

Grado en Ingeniería en Tecnologías de Telecomunicación
2020-2021

Trabajo Fin de Grado

“IMPLEMENTACIÓN DE UN MECANISMO DE GESTIÓN DE CLAVES PARA OSCORE”

Guillermo García Hernández

Tutor/es

FLORINA ALMENARES MENDOZA

Director

JAIME PÉREZ DÍAZ

Leganés, 16 de julio de 2021



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

Resumen

Los sistemas IoT utilizan diversos protocolos de comunicación, cuyo soporte de seguridad se define en distintas capas. Uno de los protocolos es OSCORE (*Object Security for Constrained RESTful Environments*) que se trata de una extensión del protocolo CoAP (*Constrained Application Protocol*) que opera en la capa de aplicación, y proporciona un cifrado y firma de la información, pero entre sus especificaciones no se encuentra la de gestión y distribución de sus claves, provocando así que las implementaciones de este protocolo conlleven una gestión insegura de dichas claves, siendo ésta la motivación del proyecto.

El objetivo principal de este proyecto es implementar un sistema de gestión de claves para OSCORE, el cual permita añadir una capa de seguridad a las comunicaciones con sistemas IoT, algo altamente demandado y requerido. Esto es porque, debido a los tipos de recursos que se utilizan en dichas comunicaciones, se exigen entre otras cosas que sean livianos, por ello son objetivos de diferentes tipos de ataque.

Para alcanzar este objetivo principal se ha dividido el sistema en tres partes. Una inicial en la que se procederá a generar una Función Física no Clonable (PUF), que permitirá dar una señal de identidad al dispositivo en cuestión. Tras ello, se creará un subsistema de inscripción de un dispositivo, que estará ligado a uno de autenticación, permitiendo a un dispositivo autenticarse, usando en ambas tareas la PUF generada y estando implementadas mediante *sockets* TLS (*“Transport Layer Security”*). Por último, se extenderá la funcionalidad de OSCORE/CoAP, integrando el proceso de inscripción y

autenticación, con el fin de poder obtener de forma dinámica el contexto, la información necesaria para iniciar una conversación.

Con el fin de poder construir y validar el sistema propuesto, se utilizó una Raspberry Pi, como dispositivo cliente IoT que será con quien realizaremos la comunicación con un ordenador, que tendrá instalada una Máquina Virtual y actúa como servidor. Dicha Raspberry Pi, a su vez, se comunicará con una memoria SRAM, de la que podrá extraer sus bits estables, generando la PUF propuesta.

El código de este proyecto está disponible en el repositorio de GitHub:
<https://github.com/GuillermoGarH/OSCORE-with-PUF-for-key-management-system>

Palabras clave: Internet de las Cosas, CoAP, OSCORE, Californium, PUF, Serial Peripheral Interface, Inscripción, Autenticación

Abstract

IoT systems use various communication protocols, whose security support is defined in the different layers. One of the protocols is OSCORE (Object Security for Constrained RESTful Environments) which is an extension of the CoAP (Constrained Application Protocol). OSCORE operates at the application layer, and provides encoding and signing of information, but within its specifications, it does not cover the management and distribution of its keys, thus causing the implementations of this protocol to lead to an insecure management of these keys, this being the motivation of the project.

The main objective of this project is to create a key management system for OSCORE, which allows adding a layer of security to communications with IoT systems, something highly demanded and required. This is because, due to the types of resources that are used in said communications, they require, among other things, that they be light, which is why they are targets of different types of attack.

To achieve this main objective, the system has been divided into three parts, an initial one in which a Non-Clonable Physical Function (PUF) will be generated, which will allow an identity signal to be given to the device in question. After that, a device registration subsystem will be created, which will be linked to an authentication subsystem, allowing a device to authenticate itself, using the generated PUF in both tasks and being implemented using TLS sockets. Finally, the functionality of OSCORE / CoAP will be extended, in order to dynamically obtain the context, the information necessary to start a conversation.

In order to build the proposed system, we will need a Raspberry Pi, which will be the one with whom we will communicate with a computer, which will have a Virtual Machine installed. Said Raspberry Pi, in turn, will communicate with an SRAM memory, from which it will be able to extract its stable bits, generating the proposed PUF.

The code for this project will be available in the GitHub repository:
<https://github.com/GuillermoGarH/OSCORE-with-PUF-for-key-management-system>

Keywords: Internet of Things, CoAP, OSCORE, Californium, PUF, Serial Peripheral Interface, Enrollment, Authentication

Dedicatoria

Quiero dar las gracias inicialmente a mi familia, especialmente a mis padres, Sonia y Javier, que sin ellos no habría llegado hasta aquí, y a mis hermanas, Laura y Lucía, que, aunque haya los típicos rifirrafes de hermanos siempre van a estar ahí para mí.

También le quiero dar las gracias a mi novia, Mariel, por su infinita paciencia en un proyecto que me ha hecho envejecer cinco años de golpe del estrés, y ella siempre ha estado ahí cuando me he venido abajo.

No puedo olvidarme de mis amigos de la universidad y el colegio, con los que a lo largo de los años he creado relaciones o mejorado las que ya tenía, y han llenado estos años de miles de momentos para repetir y recordar.

Y por último a Florina, mi tutora y la persona que ha impulsado este proyecto, con la que he podido resolver cientos de dudas a lo largo del proyecto, y a Jaime, quien me ha estado ayudando también durante todo el proyecto, sacando horas de su tiempo aun cuando no tenía ninguna obligación.

