciones o entornos productivos sin necesidad de utilizar las mismas Smart Card utilizadas en esta tesis.

Una vez se ha realizado la propuesta teórica de este nuevo sistema de seguridad para el protocolo MQTT se han desarrollado diferentes prototipos que cubren todas las partes del sistema. Se ha presentado así una demostración real de la utilización de este esquema de seguridad, con un broker MQTT modificado para poder tratar con este esquema de seguridad y con diferentes clientes que pueden ser generadores o receptores de datos. Los controladores de estos clientes se han basado en un μprocesador, un PLC y un μPC, se ha cubierto así una amplia gama de dispositivos utilizados en la industria.

Se ha llevado a cabo la experimentación necesaria para comprobar los tiempos añadidos por la utilización de este esquema de seguridad. Se ha efectuado un estudio de los tiempos de envío de comandos a la Smart Card, ejecución de las instrucciones y recepción de las respuestas por parte de los controladores para todas las instrucciones a ejecutar en las Smart Card. Se han obtenido unos resultados que nos permiten conocer el tiempo añadido a la comunicación de los datos por el hecho de introducir este sistema de seguridad. Esto permitirá conocer a los posibles usuarios de este esquema de seguridad si es adecuado para sus necesidades o no lo es. Se ha considerado que estos tiempos son aceptables para comunicaciones de datos de funcionamiento a los sistemas de toma de decisiones, en ningún caso se ha pretendido que este sistema sea válido para su utilización en comunicaciones que requieran la toma de datos en tiempo real o con tiempo de ciclo de adquisición asegurado. Para estos fines, propios de la automatización, se deben seguir utilizando los protocolos deterministas ya utilizados en las redes de automatización.

En este documento se han realizado las experimentaciones que han sido posibles, con los aparatos de medida de los que se dispone, para el estudio del consumo

energético derivado de la utilización de las Smart Card en los clientes. Este estudio nos ha aportado los datos para valorar el aumento necesario en la capacidad de carga de las baterías de dispositivos que vayan a utilizar este sistema de seguridad y deban ser alimentados mediante baterías. El estudio de consumo realizado, si bien no se considera preciso por las limitaciones del dispositivo de medida utilizado, permite el estudio de la viabilidad de la utilización de este esquema de seguridad en diferentes dispositivos.

7.2. Líneas futuras de investigación

Existen muchas líneas de investigación que quedan abiertas una vez finalizada esta tesis.

En diversas publicaciones relacionadas con la criptografía asimétrica aparecidas en los últimos años se demuestra que la criptografía basada en curvas elípticas tiene una seguridad similar a la basada en el algoritmo RSA con la utilización de longitudes de clave mucho menores [75]-[77]. Los algoritmos basados en curvas elípticas permiten mejorar los tiempos de ejecución de los cifrados y descifrados con pares de claves público-privadas, así como en el consumo energético en su ejecución, para un nivel de seguridad similar al algoritmo RSA. Por ello, una futura línea de investigación es la utilización de este tipo de algoritmos en el esquema de seguridad propuesto en esta tesis en sustitución del algoritmo RSA.

Las funciones relacionadas con el cifrado con curvas elípticas disponibles en JCVM permiten su utilización para la firma (generación de la función hash de los datos a firmar y cifrado con su clave privada del resultado) y su verificación, pero no permiten la utilización de sus primitivas para el cifrado y descifrado de datos. Sin embargo, existe un proyecto denominado JCMathLib [78] que posibilita la utilización de estas primitivas para el cifrado y descifrado de datos. Es este un buen punto de partida para la línea de investigación que pretenda sustituir el cifrado con RSA por algún algoritmo basado en curvas elípticas.

De igual modo en esta tesis se ha utilizado como algoritmo de cifrado simétrico el AES, pero existen multitud de algoritmos simétricos presentes en diversas investigaciones [79]-[81], una futura línea de investigación podría realizar un estudio de

estos algoritmos para decidir el mejor algoritmo simétrico a introducir en el esquema de seguridad propuesto.

Como hardware criptográfico para los clientes en esta tesis se introducen Smart Card. En este momento este hardware es de muy fácil acceso y para un prototipado rápido es de fácil integración con los dispositivos utilizados en los prototipos generados en esta tesis. Sin embargo existen en la actualidad diversos hardwares criptográficos ya disponibles, así como en proceso de investigación que podrían tener cabida como sustitutos de las Smart Card utilizadas en esta tesis [82]-[84]. Esta es otra línea de investigación que queda abierta una vez finalizada esta tesis.

El hardware criptográfico utilizado en el prototipo de broker realizado en esta tesis es una Smart Card. Se ha utilizado este hardware por la dificultad, durante el desarrollo de esta tesis, de acceder a otro tipo de hardware más adecuado para su utilización en el lado del broker. Queda abierta por tanto una línea de investigación para el testeo de diferentes hardwares criptográficos a utilizar por parte del broker.

El prototipo del broker creado en esta tesis se ha basado en el broker mosquitto, este broker no resuelve su utilización en un entorno distribuido con cargas balanceadas. Una futura línea de investigación debe crear un prototipo de broker que resuelva su utilización en entornos distribuidos.

Otra de las líneas de investigación que queda abierta a la finalización de esta tesis es el estudio preciso del consumo energético de la Smart Card del cliente. Con la utilización de aparatos de medida precisos y la realización del estudio energético con diferentes tipos de Smart Card.

Por último, una línea de investigación que complementaría esta investigación es la utilización de técnicas de generación automática de código para la generación de los módulos de software, para la comunicación con este esquema de seguridad, en los diferentes entornos industriales que lo puedan utilizar. Esto permitiría una rápida implantación en los diferentes clientes y mejoraría la aceptación por parte de la industria del esquema de seguridad propuesto.

Bibliografía

Referencias

[1] Y. Cheng, R. Matthiesen, S. Farooq, J. Johansen, H. Hu y L. Ma, «The evolution of investment patterns on advanced manufacturing technology (AMT) in manufacturing operations: A longitudinal analysis,» *International Journal of Production Economics*, vol. 203, págs. 239-253, 2018, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2018.06.019. dirección: http://www.sciencedirect.com/science/article/pii/S0925527318302639 (vid. pág. 2).

[2] E. Ruschel, E. A. P. Santos y E. d. F. R. Loures, «Establishment of maintenance inspection intervals: an application of process mining techniques in manufacturing,» *Journal of Intelligent Manufacturing*, vol. 31, n.º 1, págs. 53-72, ene. de 2020. DOI: 10.1007/s10845-018-1434-7. dirección: https://doi.org/10.1007/s10845-018-1434-7 (vid. pág. 2).

[3] P. Lara, M. Sánchez y J. Villalobos, «OT Modeling: The Enterprise Beyond IT,» *Business & Information Systems Engineering*, vol. 61, DOI: 10.1007/s12599-018-0543-3. dirección: https://doi.org/10.1007/s12599-018-0543-3 (vid. pág. 2).

[4] C. Cronin, A. Conway y J. Walsh, «Flexible manufacturing systems using IIoT in the automotive sector,» en *Procedia Manufacturing*, vol. 38, Elsevier B.V., ene. de 2019, págs. 1652-1659. DOI: 10.1016/j.promfg.2020.01.119 (vid. pág. 2).

[5] J.-R. Jiang, «An improved cyber-physical systems architecture for Industry 4.0 smart factories,» *Advances in Mechanical Engineering*, vol. 10, n.º 6, jun. de 2018, ISSN: 1687-8140. DOI: 10.1177/1687814018784192. dirección: http://journals.sagepub.com/doi/10.1177/1687814018784192 (vid. pág. 3).

[6] «Survey: Manufacturers Behind the IIoT Curve,» English, *Manufacturing Engineering*, vol. 163, n.º 4, págs. 21-22, oct. de 2019, Nombre - Software AG; Copyright - Copyright SME Oct 2019; Última actualización - 2021-02-14, ISSN: 03610853. dirección: https://www-proquest-com.ezproxy.uned.es/trade-journals/survey-manufacturers-behind-iiot-curve/docview/2313897487/se-2?accountid=14609 (vid. pág. 3).

[7] S. Mantravadi, R. Schnyder, C. Moller y T. D. Brunoe, «Securing IT/OT Links for Low Power IIoT Devices: Design Considerations for Industry 4.0,» *IEEE Access*, vol. 8, págs. 200 305-200 321, nov. de 2020, ISSN: 2169-3536. DOI: 10.1109/access.2020.3035963 (vid. pág. 3).

[8] G. Spathoulas y S. Katsikas, «Towards a Secure Industrial Internet of Things,» en. mayo de 2019, págs. 29-45, ISBN: 978-3-030-05733-6. DOI: 10.1007/978-3-030-12330-7_2 (vid. págs. 4, 11).

[9] M. Al-Hawawreh y E. Sitnikova, «Developing a Security Testbed for Industrial Internet of Things,» *IEEE Internet of Things Journal*, vol. 8, n.º 7, págs. 5558-5573, 2021. DOI: 10.1109/JIOT.2020.3032093 (vid. págs. 4, 11).

[10] B. Mishra y A. Kertesz, «The Use of MQTT in M2M and IoT Systems: A Survey,» *IEEE Access*, vol. 8, págs. 201 071-201 086, 2020. DOI: 10.1109/ACCESS.2020.3035849 (vid. pág. 5).

[11] M. Amoretti, R. Pecori, Y. Protskaya, L. Veltri y F. Zanichelli, «A Scalable and Secure Publish/Subscribe-based Framework for Industrial IoT,» *IEEE Transactions on Industrial Informatics*, págs. 1-1, 2020. DOI: 10.1109/TII.2020.3017227 (vid. pág. 5).

[12] O. Ercan y G. Samet, «Literature review of Industry 4.0 and related technologies,» English, *Journal of Intelligent Manufacturing*, vol. 31, n.º 1, págs. 127-182, ene. de 2020, Copyright - Journal of Intelligent Manufacturing is a copyright of Springer, (2018). All Rights Reserved; Última actualización - 2020-11-17. dirección: https://search-proquest-com.ezproxy.uned.es/scholarly-journals/literature-review-industry-4-0-related/docview/2075215008/se-2?accountid=14609 (vid. pág. 9).

[13] S. Madakam y T. Uchiya, «Industrial Internet of Things (IIoT): Principles, Processes and Protocols,» en *The Internet of Things in the Industrial Sector: Security and Device Connectivity, Smart Environments, and Industry 4.0*, Z. Mahmood, ed. Cham: Springer International Publishing, 2019, págs. 35-53,

ISBN: 978-3-030-24892-5. DOI: 10.1007/978-3-030-24892-5_2. dirección: https://doi.org/10.1007/978-3-030-24892-5_2 (vid. pág. 9).

[14] S. Figueroa-Lorenzo, J. Añorga y S. Arrizabalaga, «A Survey of IIoT Protocols: A Measure of Vulnerability Risk Analysis Based on CVSS,» *ACM Comput. Surv.*, vol. 53, n.º 2, abr. de 2020, ISSN: 0360-0300. DOI: 10.1145/3381038. dirección: https://doi.org/10.1145/3381038 (vid. pág. 9).

[15] S. Jaloudi, «Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study,» *Future Internet*, vol. 11, mar. de 2019. DOI: 10.3390/fi11030066 (vid. pág. 10).

[16] L. Ma y Z. Xiu, «Industrial Internet of Things Multi-Protocol Convergence Gateway Research and Experiment,» en *2020 39th Chinese Control Conference (CCC)*, 2020, págs. 5155-5160. DOI: 10.23919/CCC50068.2020.9189200 (vid. pág. 10).

[17] R. Ala-Laurinaho, J. Autiosalo y K. Tammi, «Open Sensor Manager for IIoT,» *Journal of Sensor and Actuator Networks*, vol. 9, n.º 2, pág. 30, 2020, Copyright - © 2020. This work is licensed under http://creativecommons.org/licenses/by/3.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Última actualización - 2020-06-24. dirección: https://search-proquest-com.ezproxy.uned.es/scholarly-journals/open-sensor-manager-iiot/docview/2416717974/se-2?accountid=14609 (vid. pág. 10).

[18] E. Barrientos-Avendaño, Y. Areniz-Arevalo, L. Coronel-Rojas, F. Cuesta-Quintero y D. Rico-Bautista, «Modelo de incursión en la industria 4.0 aplicado a la compañía alimenticia tu pan gourmet SAS: Estrategia para el renacer en la pandemia ocasionada por COVID-19 (SARS-CoV-2),» Spanish, *Revista Ibérica de Sistemas e Tecnologias de Informação*, págs. 436-449, sep. de 2020. dirección: https://search-proquest-com.ezproxy.uned.es/scholarly-journals/modelo-de-incursi%C3%B3n-en-la-industria-4-0-aplicado/docview/2452330256/se-2?accountid=14609 (vid. pág. 11).

[19] Y. Cohen, H. Naseraldin, A. Chaudhuri y F. Pilati, «Assembly systems in Industry 4.0 era: a road map to understand Assembly 4.0,» *International Journal of Advanced Manufacturing Technology*, vol. 105, n.º 9, págs. 4037-4054, dic. de 2019, ISSN: 14333015. DOI: 10.1007/s00170-019-04203-1. dirección: https://doi.org/10.1007/s00170-019-04203-1 (vid. pág. 11).

[20] C. Paniagua y J. Delsing, «Industrial Frameworks for Internet of Things: A Survey,» *IEEE Systems Journal*, vol. 15, n.º 1, págs. 1149-1159, 2021. DOI: 10.1109/JSYST.2020.2993323 (vid. pág. 11).

[21] S. J. Kim, J. Cho, C. Lee y T. Shon, «Smart seed selection-based effective black box fuzzing for IIoT protocol,» *Journal of Supercomputing*, vol. 76, n.º 12, págs. 10 140-10 154, dic. de 2020, ISSN: 15730484. DOI: 10.1007/s11227-020-03245-7. dirección: https://doi.org/10.1007/s11227-020-03245-7 (vid. pág. 11).

[22] N. O. Tippenhauer, «Design and Realization of Testbeds for Security Research in the Industrial Internet of Things,» en *Security and Privacy Trends in the Industrial Internet of Things*, C. Alcaraz, ed. Cham: Springer International Publishing, 2019, págs. 287-310, ISBN: 978-3-030-12330-7. DOI: 10.1007/978-3-030-12330-7_14. dirección: https://doi.org/10.1007/978-3-030-12330-7_14 (vid. pág. 11).

[23] J. Gardiner, B. Craggs, B. Green y A. Rashid, «Oops I Did It Again: Further Adventures in the Land of ICS Security Testbeds,» en *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, ép. CPS-SPC'19, London, United Kingdom: Association for Computing Machinery, 2019, págs. 75-86, ISBN: 9781450368315. DOI: 10.1145/3338499.3357355. dirección: https://doi-org.ezproxy.uned.es/10.1145/3338499.3357355 (vid. pág. 11).

[24] R. Auliya, R.-K. Sheu, D. Liang y W.-J. Wang, «IIoT Testbed: A DDS-Based Emulation Tool for Industrial IoT Applications,» jun. de 2018, págs. 1-4. DOI: 10.1109/ICSSE.2018.8520091 (vid. pág. 11).

[25] S. G. Abbas, F. Hashmat y G. A. Shah, «A Multi-layer Industrial-IoT Attack Taxonomy: Layers, Dimensions, Techniques and Application,» en *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, págs. 1820-1825. DOI: 10.1109/TrustCom50675.2020.00249 (vid. pág. 12).

[26] D. S. Gupta, S. H. Islam, M. S. Obaidat, P. Vijayakumar, N. Kumar e Y. Park, «A Provably Secure and Lightweight Identity-Based Two-Party Authenticated Key Agreement Protocol for IIoT Environments,» *IEEE Systems Journal*, vol. 15, n.º 2, págs. 1732-1741, 2021. DOI: 10.1109/JSYST.2020.3004551 (vid. pág. 12).

[27] A. Shamir, «Identity-Based Cryptosystems and Signature Schemes,» en *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 196 LNCS, Springer Verlag, 1985, págs. 47-53, ISBN: 9783540156581. DOI: 10.1007/3-540-39568-7_5. dirección: https://link.springer.com/chapter/10.1007/3-540-39568-7_5 (vid. pág. 12).

[28] Y. Li, Q. Cheng y W. Shi, «Security Analysis of a Lightweight Identity-Based Two-Party Authenticated Key Agreement Protocol for IIoT Environments,» English, *Security and Communication Networks*, vol. 2021, 2021. dirección: https://www-proquest-com.ezproxy.uned.es/scholarly-journals/security-analysis-lightweight-identity-based-two/docview/2497885095/se-2?accountid=14609 (vid. pág. 12).

[29] J. Leevinson, V. Vijayaraghavan y M. Dammodaran, «Blockchain Mechanisms as Security-Enabler for Industrial IoT Applications,» en. ago. de 2019, págs. 145-162, ISBN: 978-3-030-24891-8. DOI: 10.1007/978-3-030-24892-5_7 (vid. pág. 12).

[30] H. Xiong, Y. Wu, C. Jin y S. Kumari, «Efficient and Privacy-Preserving Authentication Protocol for Heterogeneous Systems in IIoT,» *IEEE Internet of Things Journal*, vol. 7, n.º 12, págs. 11 713-11 724, 2020. DOI: 10.1109/JIOT.2020.2999510 (vid. pág. 12).

[31] H. Abdi Nasib Far, M. Bayat, A. Kumar Das, M. Fotouhi, S. M. Pournaghi y M. A. Doostari, «LAPTAS: lightweight anonymous privacy-preserving three-factor authentication scheme for WSN-based IIoT,» *Wireless Networks*, vol. 27, n.º 2, págs. 1389-1412, feb. de 2021, ISSN: 1572-8196. DOI: 10.1007/s11276-020-02523-9. dirección: https://doi.org/10.1007/s11276-020-02523-9 (vid. pág. 12).

[32] C. Patel y N. Doshi, «"A Novel MQTT Security framework In Generic IoT Model",» *Procedia Computer Science*, vol. 171, págs. 1399-1408, 2020, Third International Conference on Computing and Network Communications (CoCoNet'19), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.150. dirección: https://www.sciencedirect.com/science/article/pii/S1877050920311297 (vid. pág. 12).

[33] T. Dierks y E. Rescorla, «The Transport Layer Security (TLS) Protocol Version 1.2,» RFC, inf. téc., ago. de 2008. DOI: 10.17487/RFC5246. dirección:

https://www.rfc-editor.org/rfc/rfc5246.html (visitado 20-12-2020) (vid. págs. 14, 23).

[34] R. R. Pahlevi, P. Sukarno y B. Erfianto, «Implementation of Event-Based Dynamic Authentication on MQTT Protocol,» *Jurnal Rekayasa Elektrika*, vol. 15, n.º 2, 2019, ISSN: 1412-4785. DOI: 10.17529/jre.v15i2.13963 (vid. pág. 23).

[35] D. Hardt, «The OAuth 2.0 Authorization Framework,» Internet Engineering Task Force (IETF), inf. téc., oct. de 2012. dirección: https://www.rfc-editor.org/rfc/pdfrfc/rfc6749.txt.pdf (vid. pág. 23).

[36] C. Sharma y N. K. Gondhi, «Communication Protocol Stack for Constrained IoT Systems,» en *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2018, págs. 1-6. DOI: 10.1109/IoT-SIU.2018.8519904 (vid. pág. 23).

[37] T. keophilavong, Widyawan y M. N. Rizal, «Data Transmission in Machine to Machine Communication Protocols for Internet of Things Application: A Review,» en *2019 International Conference on Information and Communications Technology (ICOIACT)*, 2019, págs. 899-904. DOI: 10.1109/ICOIACT46704.2019.8938420 (vid. pág. 23).

[38] S. R. U. Kakakhel, T. Westerlund, M. Daneshtalab, Z. Zou, J. Plosila y H. Tenhunen, «A Qualitative Comparison Model for Application Layer IoT Protocols,» en *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, págs. 210-215. DOI: 10.1109/FMEC.2019.8795324 (vid. pág. 23).

[39] R. Kumar N.V. y M. Kumar P., «Survey on State of Art IoT Protocols and Applications,» en *2020 International Conference on Computational Intelligence for Smart Power System and Sustainable Energy (CISPSSE)*, 2020, págs. 1-3. DOI: 10.1109/CISPSSE49931.2020.9212227 (vid. pág. 23).

[40] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo y C. Yan, «Investigating Messaging Protocols for the Internet of Things (IoT),» *IEEE Access*, vol. 8, págs. 94 880-94 911, 2020. DOI: 10.1109/ACCESS.2020.2993363 (vid. pág. 23).

[41] R. De Prisco, A. De Santis y M. Mannetta, «Reducing costs in HSM-based data centers,» *Journal of High Speed Networks*, vol. 24, págs. 363-373, 2018, 4, ISSN: 1875-8940. DOI: 10.3233/JHS-180600. dirección: https://doi.org/10.3233/JHS-180600 (vid. pág. 25).

[42] M. Navid Bin Anwar, M. Hasan, M. Hasan, J. Z. Loren y S. M. Tanjim Hossain, «Comparative Study of Cryptography Algorithms and Its' Applications,» *International Journal of Computer Networks and Communications Security*, vol. 7, n.º 5, págs. 96-103, 2019. dirección: www.ijcncs.org (vid. pág. 25).

[43] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou y C. Manifavas, «A review of lightweight block ciphers,» *Journal of Cryptographic Engineering*, vol. 8, n.º 2, págs. 141-184, jun. de 2018, ISSN: 2190-8516. DOI: 10.1007/s13389-017-0160-y. dirección: https://doi.org/10.1007/s13389-017-0160-y (vid. pág. 26).

[44] M. Bahadori, M. R. Mali, O. Sarbishei, M. Atarodi y M. Sharifkhani, «A novel approach for secure and fast generation of RSA public and private keys on SmartCard,» en *Proceedings of the 8th IEEE International NEWCAS Conference 2010*, 2010, págs. 265-268. DOI: 10.1109/NEWCAS.2010.5603937 (vid. pág. 26).

[45] P. Grassi, G. M. y F. F., «Digital Identity Guidelines,» inf. téc., 2017. DOI: 10.6028/NIST.SP.800-63-3 (vid. pág. 26).

[46] A. S. Sadeq, R. Hassan, S. S. Al-rawi, A. M. Jubair y A. H. M. Aman, «A Qos Approach For Internet Of Things (Iot) Environment Using Mqtt Protocol,» en *2019 International Conference on Cybersecurity (ICoCSec)*, 2019, págs. 59-63. DOI: 10.1109/ICoCSec47621.2019.8971097 (vid. pág. 28).

[47] K. Bhat, D. Mahto y D. Yadav, «Comparison Analysis of AES-256, RSA-2048 and Four Dimensional Playfair Cipher Fused with Linear Feedback Shift Register,» 2017 (vid. pág. 35).

[48] M. Shuai, N. Yu, H. Wang y L. Xiong, «Anonymous authentication scheme for smart home environment with provable security,» *Computers and Security*, vol. 86, págs. 132-146, 2019, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.06.002. dirección: https://www.sciencedirect.com/science/article/pii/S0167404818313701 (vid. pág. 36).

[49] N. Emamdoost, M. S. Dousti y R. Jalili, *Statistical Disclosure: Improved, Extended, and Resisted*, 2017. arXiv: 1710.00101 [cs.CR] (vid. pág. 36).

[50] E. B. Sanjuan, I. A. Cardiel, J. A. Cerrada y C. Cerrada, «Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach,» *IEEE Access*, vol. 8, págs. 115 051-115 062, 2020. DOI: 10.1109/ACCESS.2020.3003998 (vid. págs. 47, 187).

[51] R. L. Rivest, A. Shamir y L. Adleman, «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,» *Commun. ACM*, vol. 21, n.º 2, págs. 120-126, feb. de 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342. dirección: https://doi.org/10.1145/359340.359342 (vid. pág. 53).

[52] T. Jamil, «The Rijndael algorithm,» *IEEE Potentials*, vol. 23, n.º 2, págs. 36-38, 2004. DOI: 10.1109/MP.2004.1289996 (vid. pág. 53).

[53] European Telecommunications Standards Institute, «Smart Cards: Smart Cards; UICC-Terminal Interface; Physical and logical characteristics (Release 13),» inf. téc., mayo de 2016. dirección: https://www.etsi.org/deliver/etsi_ts/102200_102299/102221/13.01.00_60/ts_102221v130100p.pdf (vid. págs. 74, 77, 149, 162).

[54] K. Mayes, «Performance Evaluation and Optimisation for Kyber on the MULTOS IoT Trust-Anchor,» en *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2020, págs. 1-8. DOI: 10.1109/SmartIoT49966.2020.00010 (vid. pág. 76).

[55] R. N. Akram y K. Markantonakis, «Smart cards: State-of-the-art to future directions,» en *IEEE International Symposium on Signal Processing and Information Technology*, 2013, págs. 000 154-000 162. DOI: 10.1109/ISSPIT.2013.6781871 (vid. pág. 76).

[56] *Security Policy*, nCipher Security Limited, oct. de 2019. dirección: https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp2764.pdf (vid. pág. 76).

[57] M. Randolph y W. Diehl, «Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman,» *Cryptography*, vol. 4, n.º 2, 2020, ISSN: 2410-387X. dirección: https://www.mdpi.com/2410-387X/4/2/15 (vid. pág. 76).

[58] L. Batina, Ł. Chmielewski, L. Papachristodoulou, P. Schwabe y M. Tunstall, «Online template attacks,» *Journal of Cryptographic Engineering*, vol. 9, n.º 1, págs. 21-36, abr. de 2019, ISSN: 2190-8516. DOI: 10.1007/s13389-017-0171-8. dirección: https://doi.org/10.1007/s13389-017-0171-8 (vid. pág. 76).

[59] H. J. Mahanta, A. K. Azad y A. K. Khan, «Power analysis attack: A vulnerability to smart card security,» en *2015 International Conference on Signal Processing and Communication Engineering Systems*, 2015, págs. 506-510. DOI: 10.1109/SPACES.2015.7058206 (vid. págs. 76, 148).

[60] S. Malhotra, A. Kumar y R. Dutta, «Effect of integrating IoT courses at the freshman level on learning attitude and behaviour in the classroom,» *Education and Information Technologies*, vol. 26, n.º 3, págs. 2607-2621, mayo de 2021, ISSN: 1573-7608. DOI: 10.1007/s10639-020-10376-0. dirección: https://doi.org/10.1007/s10639-020-10376-0 (vid. pág. 99).

[61] D. Eridani, E. D. Widianto, R. Septiana, E. Y. Indrasto, K. T. Martono y A. Fauzi, «Data Comparison of NFC PN532 on Wemos Dl and MKR1000 Board through MQTT Protocol,» en *2018 Third International Conference on Informatics and Computing (ICIC)*, 2018, págs. 1-5. DOI: 10.1109/IAC.2018.8780409 (vid. pág. 99).

[62] F. B. Alam y F. T. Zohura, «Automated IoT-based water quality measurement and analysis tool,» en *2020 IEEE Region 10 Symposium (TENSYMP)*, 2020, págs. 1523-1526. DOI: 10.1109/TENSYMP50017.2020.9230930 (vid. pág. 99).

[63] C. Sun, F. Zheng, G. Zhou y K. Guo, «Design and Implementation of Cloud-based Single-channel LoRa IIoT Gateway Using Raspberry Pi,» en *2020 39th Chinese Control Conference (CCC)*, 2020, págs. 5259-5263. DOI: 10.23919/CCC50068.2020.9189480 (vid. pág. 117).

[64] E. Palomar, C. Cruz, I. Bravo y A. Gardel, «Performance Evaluation of a Collaborative IoT Framework for Energy-Efficient Communities,» en *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2020, págs. 861-866. DOI: 10.1109/HPCC-SmartCity-DSS50907.2020.00114 (vid. pág. 117).

[65] M. Sălăgean y D. Zinca, «IoT Applications based on MQTT Protocol,» en *2020 International Symposium on Electronics and Telecommunications (ISETC)*, 2020, págs. 1-4. DOI: 10.1109/ISETC50328.2020.9301055 (vid. pág. 117).

[66] X. Shao, X. Kang, X. Wang y X. Yuan, «Design of special vehicle condition monitoring system based on J1939,» *Journal of Physics: Conference Series*, vol. 1549, pág. 032 092, jun. de 2020. DOI: 10.1088/1742-6596/1549/3/032092. dirección: https://doi.org/10.1088/1742-6596/1549/3/032092 (vid. pág. 121).

[67] S. Mukherjee, J. C. Van Etten, N. R. Samyukta, J. Walker, I. Ray e I. Ray, «TruckSTM: Runtime Realization of Operational State Transitions for Medium and Heavy Duty Vehicles,» *ACM Trans. Cyber-Phys. Syst.*, vol. 4, n.º 1, nov. de 2019, ISSN: 2378-962X. DOI: 10.1145/3300183. dirección: https://doi.org/10.1145/3300183 (vid. pág. 121).

[68] P. Killeen, B. Ding, I. Kiringa y T. Yeap, «IoT-based predictive maintenance for fleet management,» *Procedia Computer Science*, vol. 151, págs. 607-613, 2019, The 10th International Conference on Ambient Systems, Networks and Technologies (ANT 2019) / The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019) / Affiliated Workshops, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.04.184. dirección: https://www.sciencedirect.com/science/article/pii/S1877050919306519 (vid. pág. 121).

[69] M. Xiao, W. Wang, K. Wang, W. Zhang y H. Zhang, «Fault Diagnosis of High-Power Tractor Engine Based on Competitive Multiswarm Cooperative Particle Swarm Optimizer Algorithm,» English, *Shock and Vibration*, vol. 2020, 2020, ISSN: 10709622. dirección: https://www-proquest-com.ezproxy.uned.es/scholarly-journals/fault-diagnosis-high-power-tractor-engine-based/docview/2434421536/se-2?accountid=14609 (vid. pág. 121).

[70] R. A. Light, «Mosquitto: server and client implementation of the MQTT protocol,» *Journal of Open Source Software*, vol. 2, n.º 13, pág. 265, 2017. DOI: 10.21105/joss.00265. dirección: https://doi.org/10.21105/joss.00265 (vid. pág. 126).

[71] D. De Almeida Braga, P.-A. Fouque y M. Sabt, «The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10,» *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, n.º 3, págs. 196-218, jun. de 2020. DOI: 10.13154/tches.v2020.i3.196-218. dirección: https://tches.iacr.org/index.php/TCHES/article/view/8588 (vid. pág. 136).

[72] N. Benhadjyoussef, M. Karmani y H. Mestiri, «Power Analysis for Smartcard's Authentication-Protocol,» en *2019 International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, 2019, págs. 268-272. DOI: 10.1109/ASET.2019.8870994 (vid. pág. 148).

[73] «Smart Card Commands,» en *Smart Card Handbook*. John Wiley & Sons, Ltd, 2010, cap. 11, págs. 353-419, ISBN: 9780470660911. DOI: https://doi.org/10.1002/9780470660911.ch11. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470660911.ch11 (vid. págs. 160, 209).

[74] L. Wenli y Z. Jitianbai, «The Power Consumption Analysis of Java Card on Different Cryptographic Algorithm,» en *2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, 2020, págs. 150-153. DOI: 10.1109/ICBAIE49996.2020.00038 (vid. pág. 161).

[75] F. Mallouli, A. Hellal, N. Sharief Saeed y F. Abdulraheem Alzahrani, «A Survey on Cryptography: Comparative Study between RSA vs ECC Algorithms, and RSA vs El-Gamal Algorithms,» en *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2019, págs. 173-176. DOI: 10.1109/CSCloud/EdgeCom.2019.00022 (vid. pág. 166).

[76] B. S. Gowda, «Implementation of Elliptic Curve Cryptography over a Server-Clie—nt network,» en *2020 5th International Conference on Devices, Circuits and Systems (ICDCS)*, 2020, págs. 116-119. DOI: 10.1109/ICDCS48716.2020.243562 (vid. pág. 166).

[77] A. Mullai y K. Mani, «Enhancing the security in RSA and elliptic curve cryptography based on addition chain using simplified Swarm Optimization and Particle Swarm Optimization for mobile devices,» *International Journal of Information Technology*, vol. 13, n.º 2, págs. 551-564, abr. de 2021, ISSN: 2511-2112. DOI: 10.1007/s41870-019-00413-8. dirección: https://doi.org/10.1007/s41870-019-00413-8 (vid. pág. 166).

[78] V. Mavroudis y P. Svenda, «JCMathLib: Wrapper Cryptographic Library for Transparent and Certifiable JavaCard Applets,» págs. 89-96, 2020 (vid. pág. 166).

[79] R. Geetha, T. Padmavathy, T. Thilagam y A. Lallithasree, «Tamilian Cryptography: An Efficient Hybrid Symmetric Key Encryption Algorithm,» *Wireless Personal Communications*, vol. 112, n.º 1, págs. 21-36, mayo de 2020, ISSN: 1572-834X. DOI: 10.1007/s11277-019-07013-6. dirección: https://doi.org/10.1007/s11277-019-07013-6 (vid. pág. 166).

[80] J. Rodríguez, B. Corredor y C. Suárez, «Genetic Operators Applied to Symmetric Cryptography,» *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. IP, pág. 1, dic. de 2019. DOI: 10.9781/ijimai.2019.07.006 (vid. pág. 166).

[81] D. A. F. Saraiva, V. Leithardt, D. D. Paula, A. S. Mendes, G. Villarrubia y P. Crocker, «PRISEC: Comparison of Symmetric Key Algorithms for IoT Devices,» *Sensors (Basel, Switzerland)*, vol. 19, 2019 (vid. pág. 166).

[82] L. Nan, X. Yang, X. Zeng, W. Li, Y. Du, Z. Dai y L. Chen, «A VLIW architecture stream cryptographic processor for information security,» *China Communications*, vol. 16, n.º 6, págs. 185-199, 2019. DOI: 10.23919/JCC.2019.06.015 (vid. pág. 167).

[83] A. Ibrahim, «Systolic Processor Core for Finite-Field Multiplication and Squaring in Cryptographic Processors of IoT Edge Devices,» *IEEE Internet of Things Journal*, págs. 1-1, 2021. DOI: 10.1109/JIOT.2021.3087274 (vid. pág. 167).