Índice general

Introducción y Objetivos	26
1.1. Motivación del Trabajo.....	28
1.2. Objetivos.....	31
1.3. Estructura de la memoria	32
Estado del arte	35
2.1. CoAP.....	37
2.1.1. Mensaje CoAP.....	38
2.1.2. Tipos de peticiones CoAP	40
2.1.3. Ataques en CoAP	41
2.1.4. Diferentes implementaciones de CoAP	42
2.1.5. Californium.....	42
2.1.6. CoAP vs MQTT	44
2.2. OSCORE.....	45
2.2.1. OSCORE: La seguridad de CoAP.....	46
2.2.2. Contexto de seguridad de OSCORE.....	48
2.3. Funciones físicas no clonables (PUF).....	50
2.3.1. Propiedades.....	51
2.3.2. Tipos de PUFs	52
2.3.2.1. Arbiter PUF.....	52
2.3.2.2. SRAM PUF.....	52

2.3.2.3. <i>Butterfly</i> PUF	53
2.3.3. Autenticación mediante la PUF.....	53
2.3.3.1. Registro (<i>Enrollment</i>)	54
2.3.3.2. Autenticación (<i>Authentication</i>).....	54
2.3.3.3. Parejas Reto-Respuesta (<i>Challenge-Response</i>)	55
2.3.4. Derivación de la clave maestra.....	55
2.4. Trabajos relacionados	57
Descripción del sistema.....	60
3.1. Generación de la PUF	61
3.1.1. Raspberry Pi 3	61
3.1.2. Circuito integrado 23LC1024.....	62
3.1.3. Comunicación Raspberry Pi – SRAM a través de SPI.....	64
3.1.4. Diseño del circuito de comunicación entre el dispositivo y la SRAM.....	66
3.2. Inscripción y derivación de la clave para autenticación	67
3.3. Comunicación Cliente/Servidor mediante OSCORE	68
Implementación del sistema	71
4.1. Spidev: Librería de comunicación SPI para generar la PUF	71
4.1.1. Importación de librerías.....	72
4.1.2. Inicialización de variables	72
4.1.3. Extracción inicial de los bits.....	74
4.1.4. Detección de bits estables.....	75
4.1.5. Creación de la salida de la PUF.....	76
4.2. Inscripción y autenticación de la Raspberry	77
4.2.1. Implementación de <i>sockets</i> TLS.....	78
4.2.2. Selección de la funcionalidad	79
4.2.3. Inscripción del dispositivo.....	80
4.2.4. Autenticación del dispositivo	81
4.3. Comunicación OSCORE entre dispositivos	83
4.4. Integración de todos los subsistemas	86

Pruebas del sistema	89
5.1. Primera prueba: Generación de la PUF	90
5.2. Segunda prueba: Inscripción y autenticación del dispositivo	94
5.3. Tercera prueba: Comunicación OSCORE-CoAP	98
5.4. Evaluación de las pruebas	100
Entorno socio-económico y planificación.....	104
6.1. Entorno socio-económico	104
6.2. Planificación y presupuesto	106
6.2.1. Planificación del proyecto	106
6.2.2. Desarrollo de la comunicación CoAP/OSCORE.....	107
6.2.3. Desarrollo de la generación de la PUF	108
6.2.4. Conclusiones del desarrollo.....	109
6.3. Presupuesto	110
6.3.1. Costes materiales del proyecto	110
6.3.2. Costes por persona del proyecto.....	111
6.3.3. Costes finales del proyecto	112
Marco regulador.....	114
Conclusión y líneas futuras	122
8.1. Revisión de los objetivos propuestos.....	122
8.2. Líneas futuras.....	123
Referencias	126
Anexos.....	133
1. Summary.....	133
1.1. Introduction.....	133
1.2. Objectives	134
1.3. System Implementation	135

1.3.1. PUF generation.....	136
1.3.2. Enrollment and Authentication	137
1.3.2.1. Enrollment	138
1.3.2.2. Autenticación.....	138
1.3.3. OSCORE communication between devices	139
1.3.4. Integration of all subsystems	140
1.4. System tests.....	141
1.5. Conclusion of the project.....	141

Índice de figuras

Fig. 2.2 Formato del mensaje CoAP (32 bits)	38
Fig. 2.3 Opciones de CoAP (Opción OSCORE resaltada)	40
Fig. 2.1. Comparación de arquitectura de CoAP frente a MQTT	45
Fig. 2.5. Comunicación OSCORE.....	48
Fig. 2.6. Diagrama de un Arbiter PUF	52
Fig. 2.7. Diagrama de una SRAM PUF.....	53
Fig. 2.8. Diagrama de una Butterfly PUF.....	53
Fig. 2.9. Diagrama de generación inicial de la clave maestra	56
Fig. 2.10. Diagrama de reproducción de la clave maestra.....	57
Fig. 3.1. GPIO de la Raspberry	62
Fig. 3.2. Diagrama de la SRAM	63
Fig. 3.3. Diagrama de la SRAM	64
Fig. 3.4. Conexión Raspberry-SRAM con condensador de 0.1 μ F	67

Fig. 4.1. Secuencia de lectura byte a byte	74
Fig. 4.2. Diagrama de funciones del servidor (E y A).....	78
Fig. 4.3. Diagrama de funciones del cliente (E y A)	78
Fig. 4.4. Diagrama de funciones del servidor y cliente	84
Fig. 4.5. Diagrama de clases del servidor y cliente	84
Fig. 5.1. Lectura inicial de los bits	90
Fig. 5.2. Secuencia de bits	91
Fig. 5.3. Primera iteración del código	92
Fig. 5.4. Segunda iteración del código	92
Fig. 5.5. Última iteración del código	93
Fig. 5.6. Representación de la tabla clasificada como bits estables o inestables.....	93
Fig. 5.7. Evolución del algoritmo hasta alcanzar la estabilidad	94
Fig. 5.8. Inscripción del dispositivo en local	95
Fig. 5.9. Autenticación del dispositivo en local	95
Fig. 5.10. Pantalla de error del JAR	96
Fig. 5.11. Inscripción y autenticación (Servidor)	97
Fig. 5.12. Inscripción y autenticación (Cliente)	97
Fig. 5.13. Autenticación incorrecta (Servidor)	98
Fig. 5.14. Autenticación incorrecta (Cliente)	98
Fig. 5.14. Terminales del Cliente y Servidor con <i>debug</i>	99
Fig. 5.15. Terminales del Cliente	99

Fig. 5.16. Terminales del Servidor	100
Fig. A.1. Raspberry-SRAM connection with 0.1 μ F capacitor	135
Fig. A.2. Table representation classified as stable or unstable bits	136
Fig. A.3 Bit stream	136

Índice de tablas

Tabla 5.1. Evaluación de las pruebas.....	102
Tabla 6.1. Análisis del estudio previo	106
Tabla 6.2. Análisis de progreso de la comunicación OSCORE/CoAP.....	107
Tabla 6.3. Análisis de progreso de la extracción de la PUF	108
Tabla 6.4. Análisis del proyecto.....	109
Tabla 6.5. Presupuesto de los materiales	110
Tabla 6.6. Presupuesto del personal	111

Glosario de acrónimos

- AMQP: Protocolo avanzado de cola de mensajes
- API: Interfaz de Programación de Aplicaciones
- CBOR: Representación Concisa de Objetos Binarios
- CCN: Centro Criptológico Nacional
- CNPIC: Centro Nacional de Protección de Infraestructuras Críticas
- CoAP: Protocolo de aplicación restringida
- COSE AEAD: COSE Cifrado Autenticado con Datos Asociados
- COSE: CBOR Firma y Cifrado de Objetos
- CS: Selector de Chip
- CSIRT: Equipo de Respuesta ante Emergencias Informáticas
- DoS: Denegación de Servicio
- DRAM: Memoria de acceso aleatorio dinámico
- DTLS: Seguridad de la capa de transporte de datagramas
- ECC: Código de Corrección de Errores
- EEPROM: Memoria programable y borrrable eléctricamente de sólo lectura
- ENCS: Estrategia Nacional de Ciberseguridad
- ENS: Esquema Nacional de Seguridad
- FFCCSE: Fuerzas y Cuerpos de Seguridad del Estado
- GPIO: Entrada/Salida de Propósito General
- HDMI: Interfaz Multimedia de Alta Definición

- HTTP: Protocolo de Transferencia de Hipertexto
- IaaS: Infraestructura como Servicio
- IC: Infraestructuras críticas
- IoT: Internet de las Cosas
- IP6LoWPAN: IPv6 sobre WPAN de baja potencia
- IPv6: Protocolo de Internet versión 6
- ISO: Organización Internacional de Normalización
- IT: Tecnologías de la Información
- IV: Vector de inicialización
- JAR: Archivo Java
- JSON: Notación de Objeto de JavaScript
- M2M: Maquina a Maquina
- MISO: Maestro in – Esclavo out
- MOSI: Maestro out – Esclavo in
- MQTT: Transporte de Telemetría de Cola de Mensajes
- MRAM: Memoria de acceso aleatorio magnetoresistiva
- OCC: Oficina de Coordinación de Ciberseguridad
- OS: Sistema Operativo
- OSCORE: Seguridad de objetos para entornos RESTful restringidos
- OSE: Operadores de Servicios Esenciales
- OT: Tecnología de las Operaciones
- PEM: Correo con Privacidad Mejorada
- PKCS12: Estándar de Criptografía de Clave Pública # 12
- PPE: Plan de Protección Específico
- PSD: Proveedores de Servicios Digitales
- PSO: Plan de Seguridad del Operador
- PUF: Función Física no Clonable
- ReRAM: Memoria de acceso aleatorio resistiva
- REST: Transferencia de Estado Representacional
- RFC: Solicitud de Comentarios
- SCLK: Reloj SPI
- SCTP: Protocolo de Control de Transmisiones de Corrientes

- SPI: Interfaz periférica serial
- SRAM: Memoria estática de acceso aleatorio
- SSL: Seguridad de la Capa de Transporte
- TCP: Protocolo de Control de Transmisión
- TLS: Seguridad de la capa de transporte
- UDP: Protocolo de Datagramas de Usuario
- USB: Bus Universal en Serie
- Wifi: Fidelidad inalámbrica

Capítulo 1

Introducción y Objetivos

El mundo de la tecnología está en constante desarrollo, de una manera exponencial, lo que propicia la aparición de nuevas tecnologías disruptivas. Pero esto es solo la punta del iceberg, mucho más progreso está a la vuelta de la esquina, con nuevos elementos para cambiar nuestra concepción de la realidad y el día a día. Este proceso de cambio se ha dado en diversas ocasiones en nuestra historia, empezó con la primera revolución industrial, a finales del siglo XVIII, con la invención de la máquina de vapor, la siguiente gran revolución, la segunda revolución industrial, situándose en los inicios del siglo XX, con un gran avance en la industrialización, para dar pie a finales del mismo siglo a la tercera revolución industrial, con la automatización de las diferentes industrias.

Esto último aceleró de manera exponencial la creación de nuevas tecnologías, ya no era tanta la preocupación de cómo hacer llegar algo a la gente, debido a una sociedad altamente globalizada, sino más bien, qué idea sería más aceptada por ella. Consecuencia de esto, estos cambios desembocaron en la cuarta revolución [1], la era de la digitalización industrial, donde la interconexión y globalización están en constante expansión, fruto de estos acontecimientos aparecieron nuevas tecnologías nunca desarrolladas tan profundamente, como la inteligencia artificial, el *Machine Learning*, el *Big Data*, el

Cloud y la robótica, entre otros. Este cambio de paradigma provocó que cobrara gran importancia el internet, que pasó de ser algo utilizado para determinadas transacciones a constituirse en la base donde suceden las operaciones y se alberga la información. Este cambio, y la consiguiente explosión de los sistemas en la nube, conlleva la generación de nuevas capacidades de negocio y de relación, que aceleran exponencialmente las capacidades de los negocios y las personas, a veces con preponderancia sobre el mundo físico. Las empresas han entrado en una dinámica en la que deben subirse al mundo digital y migrar sus sistemas a la nube si quieren destacar por encima de otras, dejando obsoleta e insuficiente la tecnología ya presente.