[84] X. Hu, X. Zheng, S. Zhang, W. Li, S. Cai y X. Xiong, «A High-Performance Elliptic Curve Cryptographic Processor of SM2 over GF(p),» *Electronics*, vol. 8, n.º 4, 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8040431. dirección: https://www.mdpi.com/2079-9292/8/4/431 (vid. pág. 167).

[85] E. Buetas, I. Abad, J. A. Cerrada y C. Cerrada, «A propagation breakdown management model for the industrial internet of things,» *Computers in Industry*, vol. 123, pág. 103 305, 2020, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2020.103305. dirección: http://www.sciencedirect.com/science/article/pii/S016636152030539X (vid. pág. 188).

Estándares

[s1] «MQTT Version 5.0,» OASIS, Standard, dic. de 2017. dirección: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html (visitado 17-01-2021) (vid. págs. 5, 14, 20, 21, 22, 23, 48, 49, 129).

[s2] «MQTT Version 3.1.1,» OASIS, Standard, oct. de 2014. dirección: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (visitado 17-01-2021) (vid. págs. 14, 23, 49).

[s3] «MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2,» OASIS, Standard, 2013. dirección: https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf (visitado 17-01-2021) (vid. pág. 14).

[s4] J. Sermersheim, «Lightweight Directory Access Protocol (LDAP): The Protocol,» RFC, inf. téc., jun. de 2006. DOI: 10.17487/RFC4511. dirección: https://www.rfc-editor.org/rfc/rfc4511.html (vid. pág. 23).

[s5] S. Josefsson, «The Base16, Base32, and Base64 Data Encodings,» RFC Editor, RFC 4648, oct. de 2006, págs. 1-18. dirección: https://www.rfc-editor.org/rfc/rfc4648.txt (vid. pág. 28).

[s6] R. S. A. Laboratories, «PKCS #1: RSA Cryptography Standard - v2.2,» inf. téc., 2012. dirección: https://tools.ietf.org/html/rfc8017 (vid. pág. 53).

[s7] The International Organization for Standardization (ISO), «Part 3: Cards with contacts — Electrical interface and transmission protocols,» inf. téc., nov. de 2006 (vid. págs. 74, 77, 78, 215).

[s8] The International Organization for Standardization (ISO), «Part 2: Cards with contacts — Dimensions and location of the contacts,» inf. téc., oct. de 2007 (vid. pág. 74).

[s9] The International Organization for Standardization (ISO), «Part 12: Cards with contacts — USB electrical interface and operating procedures,» inf. téc., oct. de 2005 (vid. pág. 75).

[s10] *Java Card Platform Virtual Machine Specification, Classic Edition Version 3.1*, ORACLE, feb. de 2021. dirección: `https://docs.oracle.com/javacard/3.1/related-docs/JCVMS/JCVMS.pdf` (vid. pág. 76).

[s11] SAE Standars, *Data Link Layer*, Society Of Automotive Engineers, 2010 (vid. pág. 120).

[s12] SAE Standars, *Vehicle Application Layer*, Society Of Automotive Engineers, 2016 (vid. pág. 121).

[s13] SAE Standars, *Data Link Layer-Diagnostics*, Society Of Automotive Engineers, 2016 (vid. pág. 122).

[s14] *GlobalPlatform Technology Card Specification Version 2.3.1*, GlobalPlatform, 2018. dirección: `https://globalplatform.org/wp-content/uploads/2018/05/GPC_CardSpecification_v2.3.1_PublicRelease_CC.pdf` (vid. pág. 136).

[s15] The International Organization for Standardization (ISO), «Part 4: Organization, security and commands for interchange,» inf. téc., mayo de 2020 (vid. pág. 211).

Hojas de Datos

[d1] *NXP secure microcontroller family SmartMX2*, 9397 750 17276, NXP, 2012. dirección: https://www.nxp.com/docs/en/brochure/75017516.pdf (vid. pág. 76).

[d2] Siemens, *6ES7212-1HE40-0XB0*. dirección: https://support.industry.siemens.com/cs/pd/229817?pdti=td&dl=es&lc=es-ES (visitado 29-05-2021) (vid. pág. 108).

[d3] Siemens, *6ES7241-1AH32-0XB0*. dirección: https://support.industry.siemens.com/cs/pd/75966?pdti=td&dl=es&lc=es-ES (visitado 29-05-2021) (vid. pág. 108).

[d4] Raspberry, *Raspberry Pi 3 Model B+*. dirección: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf (vid. pág. 117).

[d5] HID, *OMNIKEY 3021 USB Reader*. dirección: https://www.hidglobal.com/sites/default/files/resource_files/omnikey-3021-usb-reader-ds-en.pdf (visitado 30-05-2021) (vid. pág. 117).

[d6] PicoScope, *PicoScope 2000 Series Ultra-compact PC oscilloscopes*. dirección: https://www.picotech.com/download/datasheets/picoscope-2000-series-data-sheet-en.pdf (visitado 17-07-2021) (vid. págs. 145, 149, 161).

Enlaces Web

[w1] E. Buetas, *Thesis MQTT-SCACAuth*. dirección: `https://github.com/EBuetas78/Thesis_MQTT-SCACAuth` (visitado 18-08-2021) (vid. págs. 73, 133, 151, 203).

[w2] MULTOS, *Multos SmartCard Technology*. dirección: `https://multos.com/technology/multos-smartcard-technology/` (visitado 03-02-2021) (vid. pág. 76).

[w3] ORACLE, *Eclipse Java Card Plug-in*. dirección: `https://docs.oracle.com/javacard/3.0.5/guide/eclipse-java-card-plug.htm#JCUGC126` (visitado 21-05-2021) (vid. pág. 77).

[w4] UTIMACO, *General purpose HSM*. dirección: `https://hsm.utimaco.com/products-hardware-security-modules/general-purpose-hsm/` (visitado 04-07-2021) (vid. pág. 81).

[w5] UTIMACO, *CryptoScript SDK*. dirección: `https://hsm.utimaco.com/products-hardware-security-modules/software-development-kit-sdk/cryptoscript-sdk/` (visitado 04-07-2021) (vid. pág. 81).

[w6] Amazon, *AWS CloudHSM*. dirección: `https://aws.amazon.com/es/cloudhsm/` (visitado 29-07-2021) (vid. pág. 82).

[w7] Google, *Cloud HSM*. dirección: `https://cloud.google.com/kms/docs/hsm?hl=es` (visitado 29-07-2021) (vid. pág. 82).

[w8] MUSCLE, *PCSC lite project*. dirección: `https://pcsclite.apdu.fr/` (visitado 25-05-2019) (vid. págs. 85, 86).

[w9] PC/SC Workgroup, *PC/SC Specification Files*. dirección: `https://pcscworkgroup.com/specifications/download/` (visitado 25-05-2019) (vid. pág. 85).

[w10] SIEMENS, *Descarga del SIMATIC STEP 7 (TIA Portal) V14 Trial*. dirección: `https://support.industry.siemens.com/cs/document/109740158/descarga-del-simatic-step-7-(tia-portal)-v14-trial?dti=0&lc=es-ES` (visitado 10-02-2021) (vid. pág. 109).

[w11] E. Foundation, *Open Source for IoT*. dirección: `https://iot.eclipse.org/` (visitado 09-07-2021) (vid. pág. 127).

[w12] R. Light, *Eclipse Mosquitto - An open source MQTT broker*. dirección: `https://github.com/eclipse/mosquitto` (visitado 05-04-2021) (vid. pág. 127).

[w13] E. EDA, *An Easier and Powerful Online PCB Design Tool*. dirección: `https://easyeda.com/` (visitado 20-05-2021) (vid. pág. 203).

Anexo A

Publicaciones Previas

En este anexo se muestran los artículos publicados por el autor, previos a la realización de esta tesis y que han servido como inicio de la investigación que ha dado como resultado la presente tesis.

A.1. Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach

Como parte de la investigación para la presente tesis, se ha realizado por parte del autor una profunda investigación en el entorno a las comunicaciones industriales en el ámbito de la industria 4.0, tanto en el ámbito de la seguridad como de su uso para el entorno del mantenimiento industrial. Como resultado de esta investigación se han realizado dos publicaciones en las que el autor es el investigador principal, en dos revistas dentro del primer cuartil (Q1) de la clasificación de impacto de JCR en el momento de la publicación de los artículos.

El primero de ellos, acerca de incluir un nuevo esquema de seguridad en el protocolo MQTT, incluyendo la utilización de algoritmos criptográficos, sentando la mejora de la seguridad de dicho protocolo. Este esquema de seguridad es utilizado como base para la seguridad de las comunicaciones en esta tesis doctoral. El artículo fue publicado en la revista IEEE Access (Fig. A.1) en Junio de 2020: Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach [50] (Fig. A.2).

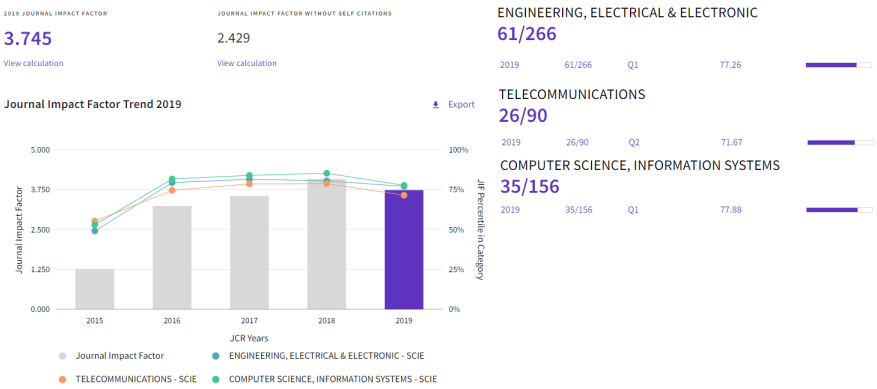


Figura A.1: Estadística IEEE Access en JCR.

Figura A.2: MQTT Security: A Cryptographic Smart Card Approach.

A.2. A propagation breakdown management model for the Industrial Internet of Things

El segundo artículo publicado por la revista Computers in Industry (Fig. A.3) en septiembre de 2020, sobre la utilización de un protocolo de publicación-suscripción en la propagación de averías en la industria "A propagation breakdown management model for the Industrial Internet of Things" [85] (Fig. A.4).

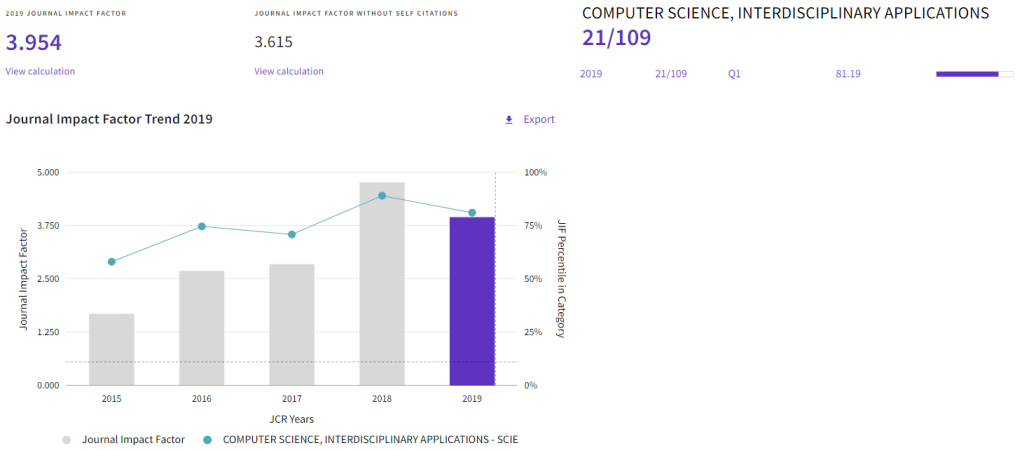


Figura A.3: Estadísticas Computers In Industry.

Figura A.4: A propagation breakdown management model for hte Industrial Internet of Things.

Anexo B

Mensajes MQTT-SCACAUTH

En este anexo se muestran ejemplos de codificación de mensajes MQTT con el esquema de seguridad MQTT-SCACAUTH, a nivel de byte, para poder mostrar con exactitud la composición de los mensajes enviados en los diferentes intercambios de datos utilizados en este protocolo.

B.1. CONNECT $C_x \Rightarrow B$

En las siguientes tablas (B.1 y B.2) podemos ver un ejemplo de composición de un mensaje CONNECT completo, para el inicio del proceso de autenticación mutua con el método de autenticación SCACAuth-RSAPKCS1-2048-AES-128 por parte del cliente cuyo UID es "12345678".

Tabla B.1: CONNECT SCACAuth-RSAPKCS1-2048-AES-128. Parte 1/2.

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x10	CONNECT
byte 1	byte 1	0xC2	Longitud restante (322)
byte 2	byte 2	0x02	
Cabecera Variable			
byte 0	byte 3	0x00	Longitud (4)
byte 1	byte 4	0x04	
byte 2-5	byte 5-8	'MQTT'	Protocolo
byte 6	byte 9	0x05	Version
byte 7	byte 10	0x00	Flag Byte
byte 8	byte 11	0x00	KeepAlive (60)
byte 9	byte 12	0x3C	
byte 10	byte 13	0xAC	Longitud prop. (300)
byte 11	byte 14	0x02	
byte 12	byte 15	0x15	METHOD code
byte 13	byte 16	0x00	Longitud método (30)
byte 14	byte 17	0x1E	
byte 15-44	byte 18-47	'SCACAuth-RSAPKCS1-2048-AES-128'	METHOD
byte 45	byte 48	0x16	DATA code
byte 46	byte 49	0x01	Longitud datos (264)
byte 47	byte 50	0x08	
byte 48-311	byte 51-314	$UID; C_{prx}(UID; RN_1)$	DATA

Tabla B.2: CONNECT SCACAuth-RSAPKCS1-2048-AES-128. Parte 2/2.

Nº byte	Nº byte (total)	Valor	Descripción
Payload			
byte 0	byte 315	0x00	Longitud (8)
byte 1	byte 316	0x08	
byte 2-9	byte 317-324	'12345678'	UID

B.2. AUTH $B \Rightarrow C_x$

En la siguiente tabla B.3 se muestra la composición de un mensaje AUTH completo como respuesta por parte del broker al CONNECT enviado por el cliente.

Tabla B.3: AUTH $C_x \Rightarrow B$ SCACAuth-RSAPKCS1-2048-AES-128.

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0xF0	AUTH
byte 1	byte 1	0xAB	Longitud restante (555)
byte 2	byte 2	0x04	
Cabecera Variable			
byte 0	byte 3	0x18	R.C. CONTINUE
byte 1	byte 4	0xA8	Longitud prop. (552)
byte 2	byte 5	0x04	
byte 3	byte 6	0x15	METHOD code
byte 4	byte 7	0x00	Longitud método (30)
byte 5	byte 8	0x1E	
byte 6-35	byte 9-38	'SCACAuth-RSAPKCS1-2048-AES-128'	METHOD
byte 36	byte 39	0x16	DATA code
byte 37	byte 40	0x02	Longitud datos (516)
byte 38	byte 41	0x04	
byte 39-554	byte 42-557	$MSB_1; LSB_1; Cpr_b(UID; RN_1);$ $MSB_2; LSB_2; Cpu_{cx}(UID; RN_u; RN_v)$	DATA

B.3. AUTH $C_x \Rightarrow B$

En la siguiente tabla B.4 se muestra la composición de un mensaje AUTH completo como respuesta por parte del cliente al mensaje AUTH enviado por el broker.

Tabla B.4: AUTH $C_x \Rightarrow B$ SCACAuth-RSAPKCS1-2048-AES-128.

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0xF0	AUTH
byte 1	byte 1	0xA7	Longitud restante (295)
byte 2	byte 2	0x02	
Cabecera Variable			
byte 0	byte 3	0x18	R.C. CONTINUE
byte 1	byte 4	0xA4	Longitud prop. (292)
byte 2	byte 5	0x02	
byte 3	byte 6	0x15	METHOD code
byte 4	byte 7	0x00	Longitud método (30)
byte 5	byte 8	0x1E	
byte 6-35	byte 9-38	'SCACAuth-RSAPKCS1-2048-AES-128'	METHOD
byte 36	byte 39	0x16	DATA code
byte 37	byte 40	0x01	Longitud datos (256)
byte 38	byte 41	0x00	
byte 39-294	byte 42-297	$Cpu_b(RN_v; RN_x; RN_y; RN_z)$	DATA

B.4. PUBLISH $B \Rightarrow C_x$

En la tabla B.5 podemos ver un mensaje PUBLISH enviado por el broker a un cliente con los elementos *topic* de longitud 14 bytes y *paylaod* de longitud 60 bytes.

Tabla B.5: PUBLISH $B \Rightarrow C_x$

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x34	PUBLISH (QoS 2)
byte 1	byte 1	0x85	Longitud restante (133)
byte 2	byte 2	0x01	
Cabecera Variable			
byte 0	byte 3	0x00	Longitud *topic* (48)
byte 1	byte 4	0x30	
byte 2-49	byte 5-52	$B64(BCE_{RNyn}(UID; 0x00; 0x0E;$ $''prueba/prueba0''))$	*topic*
byte 50	byte 53	0x00	Packet ID
byte 51	byte 54	0x01	
byte 52	byte 55	0x00	Longitud propiedades (0)
Payload			
byte 0-79	byte 56-135	$BCE_{RNxn}(UID; 0x00; 0x3C;$ $''01234 \ldots 7890123456789'')$	*payload*

B.5. PUBREC $C_x \Rightarrow B$

En la siguiente tabla B.6 podemos ver la composición del mensaje PUBREC enviado por el cliente al broker.