El mundo actual se encuentra cada vez más interconectado, desde la nevera de la cocina y termostatos, hasta coches y aplicaciones en la salud. Este crecimiento es exponencial y esto conlleva grandes ventajas como:

- Acceso a datos de forma rápida y sencilla
- Costes de operación y evolución muy bajos
- Ahorro energético
- Automatización de procesos lo que conlleva un aumento de la productividad
- Integración con otros conceptos como el *Big Data*, *Machine Learning* o Inteligencia Artificial, para crear sistemas completos con funcionalidades avanzadas y gran utilidad

La tecnología IoT (*Internet of Things*) es por ello una tecnología disruptiva, tiene mucho futuro y capacidad de crecimiento, pero eso conlleva que también tiene un gran margen de mejora al tener también diversos inconvenientes como:

- Necesidad de un estándar que permita compatibilizar fácilmente diferentes dispositivos
- Falta de confidencialidad, al transmitirse los datos a través de canales que pueden ser fácilmente interceptados
- Aunque los costes sean bajos, al ser una tecnología reciente se necesita hacer una inversión inicial alta
- Facilidad de alteración de los datos transmitidos y recibidos

- No hay un sistema propio de autenticación fiable

Este cambio produce numerosas ventajas, pero también ha traído amenazas al mundo digital que antes solo se producían en el mundo físico, han crecido los ciberataques de manera exponencial. Es por ello por lo que el crecimiento en esta área de las tecnologías ha adquirido una gran importancia y un crecimiento de la inversión, produciéndose una mayor investigación sobre la ciberseguridad en estos tiempos.

Es por ello que esta revolución, denominada Revolución 4.0, ha traído numerosas ventajas y nuevos problemas, pero que finalmente todo lleva a uno de sus grandes hitos, la creación y el desarrollo de Internet de las Cosas (IoT), tecnología que aprovecha los avances en dicho mundo, como son la comunicación entre dispositivos y el tratamiento de los datos de dichas comunicaciones, siendo necesario también introducir sistemas de protección en esta tecnología, de la que aún hay mucho por desarrollar.

En conclusión, mientras que podemos ver que el IoT nos permite realizar multitud de funcionalidades, éstas no están libres de sus riesgos, y cómo hemos podido ver varios de estos están relacionados con la seguridad, siendo este el tema que vamos a abordar.

1.1. Motivación del Trabajo

Como mencionamos anteriormente la tecnología IoT es una tecnología disruptiva, por ello ahondar en ella es abordar un tema de futuro, pero sigue siendo una tecnología con mucho desarrollo por delante, por lo que el crear un proyecto sobre ello contribuye al avance en la materia. También, el mundo de Internet de las Cosas flaquea bastante en aspectos de seguridad, no habiendo un desarrollo extenso sobre ella. Esto puede no ser crítico en implementaciones simples como puede ser un proyecto casero, pero el IoT se aplica en muchos aspectos, entre ellos Infraestructuras Críticas.

Las Infraestructuras Críticas (a partir de ahora será denominado IC), son aquellos sistemas, instalaciones o estructuras que poseen una funcionalidad esencial para la ciudadanía, administraciones y empresas que conforman un estado y que no se pueden sustituir por otras, es decir, son estructuras que no poseen un plan “B” de actuación. Es

por ello por lo que es necesario crear una implementación robusta alrededor de ella para que no fallen. Muchas de estas IC están ligadas a la Industria 4.0, y al mundo IoT, como pueden ser el sector de la Energía (petróleo, gas, electricidad, nuclear, renovable, ...), sector de las Telecomunicaciones (infraestructuras nacionales de comunicación, enlaces internacionales, etc.), sector de la Sanidad (infraestructura hospitalaria y de atención pública, ...), sector de la Defensa, etc. Muchos de estos sectores se basan en sistemas de informática transaccional de los negocios y también de forma muy intensa en sistemas industriales. Estos sistemas no pueden fallar, ya que en consecuencia pueden provocar fallos catastróficos. Es por ello por lo que la necesidad de abordar la seguridad del sistema, y crear un método de defensa contra atacantes que quieran dañar estos elementos IoT, algo altamente requerido.

Adicionalmente, los gobiernos, conscientes de la relevancia que tienen estas infraestructuras para el funcionamiento del estado y el impacto tan alto que tendría el que dejarán de funcionar, han desarrollado legislaciones nacionales y europeas al respecto. Estas legislaciones cada vez son más exigentes en materia de ciberseguridad.

Y es que en los últimos años ha habido un crecimiento exponencial de este tipo de ataques que han puesto en jaque a grandes sistemas basados en el IoT. Algunos ejemplos recientes han sido el ciberataque que sufrió The Colonial Pipeline Company [2], que debido a un *malware* introducido en la red IT, tuvo que desconectar preventivamente ciertos sistemas en la red IT y OT lo que produjo la paralización de las operaciones del oleoducto y llevaron a diferentes errores en la distribución del gas y petróleo por toda la costa Este de Estados Unidos. El domingo 9 de mayo de 2021, el presidente Biden declaró el estado de emergencia tras la escasez de suministro de gas en la costa este. El FBI confirmó el día 10 de mayo de 2021 que el ataque a The Colonial Pipeline había sido realizado por parte de los operadores del *ransomware* Darkside. Todo ello redujo el abastecimiento de combustible y conllevó una subida exponencial de los precios y escasez de gas en algunos lugares, provocando preocupación e incertidumbre entre la población.