Tabla B.6: PUBREC $C_x \Rightarrow B$

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x50	PUBREC
byte 1	byte 1	0x5A	Longitud restante (90)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x01	
byte 2	byte 4	0x00	SUCCESS
byte 3	byte 5	0x56	Longitud prop. (86)
byte 4	byte 6	0x26	USER PROPERTY
byte 5	byte 7	0x00	Long. nombre propiedad (17)
byte 6	byte 8	0x11	
byte 7-23	byte 9-25	"SCACAUTH_PROPERTY"	Nombre propiedad
byte 24	byte 26	0x00	Longitud propiedad (64)
byte 25	byte 27	0x40	
byte 26-89	byte 28-91	$B64(BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)}))$	

B.6. PUBLISH $C_x \Rightarrow B$

En la tabla B.7 podemos ver un mensaje PUBLISH enviado por el cliente al broker con los elementos *topic* de longitud 14 bytes y *paylaod* de longitud 60 bytes.

Tabla B.7: PUBLISH $C_x \Rightarrow B$

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x34	PUBLISH (QoS 2)
byte 1	byte 1	0x85	Longitud restante (133)
byte 2	byte 2	0x01	
Cabecera Variable			
byte 0	byte 3	0x00	Longitud *topic* (48)
byte 1	byte 4	0x30	
byte 2-49	byte 5-52	$B64(BCE_{RNvn}(UID; 0x00; 0x0E;$ $''prueba/prueba0''))$	*topic*
byte 50	byte 53	0x00	Packet ID
byte 51	byte 54	0x01	
byte 52	byte 55	0x00	Longitud propiedades (0)
Payload			
byte 0-79	byte 56-135	$BCE_{RNun}(UID; 0x00; 0x3C;$ $''01234 \ldots 7890123456789'')$	*payload*

B.7. PUBREC $B \Rightarrow C_x$

En la siguiente tabla B.8 podemos ver la composición del mensaje PUBREC enviado por el broker un cliente.

Tabla B.8: PUBREC $B \Rightarrow C_x$

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x50	PUBREC
byte 1	byte 1	0x5A	Longitud restante (90)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x01	
byte 2	byte 4	0x00	SUCCESS
byte 3	byte 5	0x56	Longitud prop. (86)
byte 4	byte 6	0x26	USER PROPERTY
byte 5	byte 7	0x00	Long. nombre propiedad (17)
byte 6	byte 8	0x11	
byte 7-23	byte 9-25	"SCACAUTH_PROPERTY"	Nombre propiedad
byte 24	byte 26	0x00	Longitud propiedad (64)
byte 25	byte 27	0x40	
byte 26-89	byte 28-91	$B64(BCE_{RNun}(UID; RN_{u(n+1)}; RN_{v(n+1)}))$	

B.8. SUBSCRIBE

En la tabla B.9 podemos ver la composición de un mensaje SUBSCRIBE que realiza la suscripción de los *topics* "tension/#" y "temperatura/#", codificado según es necesario para cumplir el esquema de seguridad MQTT-SCACAUTH.

Tabla B.9: SUBSCRIBE

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x82	SUBSCRIBE
byte 1	byte 1	0x61	Longitud restante (97)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x01	
byte 2	byte 4	0x00	Longitud propiedades (0)
Payload			
byte 0	byte 5	0x00	Longitud Topic 1 (44)
byte 1	byte 6	0x2C	
byte 2-45	byte 7-50	$B64(BCE_{RNzn}(UID; 0x00; 0x09; \prime\prime tension/\#\prime\prime))$	Topic 1
byte 46	byte 51	0x02	Prop. Topic 1
byte 47	byte 52	0x00	Longitud Topic 2 (44)
byte 48	byte 53	0x2C	
byte 49-92	byte 54-97	$B64(BCE_{RNzn}(UID; 0x00; 0x0D; \prime\prime temperatura/\#\prime\prime))$	Topic 2
byte 93	byte 98	0x02	Prop. Topic 2

B.9. SUBACK

En la siguientes tabla B.10 podemos ver la composición del mensaje SUBACK.

Tabla B.10: SUBACK

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x90	SUBACK
byte 1	byte 1	0x4B	Longitud restante (75)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x01	
byte 2	byte 4	0x46	Long. propiedades (70)
byte 3	byte 5	0x26	USER PROPERTY
byte 4	byte 6	0x00	Long. nombre propiedad (17)
byte 5	byte 7	0x11	
byte 6-22	byte 8-24	"SCACAUTH_PROPERTY"	Nombre propiedad
byte 23	byte 25	0x00	Longitud propiedad (48)
byte 24	byte 26	0x30	
byte 25-72	byte 27-74	$B64(BCE_{RNzn}(UID; RN_{z(n+1)}))$	
Payload			
byte 0	byte 75	0x02	Reason Code Topic 1
byte 1	byte 76	0x02	Reason Code Topic 2

B.10. UNSUBSCRIBE

En la tabla B.11 podemos ver la composición de un mensaje UNSUBSCRIBE que realiza la cancelación de la suscripción de los *topics* "tension/#" y "temperatura/#", codificado según es necesario para cumplir el esquema de seguridad MQTT-SCACAUTH.

Tabla B.11: UNSUBSCRIBE

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0xA2	UNSUBSCRIBE
byte 1	byte 1	0x5F	Longitud restante (95)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x02	
byte 2	byte 4	0x00	Longitud propiedades (0)
Payload			
byte 0	byte 5	0x00	Longitud Topic 1 (44)
byte 1	byte 6	0x2C	
byte 2-45	byte 7-50	$B64(BCE_{RNzn}(UID;''tension/\#''))$	Topic 1
byte 46	byte 51	0x00	Longitud Topic 2 (44)
byte 47	byte 52	0x2C	
byte 48-91	byte 53-96	$B64(BCE_{RNzn}(UID;''temperatura/\#''))$	Topic 2

B.11. UNSUBACK

En la siguientes tablas (tabla B.12) podemos ver la composición del mensaje UNSUBACK.

Tabla B.12: UNSUBACK Parte 1/2

Nº byte	Nº byte (total)	Valor	Descripción
Cabecera Fija			
byte 0	byte 0	0x90	UNSUBACK
byte 1	byte 1	0x4B	Longitud restante (75)
Cabecera Variable			
byte 0	byte 2	0x00	Packet ID
byte 1	byte 3	0x02	
byte 2	byte 4	0x46	Long. propiedades (70)
byte 3	byte 5	0x26	USER PROPERTY
byte 4	byte 6	0x00	Long. nombre propiedad (17)
byte 5	byte 7	0x11	
byte 6-22	byte 8-24	"SCACAUTH_PROPERTY"	Nombre propiedad
byte 23	byte 25	0x00	Longitud propiedad (48)
byte 24	byte 26	0x30	
byte 25-72	byte 27-74	$B64(BCE_{RNzn}(UID; RN_{z(n+1)}))$	
Payload			
byte 0	byte 75	0x00	Reason Code Topic 1
byte 1	byte 76	0x00	Reason Code Topic 2

Anexo C

Esquemas

En este anexo se muestran los esquemas de los dispositivos utilizados en la realización de los prototipos. Todos los esquemas se han realizado con el software de libre distribución EasyEDA [w13] y se encuentran en el repositorio de esta tesis [w1].

C.1. Prototipo MKR1000

En la fig. C.1 se muestra el esquema del dispositivo utilizado para la realización del prototipo basado en el SoC MKR1000.

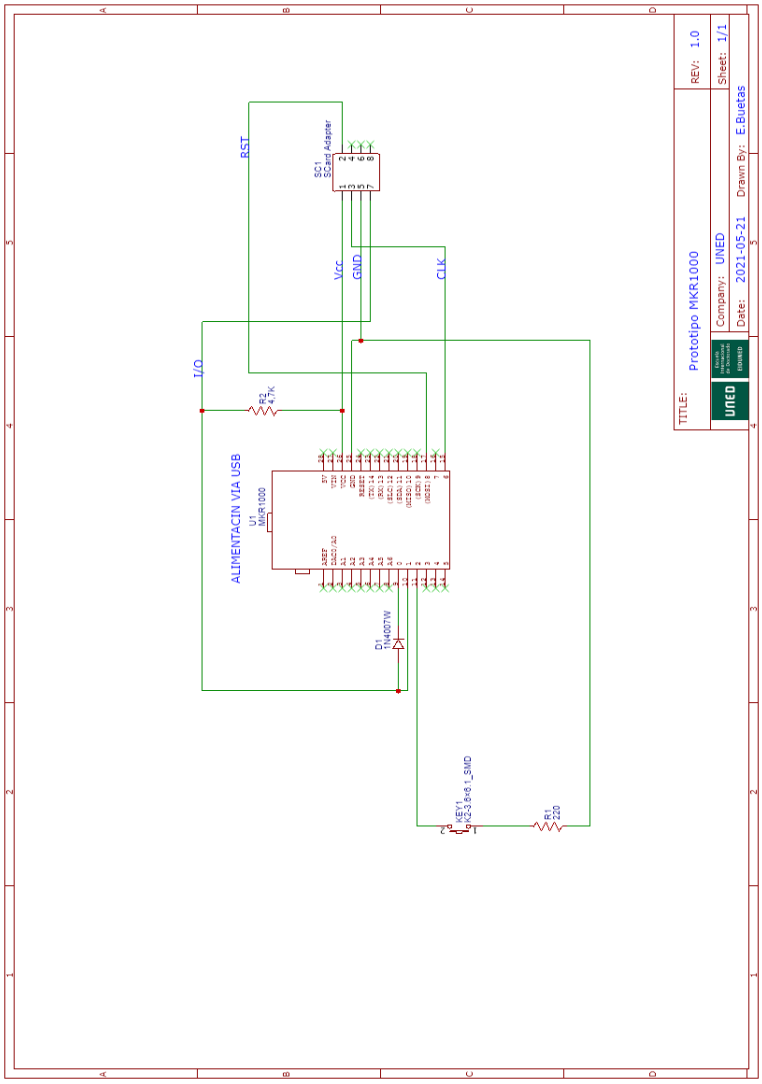


Figura C.1: Esquema prototipo SoC MKR1000.

C.2. Smart Card Interfaz RS-232

En la fig. C.2 se muestra el esquema del dispositivo de interfaz de comunicación con la Smart Card utilizado en el prototipo sobre PLC.

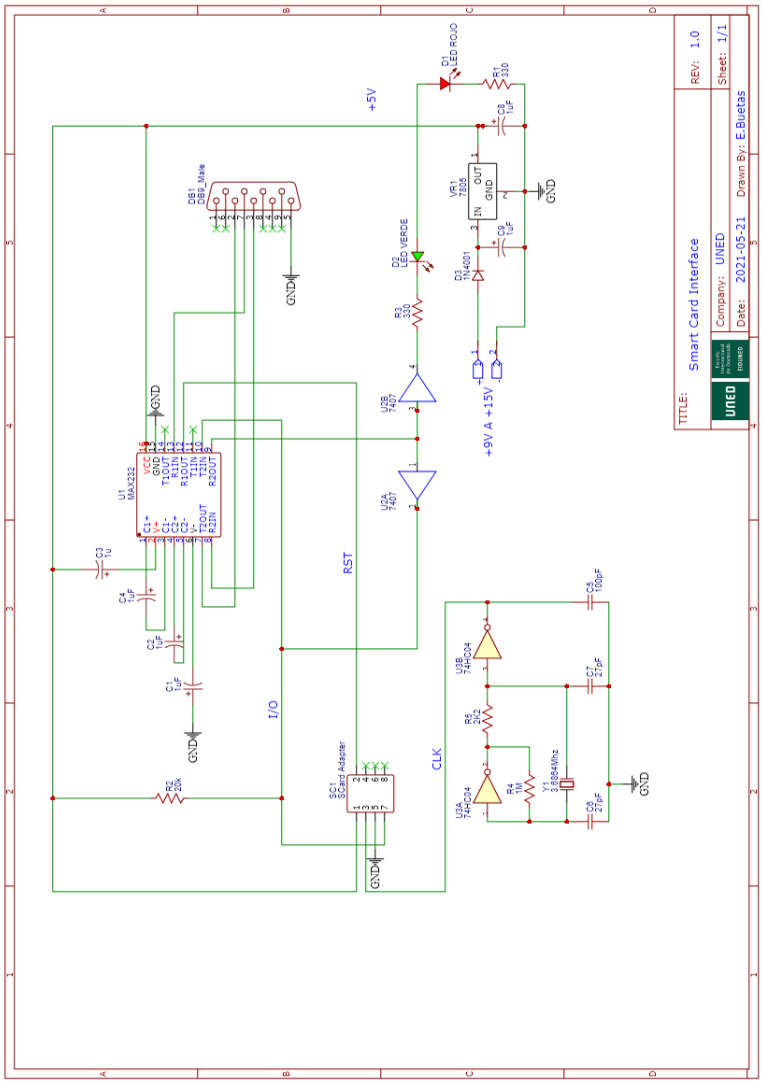


Figura C.2: Esquema dispositivo interfaz RS-232 Smart Card.

C.3. Prototipo Raspberry Pi

En la fig. C.3 se muestra el esquema del dispositivo utilizado para la realización del prototipo basado en la placa de desarrollo Raspberry Pi.

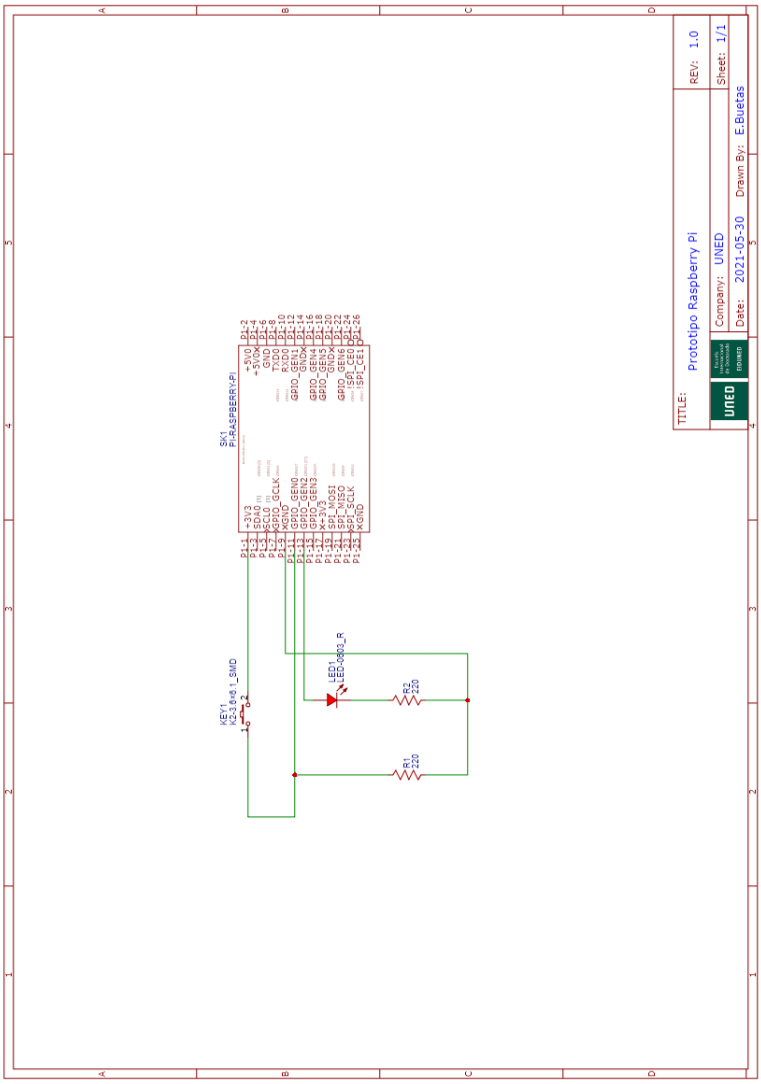


Figura C.3: Esquema prototipo RBPi.

C.4. Prototipo MKR1000 con Medición

En la fig. C.4 se muestra el esquema del prototipo utilizado para la realización de las mediciones de consumo eléctrico realizadas.