Otro caso similar fue a principios del 2021 el del ciberataque a la planta de agua en la ciudad de Oldsmar (Florida [3], Estado Unidos). Remotamente, unos *hackers*, tomaron el control de los sistemas de supervisión y control de los elementos y sustancias químicas de la planta, aumentando en gran medida la cantidad de Hidróxido de Sodio disuelto en

el agua, una sustancia altamente corrosiva y que, en el agua, en cantidades muy grandes puede conllevar que se produzca la combustión si se combina con fuentes de mucho calor, algo que solo se pudo solucionar cuando se retomó el control.

El caso más destacado tuvo incluso un alcance político, que promovió a aumentar las tensiones entre Ucrania y Rusia [4], en la que a través de *phishing* entraron a la red, y una vez dentro, atacaron utilizando un *exploit*, el cual consistió en utilizar los sistemas IoT dentro de la red para controlar los diferentes elementos de la red que estaban conectados y controlados a través de simples ordenadores Windows no debidamente bastionados y actualizados, lo que implica un bajo nivel de seguridad para un atacante con buenos conocimientos en ciberataques. Este ataque perpetrado por ciberatacantes afines o respaldados por el gobierno ruso, conllevó que no llegara electricidad a numerosos hogares, algo crítico, provocando diversos apagones. Esto provocó que las tensiones de por sí ya existentes entre Rusia y Ucrania por otros asuntos políticos subieran de manera apreciable, lo que nos da una idea de hasta cuánto daño puede hacer un sistema IoT inseguro.

Existen otros ejemplos relevantes sobre el uso que hacen los estados de los ataques informáticos para desestabilizar a sus adversarios y conseguir sus objetivos políticos y bélicos. Casos de esta naturaleza son la explosión provocada por los Estados Unidos e Israel mediante medios informáticos en una planta de enriquecimiento de uranio en Irán para frenar su programa nuclear [5], o el realizado por Irán a la empresa estatal petrolera de Arabia Saudí (Saudi Aramco) [6] cifrando y borrando todos sus discos duros, como medio para parar la producción de petróleo y desestabilizar a un enemigo regional directo.

Por todos estos ejemplos y por más que ha habido y por haber es necesario hacer un mayor desarrollo en la seguridad de este tipo de sistemas, por lo que diseñar un sistema de seguridad dentro del ámbito de Internet de las Cosas, debido al alcance que puede tener y la necesidad que presenta en el mundo actual, propone un aspecto muy interesante a abordar, a la vez que, como se acaba de determinar, algo totalmente necesario debido al alcance de las consecuencias en caso de que no se lleve a cabo una mayor investigación y desarrollo del tema. Por ello abordar un proyecto que profundice y desarrolle acerca de la seguridad IoT permite dar una nueva visión y tomar parte del avance necesario para la

transformación necesaria para crear un mundo digital más seguro en el que estén presentes este tipo de tecnologías.

1.2. Objetivos

El objetivo principal de este trabajo es desarrollar e integrar dentro de una comunicación que usa el protocolo OSCORE (*Object Security for Constrained RESTful Environments*) un sistema de gestión de claves y autenticación basado en hardware, con el fin de establecer una comunicación segura entre un cliente y servidor, dificultando a un posible atacante la manera de entrar en el sistema.

Como hemos nombrado anteriormente, dicha comunicación llevará integrada el protocolo OSCORE, con el que codificar la comunicación, que, junto a CoAP (*Constrained Application Protocol*), permitan llevar a cabo una comunicación segura entre una Raspberry Pi (actuando como cliente) y un servidor en un ordenador, con la particularidad de que OSCORE protege los datos a transferir y el método de *Request/Response*, dejando el resto desprotegido, lo cual nos permite enviar mensajes más ligeros.

Para dar una mayor capa de seguridad, en vez de tener la llave maestra con la que se realizará la comunicación directamente en el código o escrita en un fichero, se procederá a crear un sistema con el que se derivará la llave secreta. Esto se llevará a cabo con una PUF (*Physical Unclonable Function*), implementada con una SRAM (*Static Random Access Memory*).

Con el fin de cumplir el objetivo principal se han definido diversos objetivos específicos:

1. Crear una librería con la que poder leer desde una Raspberry Pi una memoria SRAM a la vez que poder discernir qué bits son estables en dicha memoria.
2. Desarrollar los protocolos de inscripción y autenticación del dispositivo en cuestión frente a un servidor para la creación de un sistema de derivación de claves.
3. Extender y desplegar un cliente CoAP en el que se integren las librerías dispuestas anteriormente, con el que abordar la comunicación entre un cliente y un servidor, a través de OSCORE.
4. Desarrollar un servidor de gestión de claves y autenticación basado en PUF.

5. Desplegar y extender un servidor de comunicación que utilice OSCORE con el nuevo mecanismo de derivación de la clave.

1.3. Estructura de la memoria

El documento por presentar será dividido en capítulos. La estructura de dicho documento es la siguiente:

- En el **capítulo 2, Estado del arte**, se tratan los conceptos teóricos de IoT, como es una tecnología en auge y en constante crecimiento, CoAP, el protocolo de comunicación entre el cliente, la Raspberry Pi, y el servidor, implementado en un ordenador desde una máquina virtual. La seguridad de la comunicación CoAP se basa en OSCORE, ya que, aporta protección a la comunicación. En este caso OSCORE es extendido para que la gestión de la clave maestra, con la cual se realiza la sesión OSCORE, sea derivada de una PUF, la cual también será la *huella dactilar* con la que se podrá autenticar el cliente frente al servidor.

Además, en este capítulo se introducen los proyectos relacionados previos, esclareciendo las diferencias con este trabajo.

- En el **capítulo 3, Descripción del sistema**, se introducen los componentes de nuestro circuito, y define la arquitectura en la que se basará el sistema, con los diferentes módulos en los que se compondrá y la funcionalidad de cada uno, para después explicar cómo se entrelazan e integran entre sí.
- En el **capítulo 4, Implementación del sistema**, se realiza una implementación en la que describirán sus características (lenguaje de programación, método de integración, estructura del código, etc.).
- En el **capítulo 5, Pruebas del sistema**, se realizan las pruebas subsecuentes, en las que se prueba cada elemento de manera independiente para después integrarlo todo en un gran sistema.