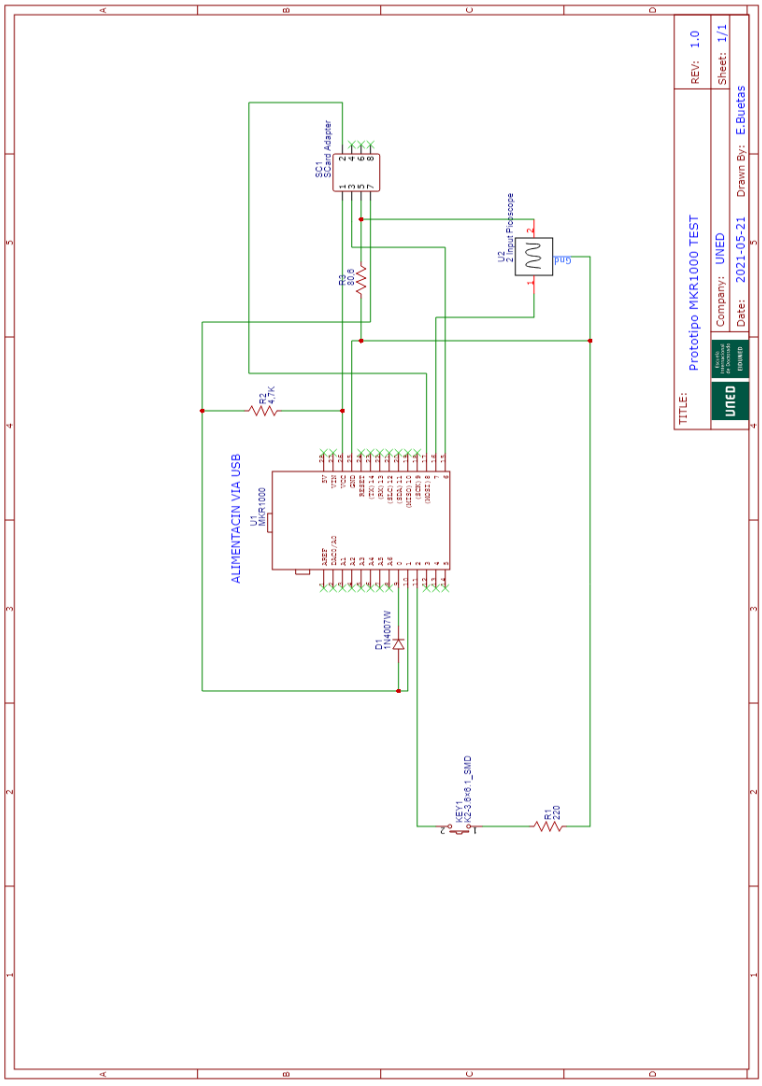


Figura C.4: Esquema prototipo SoC MKR1000 para mediciones de potencia.

Anexo D

Estructuras y Funciones Smart Card

En este anexo se muestran algunas estructuras y funciones relacionadas con las Smart Card utilizadas en este documento, para la mejor comprensión del mismo.

D.1. Answer to Reset (ATR)

El ATR es el mensaje que las Smart Card envían una vez realizado un reset a estas. En el ATR se envía información acerca de los parámetros de comunicación propuestos por la Smart Card y el estado de la misma.

En este mensaje hay datos como la frecuencia máxima de trabajo y la duración del bit de comunicación (byte TA_1), el retraso requerido entre bytes trasmitidos a la Smart Card (byte TC_1), el protocolo de comunicación sugerido (byte TD_1), el máximo numero de caracteres en un bloque de envío (byte TA_3), etc...

En [73] se realiza una definición detallada de todos los bytes que pueden componer el ATR.

D.2. Protocol Parameter Seleccion. (PPS)

El mensaje PPS se puede enviar después de la recepción del ATR para configurar los parámetros de comunicación con la Smart Card.

Este mensaje se compone de entre tres y seis bytes.

PPSS.

Este byte debe ser enviado siempre. Indica el comienzo de un mensaje PPSS y siempre debe codificarse como 0xFF.

PPS0

Este byte debe ser enviado siempre. Codifica los bytes que se envían a continuación y el protocolo de comunicación a utilizar. Su codificación se muestra en la tabla D.1.

Tabla D.1: Estructura byte PPS0 en mensjae PPS

Bit	Descripción
0..3	Indica el protocolo de transimisión a utilizar (1 a 15)
4	Indica la presencia o ausencia del byte PPS1
6	Indica la presencia o ausencia del byte PPS2
6	Indica la presencia o ausencia del byte PPS3
7	Reservado para usos futuros, siempre 0

PPS1

En este byte se envía la configuración de la velocidad de transmisión de datos con la Smart Card. Se configura con el mismo formato que el byte TA_1 del ATR. Habitualmente se envía el mismo valor que el recibido en el ATR en este byte para configurar la comunicación con el valor propuesto por la Smart Card en el ATR.

PPS2

Este byte sirve para configurar, con protocolo T=15, el uso estándar o propietario del pin C6 de la Smart Card (SPU por sus siglas en inglés). Debe ser codificado como el byte TB_i para T=15 del ATR.

PPS3

Este byte esta reservado para usos futuros.

PCK

Este byte debe ser enviado siempre. Es un byte de control que debe contener el resultado de la operación o-exclusiva del resto de bytes enviados.

D.3. Comandos APDU

En las tablas siguientes se muestra la estructura correspondiente a los mensajes intercambiados con las Smart Card, definido en la parte 4 de la norma ISO7816,

publicada por primera vez en 1995 y revisada por ultima vez en 2020 [s15].

En las tabla D.2 podemos ver la estructura de la cabecera y el cuerpo de los comandos APDU.

La cabecera del comando siempre debe tener una longitud de 4 bytes, mientras que el cuerpo del comando tiene una longitud variable dependiendo de los datos que se envían en el comando y de si existe o no la limitación del máximo de datos esperados en la respuesta.

Tabla D.2: Estructura Comando APDU

Nombre	Long.	Descripción
Cabecera		
CLA	1	Clase de instrucción: tipo de comando o fabricante
INS	1	Código de instrucción
P1	1	Parámetro 1 enviado a la instrucción
P2	1	Parámetro 2 enviado a la instrucción
Cuerpo		
L_c	0, 1 o 3	Nº bytes que siguen en el comando (N_c): Long. 0: N_c=0 Long. 1: N_c entre 1 y 255 Long. 3: El primer byte es 0, el byte 2 y 3 indican N_c entre 1 y 65535
Datos Env.	N_c	Datos enviados en el comando
L_e	0, 1, 2 o 3	Nº máximo de bytes esperados como respuesta (N_e) si el valor es 0 se solicitan todos los datos disponibles en la respuesta: Long. 0: N_e=0 Long. 1: N_e en el rango de 0..256 (0 indica N_e=256) Long. 2: N_e (Si L_c extendido esta presente en el mensaje) en el rango de 0..65536 (dos bytes iguales a 0 indica N_e=65536) Long. 3: N_e (Si L_c no esta presente en el mensaje) el primer byte debe ser 0 y los dos siguientes deben estar codificados como si se enviaran 2 bytes

En la tabla D.3 podemos ver la estructura de la respuesta que la Smart Card enviara como respuesta a un comando recibido.

Tabla D.3: Estructura Respuesta APDU

Respuesta		
Respuesta	N_r	Datos de respuesta N_r como máximo L_e en el caso de existir, todos los datos disponibles en caso contrario.
SW1-SW2	2	Estatus de respuesta

Existen siete tipos de mensajes APDU, dependiendo de los datos transmitidos en el cuerpo del mensaje, podemos ver estos tipos de mensaje en la tabla D.4.

Tabla D.4: Tipos de mensajes APDU

Tipo	Tipo Ext.	Long. Cuerpo	Long. L_c	Long. L_e	N_c	N_e
1	1	0	0	0	0	0
2	2S	1	0	1	0	1..255
NO	2E	3	0	3	0	1..65536
3	3S	$1+N_c$	1	0	1..255	0
NO	3E	$3+N_c$	0	3	1..65536	0
4	4S	$2+N_c$	1	1	1..255	1..255
NO	4E	$5+N_c$	3	3	1..65536	1..65536

D.4. Protocolo T=1

El protocolo T=1 es un protocolo de transmisión half-duplex para el intercambio de mensajes con las Smart Card. Su composición se muestra en la tabla D.5.

Este protocolo esta dividido en tres tipos de mensajes:

- Bloques de información (I-Blocks): Los cuales son usados para transferir comandos y respuestas APDUs.

- Bloques "listo para recibir" (R-Blocks): se utilizan para transferir acuses de recibo.
- Bloques de supervisión (S-Blocks): se utilizan para transferir información de control.

Tabla D.5: Estructura Mensajes T=1

Byte	Longitud	Contenido
Prólogo		
0	1	NAD
1	1	PCB
2	1	LEN
Información		
4	0..254	INF
Epílogo		
(Long. INF. + 4)	1	EDC

Byte de direcciones de nodo (NAD)

Este byte esta reservado para futuros usos, donde identificará el nodo de salida de los mensajes y el destino deseado de los mismos. En este momento debe ser codificado como 0x00 su estructura se puede ver en la tabla D.6.

Tabla D.6: Estructura Byte NAD

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
Sin uso	DAD			Sin uso	SAD		

Los elementos DAD (dirección de destino) y SAD (dirección emisora) estan reservados para futuros usos y en este momento deben ser codificadas como SAD = DAD = 0.

Byte de control de protocolo (PCB)

Este byte tiene una codificación diferente dependiendo del tipo de mensaje del que se trate, podemos ver su codificación en la siguiente tabla D.7.

Tabla D.7: Estructura Byte PCB

byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
I-Block							
0	N(S)	M	Reservado para futuros usos				
R-Block							
1	0	0	N(R)	Estatus de Respuesta (tabla D.8)			
S-Block							
1	1	X	Instrucción (tabla D.9)				

Número de secuencia enviada N(S): Este bit nos indica el número de secuencia enviada, debe cambiar en cada mensaje que se envíe por parte de un interlocutor.

Número de secuencia solicitada R(S): Este bit nos indica el número de secuencia del mensaje que solicita el emisor.

Bit de más datos (M) : Este bit se debe codificar a 1 cuando el mensaje enviado no es el último mensaje para la composición de un comando APDU extendido.

Bit 5 PCB S-Block: este bit se debe codificar a 0 cuando el mensaje es una petición y a 1 cuando es una respuesta.

Tabla D.8: Estatus Respuesta PCB R-Block

byte 3	byte 2	byte 1	byte 0	Estatus
0	0	0	0	Respuesta OK
0	0	0	1	Error EDC o paridad
0	0	1	0	Otros Errores
X	X	X	X	Otros valores RFU

Tabla D.9: Instrucción PCB S-Block

byte 4	byte 3	byte 2	byte 1	byte 0	Instrucción
0	0	0	0	0	Resincronización
0	0	0	0	1	Máxima long. de campo INF
0	0	0	1	0	Abortar
0	0	0	1	1	Extensión Time-out
0	0	1	0	0	Error VPP
X	X	X	X	X	Otros valores RFU

Byte de longitud (LEN)

En este campo se codifica la longitud del campo INF, admite valores de 0 a 254, 0 significa que no existe el campo INF y el valor 255 está reservado para futuros usos.

Campo de información (INF)

En los mensajes del tipo I-Block se coloca en este campo el comando o la respuesta APDU a transmitir. En los mensajes R-Block este campo no debe aparecer (LEN=0) y en los mensajes S-Block, dependiendo de la instrucción, se rellena con información relativa a la instrucción o no es utilizado (LEN=0).

Epílogo: Byte de código de detección de errores (EDC)

En este byte se codifica el byte de detección de errores que esta definido [s7] como una verificación de redundancia longitudinal (LRC por sus siglas en inglés). Esta verificación está basada en la realización cíclica de la operación XOR de cada byte con el resultado de la operación anterior. La operación en el primer byte se realiza con 0, dando como resultado el mismo byte.

D.5. Applet MQTT-SCACAuth para Clientes

D.5.1. Instrucciones para la inicialización de la Smart Card

Estas funciones solamente deben ser utilizadas en el proceso de inicialización de la Smart Card, previa a la primera conexión del cliente asociado a ellas. La finalización correcta de todas las funciones devuelve los valores de estatus SW1=0x90 y SW2=0x00.

Instrucción CREATE_PAIR (INS 0x20)

Este código de instrucción llama a la función "create_pair", debe ser invocada sin ningún dato adicional (L_c longitud 0).

Esta función genera el par de claves del cliente asociado a la Smart Card para la criptografía asimétrica, según el algoritmo y longitud de clave configuradas en el applet. Estas claves son almacenadas en un área de memoria persistente de la Smart Card.

Como respuesta envía el modulo y el exponente de la clave pública generada, con el formato mostrado en la tabla D.10.

Tabla D.10: Respuesta CREATE_PAIR

Byte	Longitud	Contenido
0	1	$(Long.\ exponente)_{MSB}$
1	1	$(Long.\ exponente)_{LSB}$
2	1	$(Long.\ modulo)_{MSB}$
3	1	$(Long.\ modulo)_{LSB}$
4	Long. Exponente	Exponente
(Long. Exp. + 4)	Long. Modulo	Modulo

Con la configuración utilizada, la longitud del modulo es de 256 bytes y la del exponente de la clave pública de 3 bytes por lo que la respuesta tendrá una longitud de 263 bytes.

Instrucción PUT_PUBLIC_KEY_BROKER (INS 0x22)

Este código de instrucción invoca a la función "put_public_key_broker", que debe ser llamada con datos adicionales de entrada. Estos datos contendrán el modulo y el exponente de la clave pública del broker al que se va a conectar el cliente asociado a la Smart Card. Estos datos deben tener el mismo formato que la clave pública devuelta en la instrucción CREATE_PAIR (tabla D.10), por lo que la longitud de estos datos es de 263 bytes (L_c=0x00 0x10 0x07).

Esta función almacena la clave pública transmitida a la Smart Card en un área de memoria no persistente como clave pública del broker.

Instrucción PUT_UID (INS 0x23)

Este código de instrucción invoca a la función "put_UID", que debe ser llamada con 8 bytes adicionales (L_c=0x08) que serán almacenadas en un aréa de memoria persistente de la Smart Card como el UID del cliente asociado a la Smart Card.

Instrucción INICIALICE_CIPHER (INS 0x2A)

Este código de instrucción invoca a la función "inicialice_cipher", que debe ser llamada con 8 bytes adicionales (L_c=0x08) que deben corresponder con el UID almacenado en la memoria persistente de la Smart Card.

En el caso de que los 8 byes de entrada no correspondan con el UID almacenado en la memoria persistente de la Smart Card, la función devuelve un valor especifico en el estatus de la respuesta, SW1=0xFF y SW2=0x84, comprobación UID errónea.

Si esta comprobación es correcta inicializa los objetos creados para la realización de tareas criptográficas.

Se crean cuatro objetos criptográficos, tres para criptografía asimétrica y uno para la criptografía simétrica, con los algoritmos y longitudes de clave configuradas:

- Pu_{broker}: Cifrado y descifrado con clave pública del broker, inicializándolo con la clave almacenada con la función put_public_key_broker.
- $Pu_{cliente}$: Cifrado y descifrado con clave pública del cliente, inicializándolo con la clave pública generada en la función create_pair.

- $Pr_{cliente}$: Cifrado y descifrado con clave privada del cliente, inicializándolo con la clave privada generada en la función create_pair.
- CSO: Cifrado y descifrado simétrico, en este caso no se inicializa la clave a utilizar y deberá ser pasada como parámetro en cada llamada al objeto.

Estos cuatro objetos criptográficos son almacenados en un área de memoria persistente, para su utilización en todos los procesos siguientes, incluso después de una perdida de tensión de la Smart Card.

D.5.2. Instrucciones utilizadas en la autenticación mutua entre el cliente y el broker

Estas funciones serán utilizadas en cada uno de los procesos de autenticación mutua entre el cliente y el broker.

Instrucción CREATE_AUTH_STEP1 (INS 0x25)

Este código de instrucción invoca a la función "create_auth_step1", no debe ser invocada con ningún dato adicional (L_c longitud 0).

En primer lugar genera un número aleatorio de longitud igual a la longitud de la clave configurada para la criptografía simétrica (16 bytes).

Para la generación de este número aleatorio se utiliza el algoritmo seguro de generación de números aleatorios proporcionado por el μprocesador criptográfico de la Smart Card. Este número se almacena en la memoria no persistente de la Smart Card como RN_1.

Se genera el cifrado de la concatenación de UID y RN_1 con el objeto $Pr_{cliente}$.

Una vez generado el cifrado se envía la repuesta, con el formato mostrado en la tabla D.11

Tabla D.11: Respuesta CREATE_AUTH_STEP1

Byte	Longitud	Contenido
0	8	UID
8	1	$(Long.\ Cpr_{cx}(UID; RN_1)_{MSB}$
9	1	$(Long.\ Cpr_{cx}(UID; RN_1)_{LSB}$
10	$(Long.\ Cpr_{cx}(UID; RN_1)$	$Cpr_{cx}(UID; RN_1)$

Con la configuración utilizada la longitud de los datos cifrados con criptografía asimétrica es de 256 bytes.

Instrucción CHECK_AUTH_STEP2 (INS 0x26)

Este código de instrucción invoca a la función "check_auth_step2", que debe ser llamada con 516 bytes adicionales para la configuración utilizada (L_c=0x00 0x20 0x04). Estos datos corresponden al cifrado con clave privada del broker de la concatenación de UID y RN_1 (256 bytes) más su longitud en formato *Big Endian* (2), más el cifrado con la clave pública del cliente asociado a la Smart Card de la concatenación del UID con dos números aleatorios RN_u y RN_v de longitud igual a la longitud de la clave de criptografía simétrica seleccionada (16 bytes), más su longitud en formato *Big Endian* (2 bytes).

Estos datos deben seguir el formato mostrado en la tabla D.12.

Tabla D.12: Parámetro CHECK_AUTH_STEP2

Byte	Longitud	Contenido
0	1	$(Long.\ Cpr_b(UID; RN_1)_{MSB}$
1	1	$(Long.\ Cpr_b(UID; RN_1)_{LSB}$
2	$(Long.\ Cpr_b(UID; RN_1)$	$Cpr_b(UID; RN_1)$
258	1	$(Long.\ Cpu_{cx}(UID; RN_u; RN_v)_{MSB}$
259	1	$(Long.\ Cpu_{cx}(UID; RN_u; RN_v)_{LSB}$
260	$(Long.\ Cpu_{cx}(UID; RN_u; RN_v)$	$Cpu_{cx}(UID; RN_u; RN_v)$

En primer, lugar esta función descifra la primera parte de los datos de entrada con el objeto Pu_{broker} y compara las dos partes del resultado del descifrado con los valores que están almacenados en la Smart Card como UID y RN_1. Si la comprobación es correcta continua con el proceso, si no lo es, devuelve un valor de error de autenticación en el estatus de la respuesta (SW1=0xFF SW2=0x80).

A continuación se descifra la segunda parte del mensaje con el objeto $Pr_{cliente}$ y se comparan los primeros 8 bytes del valor descifrado con el UID almacenado en la Smart Card. Si la comprobación no es correcta se devuelve un valor de error de la autenticación en el estatus de la respuesta (SW=0xFF SW=0x80). Si la comprobación es correcta se almacena en la memoria no persistente de la Smart Card el valor de los dos números aleatorios descifrados para futuros usos como RN_{un} y RN_{vn}.

Instrucción CREATE_AUTH_STEP3 (INS 0x27)

Este código de instrucción invoca a la función "create_auth_step3", que no debe ser llamada con ningún dato adicional (L_c longitud 0).

En primer lugar, genera tres números aleatorios de longitud igual a la longitud de la clave seleccionada para la criptografía simétrica (16 bytes), mediante el algoritmo ya utilizado para la generación de RN_1.

Estos tres números se almacenan en la memoria no persistente como RN_{xn}, RN_{yn} y RN_{zn}.

Se concatena el valor almacenado en la memoria persistente de la Smart Card como UID con estos tres números aleatorios. Esta concatenación de valores se cifra con el objeto Pu_{broker}.

Para finalizar, se envía el resultado de este cifrado como la respuesta a la instrucción, con el formato de la tabla D.13.

Tabla D.13: Respuesta CREATE_AUTH_STEP3

Byte	Longitud	Contenido
0	1	$(Long.\, Cpu_b(UID; RN_{xn}; RN_{yn}; RN_{zn})_{MSB}$
1	1	$(Long.\, Cpu_b(UID; RN_{xn}; RN_{yn}; RN_{zn})_{LSB}$
2	$(Long.$ $Cpu_b(UID; RN_{xn}; RN_{yn}; RN_{zn})$	$Cpu_b(UID; RN_{xn}; RN_{yn}; RN_{zn})$

Con la configuración utilizada los datos adicionales de respuesta tendrán una longitud de 258 bytes.

D.5.3. Instrucciones utilizadas en el intercambio de mensajes entre el cliente y el broker

Estas instrucciones se utilizan en los diferentes intercambios de mensajes entre el cliente y el broker una vez la autenticación mutua ha finalizado con exito.

Instrucciones CREATE_PAYLOAD (INS 0x28), CREATE_TOPIC (INS 0x38) y CREATE_TOPIC_SUB (INS 0x39)

Estas tres instrucciones invocan a la función "create_enc", esta debe recibir unos datos adicionales compuestos por una longitud variable de bytes, los cuales corresponden con el dato que se desea cifrar con el algoritmo simétrico de cifrado. El valor de L_c del comando APDU deberá ser rellenado dependiendo de la longitud de los datos a cifrar.

Dependiendo desde que instrucción es invocada esta función utiliza, una clave de cifrado u otra.

- CREATE_PAYLOAD: La clave de cifrado utilizada es RN_{un}, número aleatorio almacenado en la memoria no persistente de la SmartCard.
- CREATE_TOPIC: La clave de cifrado utilizada es RN_{vn}, número aleatorio almacenado en la memoria no persistente de la SmartCard.
- CREATE_TOPIC_SUB: La clave de cifrado utilizada es RN_{zn}, número aleatorio almacenado en la memoria no persistente de la SmartCard.

Una vez cifrados los datos recibidos como parámetro con el objeto CSO y la clave seleccionada, se envía como respuesta el resultado de este cifrado con el formato mostrado en la tabla D.14.

La longitud de la respuesta de esta función depende de la longitud de los datos a cifrar, siendo siempre el múltiplo de la longitud de la clave igual o superior a la longitud de los datos cifrados, más dos bytes que indican su longitud.

Tabla D.14: Respuesta create_enc

Byte	Longitud	Contenido
0	1	$(Long.\ BCE_{RN_?}(par\_entrada)_{MSB}$
1	1	$(Long.\ BCE_{RN_?}(par\_entrada)_{LSB}$
2	$(Long.\ BCE_{RN_?}(par\_entrada)$	$BCE_{RN_?}(par\_entrada)$

Instrucción CREATE_PUBREC (INS 0x31)

Este código de instrucción invoca a la función "create_pubrec", que debe ser llamada sin ningún dato adicional (L_c longitud 0).

Una vez iniciada esta función se generan dos números aleatorios con el algoritmo de generación seguro del μprocesador criptográfico de la Smart Card, con la longitud seleccionada.

Se concatena el UID almacenado en la memoria persistente de la Smart Card con estos dos números y se realiza el cifrado de esta concatenación con el objeto CSO, utilizando como clave el número aleatorio RN_{xn} almacenado en la memoria no persistente de la Smart Card.

Una vez realizado este cifrado, los números aleatorios generados se almacenan en la memoria no persistente de la Smart Card como los nuevos números RN_{xn} y RN_{yn}.

Como respuesta de esta función se devuelve el cifrado generado acompañado de su longitud en dos bytes, como se muestra en la tabla D.15.

Tabla D.15: Respuesta CREATE_PUBREC

Byte	Longitud	Contenido
0	1	$(Long.\ BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})_{MSB}$
1	1	$(Long.\ BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})_{LSB}$
2	$(Long.\ BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})$	$BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})$

La longitud resultante de este cifrado será de 48 bytes para la configuración utilizada, por lo que la longitud total resultante será de 50 bytes.

Instrucción CHECK_PUBREC (INS 0x29)

Este código de instrucción invoca a la función "check_pubrec", que debe ser llamada con 50 bytes adicionales para la configuración utilizada (L_c=0x32).

Estos bytes deben corresponder al resultado del cifrado, con el algoritmo simétrico seleccionado y RN_{un} como clave, de la concatenación del UID (8 bytes) del cliente asociado a la Smart Card y dos números aleatorios de longitud igual a la longitud seleccionada para la clave de cifrado simétrico (16 bytes), más dos bytes que corresponden a la longitud del resultado del cifrado.

Estos datos deben ser formateados tal como se muestra en la tabla D.16.

Tabla D.16: Parámetro CHECK_PUBREC

Byte	Longitud	Contenido
0	1	$(Long. \quad BCE_{RNun}(UID; RN_{u(n+1)}; RN_{v(n+1)})_{MSB}$
1	1	$(Long. \quad BCE_{RNun}(UID; RN_{u(n+1)}; RN_{v(n+1)})_{LSB}$
2	$(Long. BCE_{RNun}(UID; RN_{u(n+1)}; RN_{v(n+1)})$	$BCE_{RNun}(UID; RN_{u(n+1)}; RN_{v(n+1)})$

La función descifra los datos recibidos con el objeto CSO y como clave se utiliza el valor de RN_{un} almacenado en la memoria no persistente de la Smart Card.

Una vez descifrados los datos se realiza la comprobación de que los primeros 8 bytes del resultado del descifrado corresponden con el UID almacenado en la memoria persistente de la Smart Card, si la comprobación no es correcta se devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x83, indicando fallo coherencia de valores.

Si la comprobación ha sido correcta se almacenan los dos números descifrados como los nuevos valores RN_{un} y RN_{vn} en la memoria no persistente de la Smart Card.

Instrucciones READ_PAYLOAD (INS 0x48), READ_TOPIC (INS 0x58)

Estas dos instrucciones invocan a la función "read_enc", esta debe recibir unos datos adicionales compuestos por una longitud variable de bytes, los cuales corres-

ponden con el dato que se desea descifrar con el algoritmo simétrico de cifrado, encabezados por la longitud de estos datos, con una codificación igual a la respuesta de la función "create_enc" (tabla D.14). El valor de L_c del comando APDU deberá ser rellenado dependiendo de la longitud de los datos a descifrar.

Dependiendo desde qué instrucción es invocada esta función utiliza una clave de cifrado u otra.

- READ_PAYLOAD: La clave de descifrado utilizada es RN_{xn}, número aleatorio almacenado en la memoria no persistente de la SmartCard.
- READ_TOPIC: La clave de descifrado utilizada es RN_{yn}, número aleatorio almacenado en la memoria no persistente de la SmartCard.

Una vez descifrados los datos con el objeto CSO y la clave seleccionada, se comprueba, en primer lugar, que los ocho primeros bytes del resultado corresponden con el UID almacenado en la memoria persistente de la Smart Card. Si esta comprobación no resulta satisfactoria se devuelve en el estatus de la respuesta SW1=0xFF y SW2=0x83, indicando fallo coherencia de valores.

De ser la comprobación satisfactoria, se devuelve como respuesta de esta función el resto de datos descifrados, a partir del byte 8 del resultado del descifrado.

Instrucción CHECK_SUB_UNSUB_ACK (INS 0x2C)

Este código de instrucción invoca a la función "check_sub_unsub_ack", que debe ser llamada con 34 bytes adicionales para la configuración utilizada (L_c=0x22). Estos datos deben corresponder con el cifrado mediante el algoritmo simétrico seleccionado con el número RN_{zn} como clave, de la concatenación del UID del cliente con un número aleatorio de longitud igual a la longitud de clave simétrica seleccionada, acompañados de la longitud del cifrado en dos bytes, según el formato mostrado en la tabla D.17.

La función descifra los datos recibidos con el objeto CSO utilizando como clave el número RN_{zn} almacenado en la memoria no persistente de la Smart Card.

Tabla D.17: Parámetro CHECK_SUB_UNSUB_ACK

Byte	Longitud	Contenido
0	1	$(Long.\ BCE_{RNzn}(UID; RN_{z(n+1)})_{MSB}$
1	1	$(Long.\ BCE_{RNzn}(UID; RN_{z(n+1)})_{LSB}$
2	$(Long.\ BCE_{RNzn}(UID; RN_{z(n+1)})$	$(Long.\ BCE_{RNzn}(UID; RN_{z(n+1)})$

Una vez realizado el descifrado se comprueba que los ocho primeros bytes del resultado corresponden con el UID almacenado en la memoria persistente de la Smart Card. Si esta comprobación no resulta satisfactoria se devuelve en el estatus de la respuesta SW1=0xFF y SW2=0x83, indicando fallo de coherencia de valores. Si la comprobación es satisfactoria se almacena la segunda parte del descifrado como el nuevo número aleatorio RN_{zn} en la memoria no persistente de la Smart Card.

D.5.4. Instrucciones utilizadas con fines de test

Estas funciones se han introducido en el applet únicamente para comprobar el correcto funcionamiento del applet en el proceso de validación de la funcionalidad de dicho applet.

Instrucción GET_PUBLIC_KEY (INS 0x21)

Este código de instrucción invoca a la función "get_public_key", que debe ser llamada sin ningún dato adicional (L_c longitud 0) y envía como respuesta el modulo y el exponente de la clave pública almacenada en la memoria persistente de la Smart Card como clave pública del cliente, con el formato mostrado en la tabla D.10, siendo los valores de respuesta iguales a la respuesta de la función "create_pair".

Instrucción DAME_MEMORIA (INS 0x2B)

Este código de instrucción invoca a la función "dame_memoria", debe ser invocada sin ningún dato adicional (L_c longitud 0).

Esta función consulta la memoria disponible en la Smart Card. Hay tres tipos de memoria sobre las que se hace la consulta, la memoria persistente, la memoria no persistente que se libera en el momento de realizar un reset a la tarjeta y la memoria

no persistente que se libera en el momento de cambiar el applet seleccionado para su ejecución. En el caso del applet desarrollado estas dos últimas memorias tienen siempre el mismo valor ya que al existir únicamente un applet en la Smart Card no hay posibilidad de cambiar de applet.

La función utilizada para la lectura de estos tres tipos de memoria devuelve cada valor en enteros sin signo de 32 bits en formato *Big Endian*. Con estos valores se construye el valor devuelto por esta función en el formato mostrado en la tabla D.18.

Tabla D.18: Respuesta DAME_MEMORIA

Byte	Longitud	Contenido
0	4	Long. de memoria no persistente libre, liberada en el momento del reset
4	4	Long. de memoria no persistente libre, liberada en el momento del cambio de applet
8	4	Long. de memoria persistente libre

D.6. Applet MQTT-SCACAuth para broker

D.6.1. Instrucciones para la inicialización de la Smart Card

Estas instrucciones solamente deben ser invocadas en el proceso de inicialización de la Smart Card, previa a utilizar la Smart Card en el broker por primera vez. La finalización correcta de todas las funciones devuelve los valores de estatus SW1=0x90 y SW2=0x00.

Instrucción INIT_CLIENTES (INS 0x4F)

Este código de instrucción invoca a la función init_clientes, que debe ser llamada sin ningún dato adicional (L_c longitud 0).

En el byte P1 se debe incluir el número de clientes máximo que se incluyen en el almacén de claves de la Smart Card, con un máximo de 64 clientes. Si este parámetro

no está en el rango 1..64 la función devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x86.

Esta función inicializa las áreas de memoria necesarias para almacenar los datos de cada cliente: su clave pública, su UID y los números aleatorios relacionados con la comunicación con este cliente en cada momento (RN_1, RN_{un}, RN_{vn}, RN_{xn}, RN_{yn} y RN_{zn}).

Una vez inicializada una tarjeta se debe recargar el applet en la Smart Card para poder volver a redimensionar el almacén de claves de la Smart Card.

Si esta función ya ha sido ejecutada y se vuelve a intentar ejecutar se devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x85.

Instrucción CREATE_PAIR_BROKER (INS 0x20)

Este código de instrucción invoca a la función "create_pair_broker", que debe ser llamada sin ningún dato adicional (L_c longitud 0).

Esta función genera el par de claves del broker asociado a la Smart Card para la criptografía asimétrica, según el algoritmo y longitud de clave configuradas en el applet. Una vez generado este par de claves se inicializan las funciones criptográficas que se emplearán en el resto de funciones del applet.

Se crean tres objetos criptográficos, dos para criptografía asimétrica y uno para la criptografía simétrica, con los algoritmos y longitudes de clave configuradas:

- Pu_{broker}: Cifrado y descifrado con clave pública del broker.
- Pr_{broker}: Cifrado y descifrado con clave privada del broker.
- CSO: Cifrado y descifrado simétrico, en este caso no se inicializa la clave a utilizar y deberá ser pasada como parámetro en cada llamada al objeto.

Estos tres objetos criptográficos son almacenados en un área de memoria persistente, para su utilización en todos los procesos siguientes, incluso después de una perdida de tensión de la Smart Card.

Como respuesta envía el modulo y el exponente de la clave pública generada, con el formato mostrado en la tabla D.10.

Con la configuración utilizada, la longitud del modulo es de 256 bytes y la del exponente de la clave pública de 3 bytes, por lo que la respuesta tendrá una longitud de 263 bytes.

D.6.2. Instrucciones para la inicialización de un cliente

Estas funciones se deben ejecutar cada vez que se vaya a inicializar un cliente y almacenar su clave pública en la Smart Card del broker.

Instrucción GET_PUBLIC_KEY (INS 0x21)

Este código de instrucción invoca a la función "get_public_key", que debe ser llamada sin ningún dato adicional (L_c longitud 0) y envía como respuesta el modulo y el exponente de la clave pública almacenada en la memoria persistente de la Smart Card como clave pública del broker, con el formato mostrado en la tabla D.10.

Los datos recibidos como respuesta a esta instrucción se deben utilizar para almacenar la clave pública del broker en la Smart Card de cada cliente con la instrucción PUT_PUBLIC_KEY_BROKER del applet del cliente.

Instrucción CREATE_CLIENT (INS 0x40)

Este código de instrucción invoca a la función "create_client", que debe ser llamada con datos adicionales de entrada. Estos datos contendrán el UID del cliente a dar de alta, así como el modulo y el exponente de la clave pública de dicho cliente. Estos datos deben seguir el formato mostrado en la tabla D.19. Con la configuración utilizada en estos prototipos la longitud de estos datos es de 271 bytes (L_c=0x00 0x10 0x0F).

Esta función realiza una búsqueda en los clientes ya dados de alta en la Smart Card del broker y si encuentra un cliente con el mismo UID que el recibido, sustituye la clave pública de ese cliente por la recibida. Si no encuentra un cliente con el mismo UID crea uno nuevo en el primer hueco disponible en el almacén de claves. Inicializa las funciones criptográficas asimétricas para cifrar y descifrar con la clave pública del cliente y rellena el espacio de memoria destinado para el UID del cliente con el UID recibido.