- En el **capítulo 6, Desarrollo del sistema y marco regulador**, se trata al detalle la planificación y presupuesto del proyecto, definiendo el impacto socioeconómico del mismo, el impacto que tendrá dicho proyecto en un contexto global aplicado en lo que podría ser una situación real y por ello tratando también así las diferentes regulaciones a las que se sometería en ese caso, haciendo un análisis de dichas regulaciones y cómo afectan al proyecto.
- En el **capítulo 7, Conclusiones**, se define las conclusiones del proyecto, así como posibles líneas futuras a abordar, con un resumen de lo abordado y una evaluación crítica y personal de los objetivos alcanzados, limitaciones que se han encontrado e introducción a posibles siguientes pasos a tomar para seguir extendiendo el proyecto.

Capítulo 2

Estado del arte

Este proyecto incluye el uso de diversos protocolos y tecnologías que forman un sistema integrado de comunicación segura entre dispositivos IoT. Por tanto, en este capítulo se describen dichos protocolos, tecnologías y los trabajos previos relacionados con el mismo.

Al ser un campo relativamente reciente no hay tantos proyectos y ni desarrollo del tema a diferencia de lo que puede ser la comunicación entre dispositivos de mayor potencia. Es por ello por lo que el desarrollo de dicho tema se basa en conceptos menos conocidos en el mundo de la tecnología y criptografía. Para ello se están trabajando en diferentes estándares como pueden ser CoAP, el cual toma parte en este proyecto, MQTT (*Message Queue Telemetry Transport*), AMQP (*Advanced Message Queuing Protocol*), IP6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*), etc. Todo ello con el fin de abarcar y extender más los posibles estándares a abordar con el fin de crear diversos sistemas con cada uno unas características particulares para diferentes aplicaciones.

La comunicación de este proyecto, realizada bajo el protocolo CoAP (sección 2.1), ha sido derivada en diferentes lenguajes de programación, aunque eso no ha provocado que haya numerosos estudios sobre ello.

Aunque CoAP propone el uso de DTLS para asegurar la comunicación entre las partes, recientemente se ha especificado otro mecanismo que permite proporcionar seguridad a nivel de aplicación, denominado OSCORE (sección 2.2). Al ser un tema mucho más específico, sucede lo mismo que con el protocolo CoAP, gran parte de la investigación pública del apartado se encuentra en trabajos de estudiantes o algunas empresas extranjeras que se están iniciando en el mundo de las IoT. A pesar de ello, cabe pensar, que una vez las comunicaciones y tecnologías IoT estén más extendidas y sean más comunes, y por ello se produzca un mayor desarrollo.

Como método de autenticación y derivado de claves usaremos una PUF (sección 2.3), función física no clonable. Al ser un tema distinto al anterior, hay mucho más desarrollo e información sobre ellos es utilizado como método de autenticación en numerosos proyectos, debido a la versatilidad del concepto y sus múltiples opciones a la hora de usarse junto a otros proyectos, apoyándose muchos de ellos en una SRAM para su realización, encontrando menos desarrollo en otros tipos de memorias. Otras aproximaciones a la PUF fueron consideradas, incluso se contactó con el autor de una de estas variantes, pero se acabó descartando por la de la SRAM, la cual se puede realizar con una sencilla lectura mediante la utilización de la interfaz SPI, *Serial Peripheral Interface*.

En conclusión, el campo de las IoT, al igual que todo lo que lo rodea, están comenzando a expandirse y darse a conocer, poco a poco, de la mano del progresivo desarrollo de dichas tecnologías para el uso convencional y más cercano a la sociedad.

Y esto será posible cuando las IoT dejen de verse por la sociedad como algo del futuro si no algo de nuestro presente, como puede ser la implementación de la domótica en la casa, y es por ello por lo que hay que promover la investigación sobre el tema y desarrollar este tipo de proyectos, con el fin de convertir el mundo IoT en algo con lo que interactuemos en el día a día.

2.1.CoAP

CoAP [7], *Constrained Application Protocol* (Protocolo de Aplicación Restringido), es un protocolo de comunicación diseñado para M2M (*Machine-to-Machine*) con dispositivos simples y funcionalidades muy restringidas, como puede ser un sensor de temperatura o luz.

Este protocolo de comunicación se basa en el intercambio de mensajes a través de UDP entre diferentes *endpoints*. Dichos mensajes son de carga ligera, diseñados para poder ser transmitidos por dispositivos que manejan recursos también ligeros, pudiendo incluirse en los mensajes diferentes opciones con los que complementar y extender su funcionalidad, así como diferentes tipos de mensaje entre los diferentes que se ofrece, como puede ser los mensajes confirmable o no confirmable.

Esta comunicación se realiza mediante el modelo *Request/Response*, similar al que implementa HTTP [7] siendo ambos protocolos RESTful (API y recursos similares), aunque con algunas diferencias, como:

- CoAP usa UDP mientras que HTTP usa TCP, a pesar de eso CoAP tiene algunas funcionalidades como puede ser la confirmación de los mensajes.
- Mientras que HTTP es síncrono, el protocolo CoAP es asíncrono.
- El encabezado se codifica en forma binario en CoAP para optimizar el espacio, en cambio en HTTP se envían directamente.

Entre las ventajas de CoAP se encuentran:

- En una comunicación con dispositivos IoT es necesaria la utilización de un protocolo que sea ligero debido a que su memoria y recursos son escasos, lo cual CoAP lo cumple con unos costes de encabezado bajos.
- CoAP también posee una arquitectura conocida, Cliente/Servidor, lo que permite que sea un protocolo sencillo de implementar.
- Permite la integración con IPv6 mediante la utilización de 6LoWPAN, un estándar que permite el uso de IPV6 sobre redes inalámbricas de bajo voltaje
- Integración sencilla con HTTPS, ya que como hemos indicado previamente, comparten la misma API.