Tabla D.19: Datos enviados a CREATE_CLIENT

Byte	Longitud	Contenido
0	8	$UID_{cliente}$
8	1	$(Long.\ exponente)_{MSB}$
9	1	$(Long.\ exponente)_{LSB}$
10	1	$(Long.\ modulo)_{MSB}$
11	1	$(Long.\ modulo)_{LSB}$
12	Long. Exponente	Exponente
(Long. Exp. + 12)	Long. Modulo	Modulo

Si no existe ningún hueco en el almacén de claves para un nuevo cliente la función devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x85.

Se genera un objeto para la criptografía asimétrica con la clave pública del cliente generado.

- $Pu_{cliente_x}$: Cifrado y descifrado con clave pública del cliente.

Instrucción DELETE_CLIENT (INS 0x60)

Este código de instrucción invoca a la función "delete_client", que debe ser llamada con datos adicionales de entrada. Estos datos contendrán el UID del cliente a borrar del almacén de claves. Con la configuración utilizada en estos prototipos la longitud de estos datos es de 8 bytes (L_c=0x08). En los datos enviados a la instrucción únicamente se enviaran los 8 bytes correspondientes al UID del cliente a borrar.

Si no existe el cliente a borrar se devolverá en el estatus de la respuesta el error SW1=0xFF y SW2=0x87.

D.6.3. Instrucciones para la autenticación mutua entre el cliente y el broker

Estas funciones serán utilizadas en cada uno de los procesos de autenticación mutua entre el cliente y el broker.

Instrucción CHECK_AUTH_STEP1 (INS 0x41)

Este código de instrucción invoca a la función "check_auth_step1", que debe ser llamada con 266 bytes adicionales para la configuración utilizada (L_c=0x00 0x10 0x0A). Estos datos deben coincidir en su composición y formato con la estructura mostrada en la tabla D.11.

En primer lugar, esta función selecciona el cliente a utilizar con el UID recibido en los datos de entrada y con su clave pública descifra el resto de los datos.

Si el UID recibido no cifrado no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

Se descifran los datos recibidos con el objeto $Pu_{cliente_x}$ correspondiente al cliente seleccionado. Si una vez descifrados los datos, la primera parte del descifrado no corresponde con el UID recibido se devuelve un valor de error SW1=0xFF y SW2=0x80.

Si por el contrario la función finaliza exitosamente se almacena la segunda parte del descifrado como el número aleatorio RN_1 del cliente seleccionado.

Instrucción CREATE_AUTH_STEP2 (INS 0x42)

Este código de instrucción invoca a la función "create_auth_step2", como datos adicionales se le debe enviar el UID del cliente receptor de este mensaje, en la configuración utilizada una longitud de ocho bytes (L_c=0x08).

Primero selecciona el cliente con el que se va a generar el mensaje, si no se encuentra el UID del cliente recibido en el almacén de claves se devuelve un valor de error en el estatus de la respuesta igual a SW1=0xFF y SW2=0x84.

Genera dos números aleatorios que almacena como RN_{un} y RN_{vn} y los almacena asociados al cliente seleccionado. Con el objeto Pr_{broker} genera el cifrado de la concatenación del UID y el RN_1 del cliente. Por último, se cifra la concatenación del UID del cliente seleccionado con los números aleatorios generados con el objeto $Pu_{cliente_x}$ correspondiente al cliente seleccionado.

La respuesta es construida como se muestra en la tabla D.12, para la configuración utilizada esta estructura tiene una longitud de 516 bytes.

Instrucción CHECK_AUTH_STEP3 (INS 0x27)

Este código de instrucción invoca a la función "check_auth_step3", que debe ser llamada con 266 bytes adicionales para la configuración utilizada (L_c=0x00 0x10 0x0A). Los primeros 8 bytes de estos datos deben corresponder con el UID del cliente, los 258 bytes restantes deben coincidir en su composición y formato con la estructura mostrada en la tabla D.13.

En primer lugar, esta función selecciona el cliente a utilizar con el UID recibido en los datos de entrada. Si el UID recibido no cifrado no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

A continuación, se descifran el resto de datos recibidos con el objeto Pr_{broker}. Si una vez descifrados los datos, la primera parte del descifrado no corresponde con el UID recibido se devuelve un valor de error SW1=0xFF y SW2=0x80.

Si por el contrario la función finaliza exitosamente se almacenan los números descifrados en la segunda parte del descifrado como los números aleatorios RN_{xn}, RN_{yn} y RN_{zn} del cliente seleccionado.

D.6.4. Instrucciones para el intercambio de datos con un cliente

Instrucciones CREATE_PAYLOAD_BROKER (INS 0x44), CREATE_TOPIC_BROKER (INS 0x45)

Estas dos instrucciones invocan a la función "create_enc", esta debe recibir unos datos adicionales compuestos por una longitud variable de bytes, los cuales corresponden con el UID del cliente destinatario de los datos, concatenado con los datos que se desea cifrar con el algoritmo simétrico de cifrado. El valor de L_c del comando APDU deberá ser rellenado dependiendo de la longitud de los datos a cifrar.

Si el UID recibido no cifrado no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

Dependiendo desde qué instrucción es invocada esta función utiliza una clave de cifrado u otra.

- CREATE_PAYLOAD_BROKER: La clave de cifrado utilizada es el RN_{xn} asociado al cliente receptor de los datos.
- CREATE_TOPIC: La clave de cifrado utilizada es RN_{yn} asociado al cliente receptor de los datos.

Una vez cifrados los datos recibidos como parámetro con el objeto CSO y la clave seleccionada, se envía como respuesta el resultado de este cifrado con el formato mostrado en la tabla D.14

La longitud de la respuesta de esta función depende de la longitud de los datos a cifrar, siendo siempre el múltiplo de la longitud de la clave igual o superior a la longitud de los datos cifrados, más dos bytes que indican su longitud.

Instrucciones READ_PAYLOAD_BROKER (INS 0x46), READ_TOPIC_BROKER (INS 0x47) y READ_TOPIC_BROKER_SUB (INS 0x48)

Estas tres instrucciones invocan a la función "read_enc", que debe recibir unos datos adicionales compuestos por una longitud variable de bytes, los cuales corresponden con el UID del cliente que ha cifrado los datos concatenados con los datos que se desea descifrar con el algoritmo simétrico de cifrado, encabezados por la longitud de estos datos. Con una codificación igual a la respuesta de la función "create_enc" (tabla D.14). El valor de L_c del comando APDU deberá ser rellenado dependiendo de la longitud de los datos a descifrar.

Dependiendo desde qué instrucción es invocada, esta función utiliza una clave de cifrado u otra.

- READ_PAYLOAD_BROKER: La clave de descifrado utilizada es RN_{un} asociado al cliente emisor de los datos.
- READ_TOPIC_TOPIC: La clave de descifrado utilizada es RN_{vn} asociado al cliente emisor de los datos.
- READ_TOPIC_BROKER_SUB: La clave de descifrado utilizada es RN_{zn} asociado al cliente emisor de los datos.

Una vez descifrados los datos con el objeto CSO y la clave seleccionada, se comprueba en primer lugar que los ocho primeros bytes del resultado corresponden con el UID asociado al cliente seleccionado, si esta comprobación no resulta satisfactoria se devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x83, indicando fallo coherencia de valores.

De ser la comprobación satisfactoria se devuelve como respuesta de esta función el resto de datos descifrados, a partir del byte 8 del resultado del descifrado.

Instrucción CHECK_PUBREC_BROKER (INS 0x49)

Este código de instrucción invoca a la función "check_pubrec_broker", que debe ser llamada con 58 bytes adicionales para la configuración utilizada (L_c=0x3A).

Los primeros ocho bytes de los datos pasados a esta función deben corresponder con el UID del cliente que ha enviado el mensaje PUBREC. Si el UID recibido no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

El resto de los bytes deben corresponder al resultado del cifrado, con el algoritmo simétrico seleccionado y RN_{xn} del cliente seleccionado como clave, de la concatenación del UID (8 bytes) del cliente seleccionado y dos números aleatorios de longitud igual a la longitud seleccionada para la clave de cifrado simétrico (16 bytes), más dos bytes que corresponden a la longitud del resultado del cifrado.

Estos datos deben ser formateados tal como se muestra en la tabla D.20.

Tabla D.20: Parámetro CHECK_PUBREC_BROKER

Byte	Longitud	Contenido
0	8	UID
8	1	$(Long. \quad BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})_{MSB}$
9	1	$(Long. \quad BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})_{LSB}$
10	$(Long. BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})$	$BCE_{RNxn}(UID; RN_{x(n+1)}; RN_{y(n+1)})$

La función descifra los datos recibidos con el objeto CSO y como clave se utiliza el valor de RN_{xn} del cliente seleccionado.

Una vez descifrados los datos se realiza la comprobación de que los primeros ocho bytes del resultado del descifrado corresponden con el UID del cliente seleccionado. Si la comprobación no es correcta se devolverá en el estatus de la respuesta SW1=0xFF y SW2=0x83, indicando fallo coherencia de valores. Si la comprobación ha sido correcta se almacenan los dos números descifrados como los nuevos valores RN_{xn} y RN_{yn} del cliente seleccionado.

Instrucción CREATE_PUBREC_BROKER (INS 0x4A)

Este código de instrucción invoca a la función "create_pubrec_broker", que debe ser llamada con el UID del cliente receptor, con la configuración actual será de 8 bytes (L_c=0x08).

Estos ocho bytes de los datos pasados a esta función deben corresponder con el UID del cliente receptor del mensaje PUBREC. Si el UID recibido no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

Se generan dos números aleatorios con el algoritmo de generación seguro del μprocesador criptográfico de la Smart Card, con la longitud seleccionada.

Se concatena el UID del cliente seleccionado con estos dos números y se realiza el cifrado de esta concatenación con el objeto CSO, utilizando como clave el número aleatorio RN_{un} del cliente seleccionado.

Una vez realizado este cifrado, los números aleatorios generados se almacenan como los nuevos números del RN_{un} y RN_{vn} del cliente seleccionado.

Como respuesta de esta función se devuelve el cifrado generado acompañado de su longitud en dos bytes, como se muestra en la tabla D.16.

La longitud resultante de este cifrado será de 48 bytes para la configuración utilizada, por lo que la longitud total resultante será de 50 bytes.

Instrucción CREATE_SUB_UNSUB_ACK (INS 0x29)

Este código de instrucción invoca a la función "create_sub_unsub_ack", que debe ser llamada con el UID del cliente receptor, con la configuración actual será de 8 bytes (L_c=0x08).

Estos ocho bytes de los datos pasados a esta función deben corresponder con el UID del cliente receptor del mensaje SUBACK o UNSUBACK. Si el UID recibido no corresponde con ningún cliente del almacén de claves de la Smart Card, se devuelve en el estatus de la respuesta un valor de error SW1=0xFF y SW2=0x84.

Se genera un número aleatorio con el algoritmo de generación seguro del μprocesador

criptográfico de la Smart Card, con la longitud seleccionada.

Se concatena el UID del cliente seleccionado con este número aleatorio y se realiza el cifrado de esta concatenación con el objeto CSO, utilizando como clave el número aleatorio RN_{zn} del cliente seleccionado.

Una vez realizado este cifrado el número aleatorio generado se almacena como el nuevo número RN_{zn} del cliente seleccionado.

Como respuesta de esta función se devuelve el cifrado generado acompañado de su longitud en dos bytes, como se muestra en la tabla D.17.

La longitud resultante de este cifrado será de 32 bytes para la configuración utilizada, por lo que la longitud total resultante será de 34 bytes.

D.6.5. Instrucciones utilizadas con fines de test

Esta función se ha introducido únicamente para comprobar el funcionamiento del applet en el proceso de validación de la funcionalidad del applet.

Instrucción DAME_MEMORIA_BROKER (INS 0x2B)

Esta función tiene el mismo funcionamiento que la función DAME_MEMORIA del applet generado para el cliente, punto D.5.4 de este anexo.

Anexo E

SPN Enviados

En este anexo se muestran los SPN enviados en el prototipo μPC J1939 (punto 5.10).

En la tabla E.1 podemos ver los SPN configurados para su envío periódico en el prototipo μPC J1939 (punto 5.10).

Tabla E.1: SPN configurado para envío periódico μPC J1939.

PGN	Byte	SPN	Descripción
0x00F003	1	91	Posición del pedal de aceleración
0x00F003	2	92	Porcentaje de carga del motor a la velocidad actual
0x00F004	2	513	Par actual del motor (%)
0x00F004	3-4	190	Velocidad del motor (rpm)
0x00FEE9	4-7	250	Consumo total de combustible (litros)
0x00FEEE	0	110	Temperatura refrigerante (^{o}C)
0x00FEF1	1-2	84	Velocidad de la rueda (km/h)
0x00FEF2	0-1	183	Consumo medio de combustible (Litros/hora)
0x00FEF2	2-3	184	Consumo instantáneo de combustible (km/litro)

Anexo F

Gráficas análisis de consumo en la Smart Card

En este anexo se muestran las gráficas obtenidas de los datos tratados en el análisis de consumo eléctrico de la Smart Card 6.2.

CREATE_AUTH_STEP1 (fig. F.1)

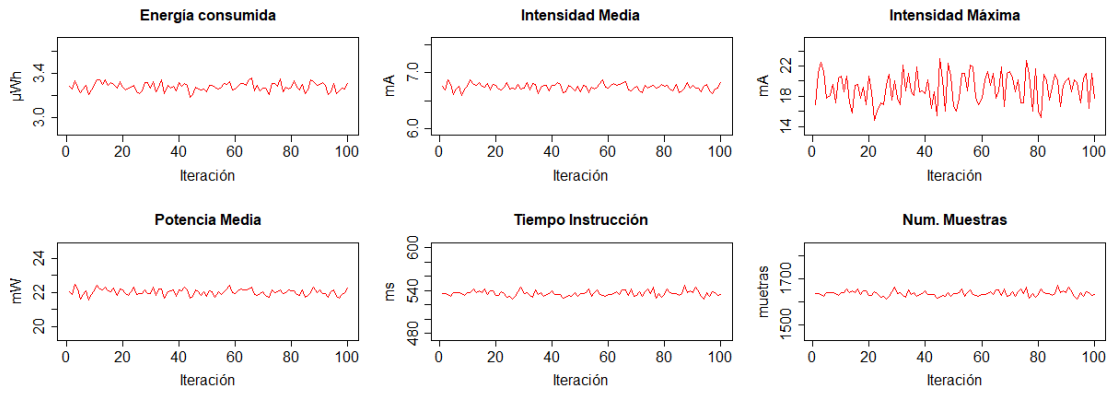


Figura F.1: Resultados mediciones eléctricas CREATE_AUTH_STEP1.

CHECK_AUTH_STEP2 (fig. F.2)

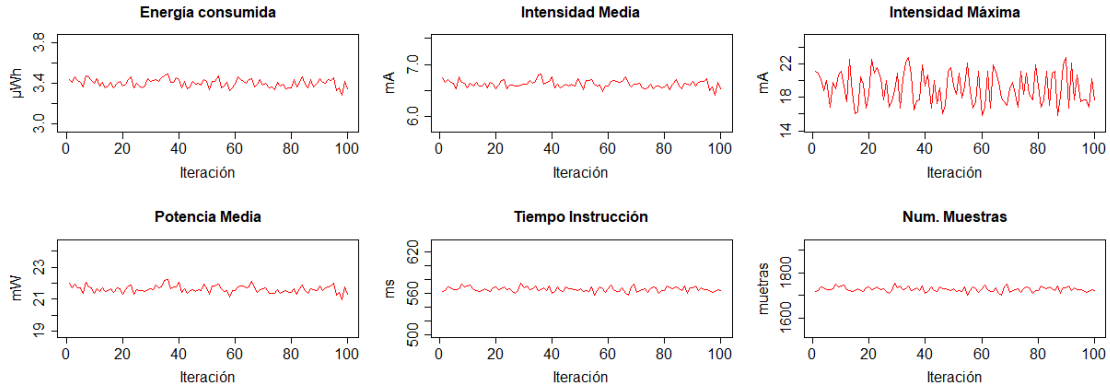


Figura F.2: Resultados mediciones eléctricas CHECK_AUTH_STEP2.

CREATE_AUTH_STEP3 (fig. F.3)

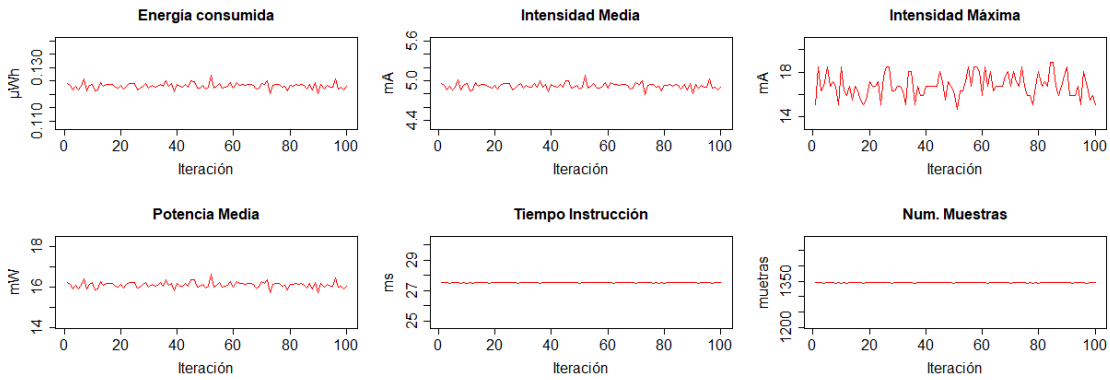


Figura F.3: Resultados mediciones eléctricas CREATE_AUTH_STEP3.