- Posibilidad de utilización del CoAP como proxy y de hacer caching, aunque esto varía con HTTP, pudiendo utilizar la función en función del código de response, no del método del *Request* como en HTTP.
- Permite hacer tanto *multicast* (comunicación con varios dispositivos a la vez) como *unicast* (con un solo dispositivo).

Obviamente también podemos encontrar desventajas en este protocolo:

- Ausencia de un modelo estandarizado, lo cual dificulta complementar diferentes dispositivos que usan diferentes modelos.
- CoAP utiliza el protocolo UDP en la transferencia de mensajes, esto puede provocar la pérdida de paquetes, y aunque implementa un método de *Request* de un *acknowledgement*, este no garantiza que se haya recibido el mensaje entero o una correcta decodificación de dicho mensaje.
- CoAP es un protocolo que no está encriptado, lo que permite que un atacante pueda leer su contenido fácilmente.

2.1.1. Mensaje CoAP

CoAP está desarrollado para el intercambio de mensajes ligeros y compactos, codificados de forma binaria. La transmisión de dicho mensaje se lleva a cabo a través UDP de manera predeterminada, aunque puede considerarse su implementación en otros protocolos como pueden ser TCP, SCTP o DTLS. El tamaño del mensaje es de 32 bits. El mensaje CoAP posee un formato sencillo:

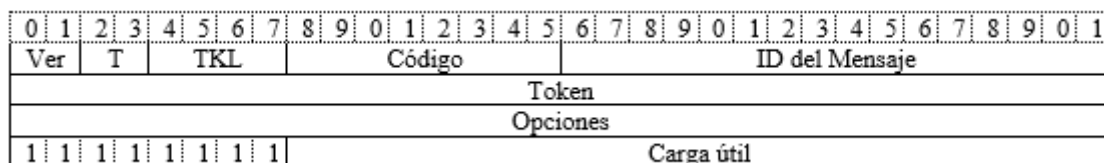


Fig. 2.2 Formato del mensaje CoAP (32 bits)

Este mensaje, como observamos en la Figura 2.2, se compone de diferentes partes, las cuales serán descritas:

- Versión (Ver) → (2 bits) Versión en la que opera CoAP.

- Tipo (T) → (2 bits) Tipo de mensaje, en este caso encontramos cuatro tipos:
 - Confirmable (00) → Se trata de un mensaje en el que requiere una respuesta de confirmación.
 - Non-Confirmable (01) → Se trata de un mensaje que no requiere una respuesta de confirmación. Esto puede ser útil para los mensajes que se mandan de forma repetida.
 - Acknowledgment (10) → Se trata de un mensaje de confirmación en respuesta a un Confirmable message (00).
 - Reset (11) → Se usa para indicar que un mensaje (ya sea confirmable o no) ha sido recibido, pero que ha llegado de forma incorrecta o faltando parte de contenido.

- Token Length (TKL) → (4 bits) Indica la longitud del campo Token.

- Código → (8 bits) Permite identificar qué clase de mensaje se manda, entre los que se pueden identificar:
 - Request → Solicitud que le hace un cliente a un servidor.
 - Response → Respuesta de un servidor a un cliente consecuente a un Request.
 - Client error response → Mensaje de error de un cliente a un servidor.
 - Server error response → Mensaje de error de un servidor a un cliente
 - Empty → Indica que el mensaje está vacío.

- ID del mensaje → (16 bits) ID único respecto a cada mensaje que se usa para poder asegurar que no hay mensajes duplicados.

- Token → (32 bits) El valor de este campo nos permite asociar *Requests* con *Responses*.

- Opciones → (32 bits) Nos permite definir la opción que se aplicara en el mensaje CoAP. Como definiremos más tarde, en este proyecto aplicaremos la opción nueve, la que nos permite determinar el mensaje CoAP con la opción OSCORE.

1	If-Match
3	Uri-Host
4	ETag
5	If-None-Match
6	Observe
7	Uri-Port
8	Location-Path
9	OSCORE
11	Uri-Path
12	Content-Format
14	Max-Age
15	Uri-Query
17	Accept
20	Location-Query
23	Block2
27	Block1
28	Size2
35	Proxy-Uri
39	Proxy-Scheme
60	Size1
258	No-Response

Fig. 2.3 Opciones de CoAP (Opción OSCORE resaltada)

- Carga útil → (32 bits) Datos transmitidos en el mensaje CoAP.

Como observamos, el mensaje CoAP, es bastante simple y ligero, lo cual contribuye a la necesidad de las comunicaciones IoT de ser livianas, con solo los campos necesarios para abarcar la comunicación.

2.1.2. Tipos de peticiones CoAP

A pesar de que CoAP utiliza un sistema de cliente/servidor, este se basa en la arquitectura REST, lo que le permite ser compatible con la comunicación HTTP, es por ello por lo que utiliza métodos definidos en HTTP.

Esto son el método GET, que permite pedir información en particular al servidor, esto se puede llevar a cabo mediante la obtención de un recurso del servidor , el método POST,

permite enviar información al servidor, esto nos permite crear un recurso, el cual tendrá la información que será mandada al servidor; el método DELETE , con el que el cliente puede solicitar al servidor que elimine determinada información, de esta manera, una vez invocado este método, con la confirmación del servidor, se procede a borrar un recurso, y por último el método PUT, con el que el cliente puede actualizar determinada información, de esta manera, un recurso previamente actualizado puede ser actualizado.

2.1.3. Ataques en CoAP

CoAP, como muchos protocolos de comunicación presenta vulnerabilidades a los que tiene que hacer frente, estos ataques pueden ser críticos pudiendo llegar a provocar daños catastróficos.