CREATE_ENC 16 bytes (fig. F.4)

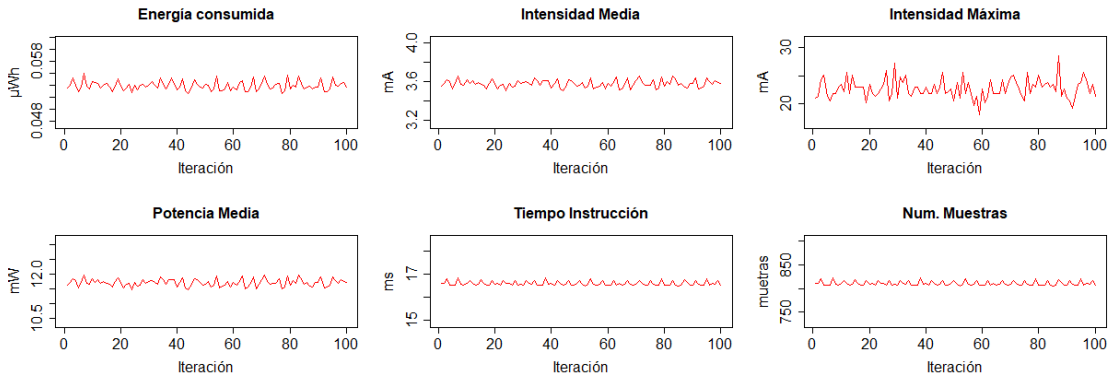


Figura F.4: Resultados mediciones eléctricas CREATE_ENC 16 bytes.

CREATE_ENC 32 bytes (fig. F.5)

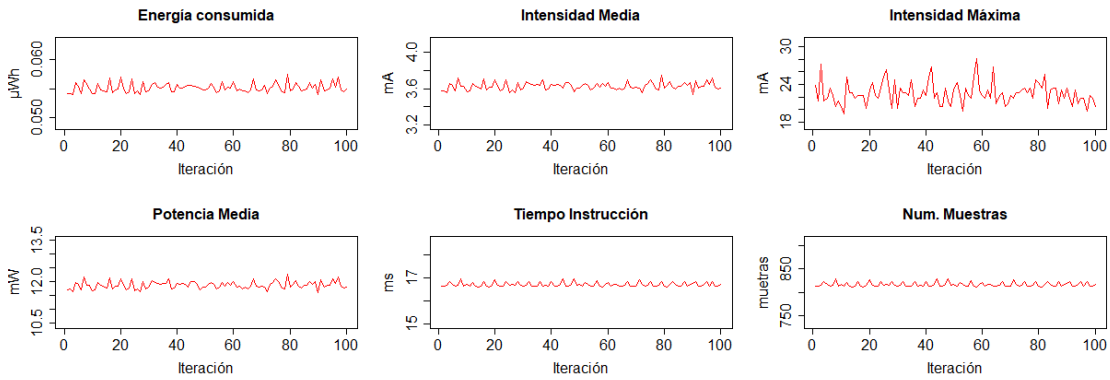


Figura F.5: Resultados mediciones eléctricas CREATE_ENC 32 bytes.

CREATE_ENC 64 bytes (fig. F.6)

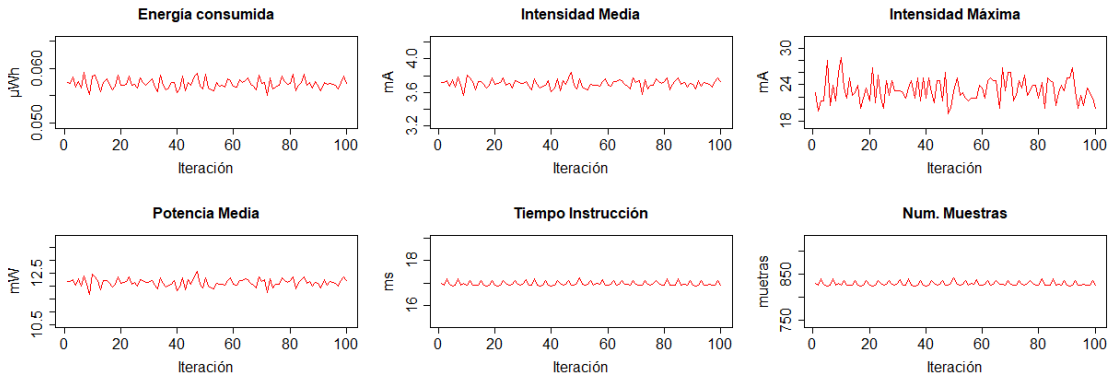


Figura F.6: Resultados mediciones eléctricas CREATE_ENC 64 bytes.

CREATE_ENC 128 bytes (fig. F.7)

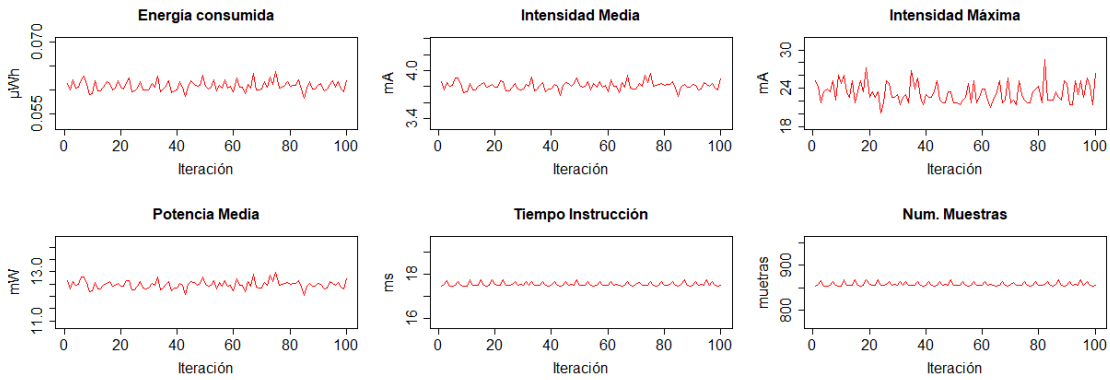


Figura F.7: Resultados mediciones eléctricas CREATE_ENC 128 bytes.

CREATE_ENC 256 bytes (fig. F.8)

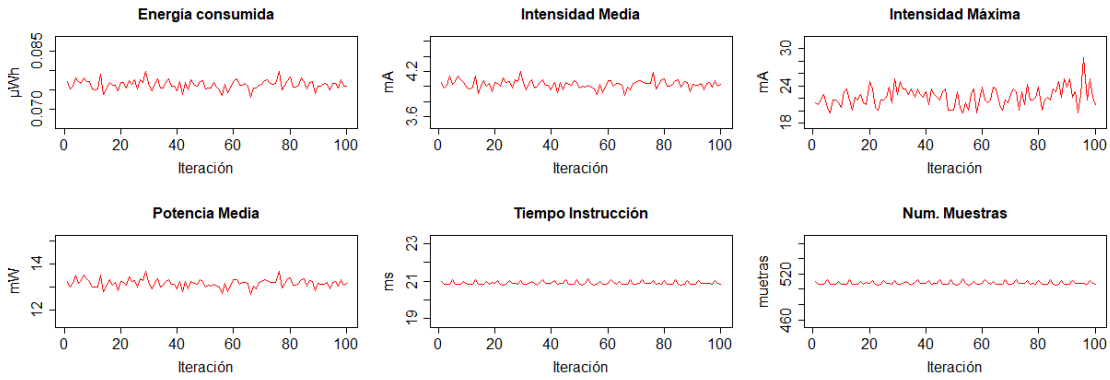


Figura F.8: Resultados mediciones eléctricas CREATE_ENC 256 bytes.

READ_ENC 16 bytes (fig. F.9)

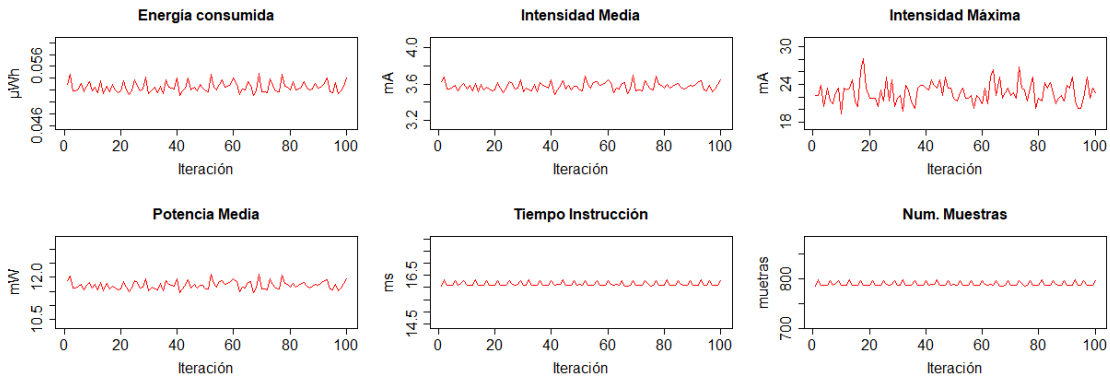


Figura F.9: Resultados mediciones eléctricas READ_ENC 16 bytes.

READ_ENC 32 bytes (fig. F.10)

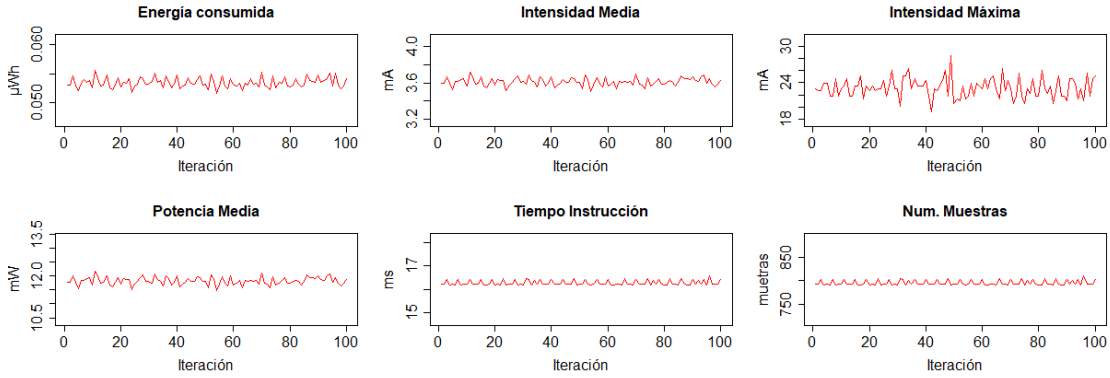


Figura F.10: Resultados mediciones eléctricas READ_ENC 32 bytes.

READ_ENC 64 bytes (fig. F.11)

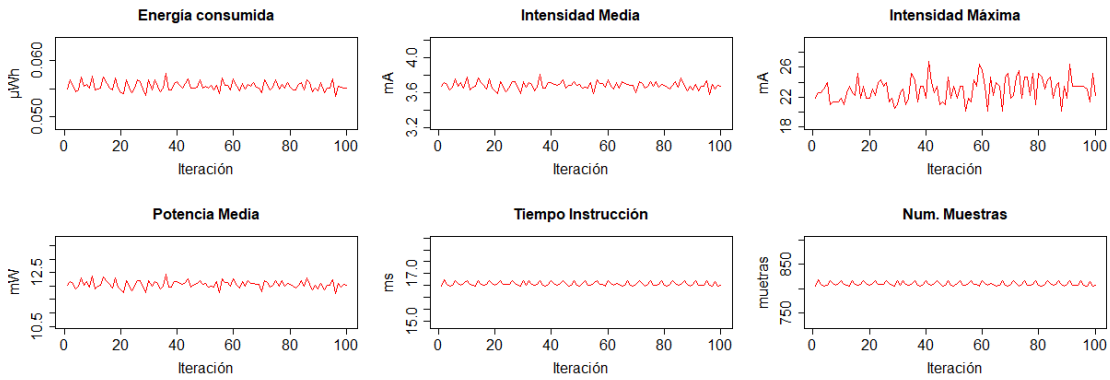


Figura F.11: Resultados mediciones eléctricas READ_ENC 64 bytes.

READ_ENC 128 bytes (fig. F.12)

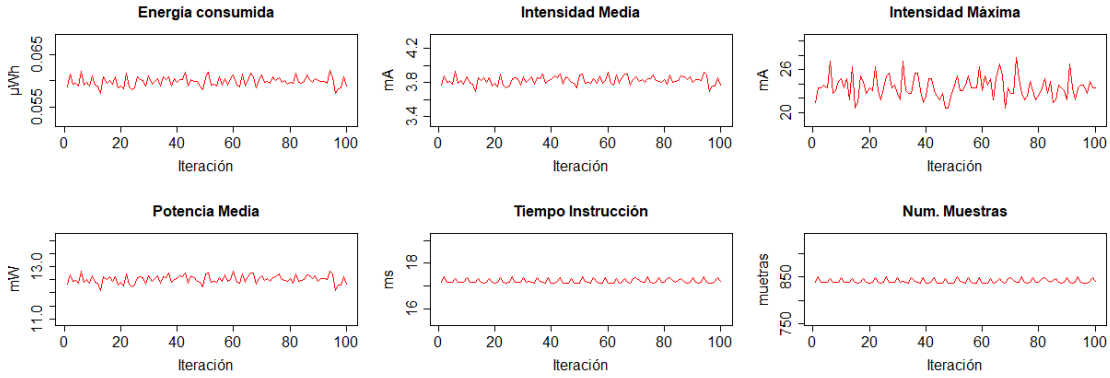


Figura F.12: Resultados mediciones eléctricas READ_ENC 128 bytes.

READ_ENC 256 bytes (fig. F.13)

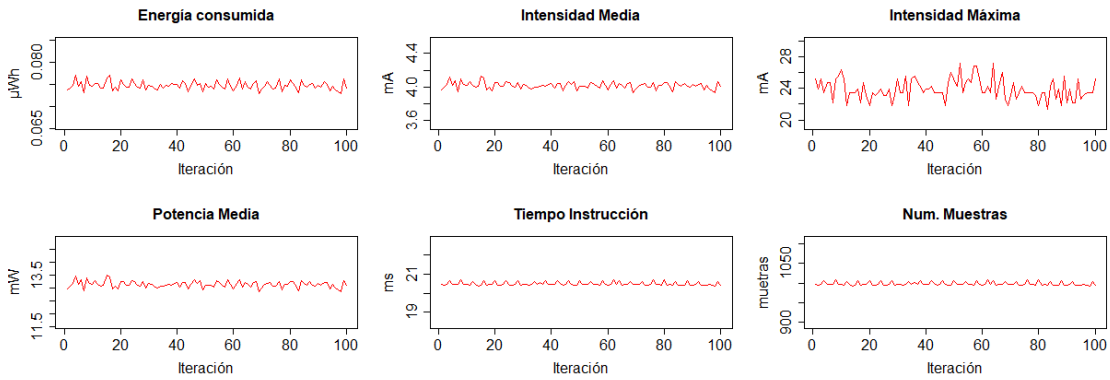


Figura F.13: Resultados mediciones eléctricas READ_ENC 256 bytes.

CREATE_PUBREC (fig. F.14)

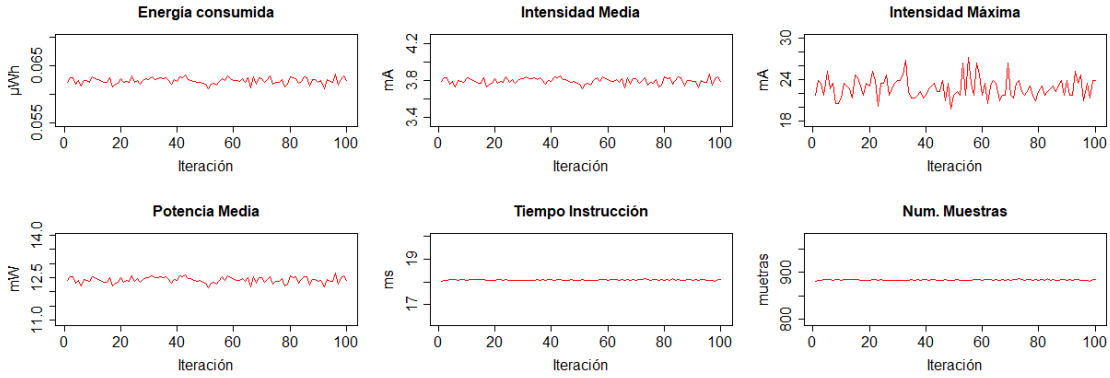


Figura F.14: Resultados mediciones eléctricas CREATE_PUBREC.