El ataque más común es el de *Man-In-The-Middle*. En este caso, un dispositivo intercepta las comunicaciones entre dispositivos, como puede ser la comunicación de un servidor con un sensor, en el que recaba los datos que recoge el sensor, pudiendo desvelar información confidencial, o modifica las mediciones obtenidas por el sensor, que como hemos determinado, puede llegar a provocar muchísimo daño, al poder provocar un fallo total del sistema.

Las técnicas que se suelen emplear en este ataque suelen ser ataques DoS (*Denial of Service*), en la que impide que se conecte el cliente con el servidor, robo de identidad; *Spoofing*, permitiendo pasarse por el dispositivo frente al servidor; *Sniffing*, como hemos introducido antes, puede provocar un robo de datos; *Hijacking*, secuestro de un dispositivo, pudiendo adueñarse de un dispositivo de forma física o no. También hay técnicas menos comunes, como puede ser *Cross-Platform attack*, en la que se infecta el cliente, pero no le provoca daños, sin embargo, transmite la información corrupta al servidor, el cual si acaba siendo dañado.

Todos estos ataques son comunes también en otros tipos de comunicaciones, pero debido a la falta de robustez que las comunicaciones CoAP suelen poseer, provoca que, si no se le aplica una capa de seguridad externa a la comunicación, estos ataques sean muy fáciles de llevar a cabo. Es por eso por lo que, con el fin de minimizar el daño, es necesario tener

un planteamiento claro de los vectores de ataque en la comunicación, y qué elementos pueden ser explotados con el fin de dañar dicho intercambio de recursos.

2.1.4. Diferentes implementaciones de CoAP

CoAP está presente en muchos sistemas, lo cual ha conllevado la necesidad de adaptar este protocolo de comunicación a las diferentes situaciones en la que se intente llevar a cabo una comunicación de mensajes CoAP. Es por ello por lo que existe dicho protocolo desarrollado en diferentes lenguajes de programación, lo cual permite que su integración en cualquier sistema sea más sencilla y personalizada, debido a que el desarrollo del protocolo en el lenguaje de programación requerido reduce costes y tiempo de integración, a la par que aumenta la productividad y eficiencia.

Las principales implementaciones de CoAP son: *aiocoap* [8], desarrollado en Python 3, *CoAPthon* [8], para Python, *libcoap* [8] programado en C y la que integramos en este proyecto, *Californium*, desarrollado en Java y que describiremos a continuación. También podemos encontrar otras implementaciones como pueden ser: CoAP.NET en C#, Go-CoAP en GO, node-coap en Javascript o SwiftCoAP en Swift.

Solo hemos mencionado algunas de las múltiples variantes que existen del protocolo, lo cual nos da una visión de lo extendido que está el uso de dicho protocolo en dispositivos IoT, así como la posibilidad de crear diversos sistemas dispares, usando dicho protocolo implementado en un lenguaje de programación u otro, convirtiendo a CoAP en una opción predilecta para desarrollar dicha comunicación entre dispositivos IoT.

2.1.5. Californium

Californium es un *framework* de código abierto que implementa el protocolo CoAP, programado en Java, creado por Eclipse cuyo objetivo es interactuar con el back-end de una IaaS (Infraestructura como Servicio), que es la infraestructura que brinda recursos bajo demanda necesarios para el almacenamiento, creación de redes y ejecución de procesos, lo que lo convierte en un candidato perfecto para las comunicaciones IoT en la nube, debido a su alta escalabilidad que permite superar la eficiencia de servidores de

HTTP de alto rendimiento, además, los costes de la comunicación CoAP y la flexibilidad del Californium le permite aplicarse en aplicaciones muy diversas, facilitando su expansión.

Este *framework* está centrado en entornos sin restricciones como proxies, directorios de recursos, necesarios para localizar servicios y programas, y servicios en la nube, algo en lo que el IoT está muy presente y otros entornos con pocas restricciones como pueden ser la domótica de una casa.

Californium se compone de diversas librerías que permite cada una implementar diferentes funcionalidades, entre ellas podemos encontrar las dos que usaremos, que será la *core*, en la que está la implementación de CoAP en su modelo más básico y *cf-oscore*, la cual contiene la implementación propia del protocolo OSCORE. También encontramos otras librerías con las que podemos implementar los *proxies*, así como múltiples librerías de *prueba* de las diferentes configuraciones. En aquellas librerías que utilizaremos disponemos de varios servidores y clientes de prueba, los cuales serán utilizados más adelante, serán extendidos y modificados.

El objetivo de Californium es de proporcionar un servicio con una facilidad de escalabilidad en una aplicación IoT, con todas las funcionalidades y extras, como un modelo de publicación para CoAP, introducción de proxies, entre los que incorpora proxies de reenvío, en la que recoge todas las solicitudes que se realizan a la red y las transmite en su nombre al servidor, de inversión, donde realiza el proceso contrario, recogiendo los recursos del servidor y retransmitiéndolos al cliente en su nombre y de plataformas cruzadas, algo esencial en CoAP, permitiendo la comunicación entre CoAP y HTTP, haciendo un mapeo de los mensajes para que sean entendidos entre ellos.

También CoAP permite incorporar DTLS (*Datagram Transport Layer Security*), lo cual permite añadir una capa de privacidad y seguridad en las comunicaciones CoAP, un añadido esencial, ya que, como hemos nombrado antes, CoAP no posee un buen nivel de privacidad, lo que permite realizar con facilidad ataques a dicha comunicación, pudiendo leer e interceptar las comunicaciones.

En este proyecto, aplicaremos una adición a la capa de CoAP, aplicando en la comunicación OSCORE, que como explicaremos más tarde, es un método de protección de la capa de la aplicación.

Californium, a diferencia de otras implementaciones de CoAP, introduce una comunicación entre cliente y servidor, bastante común pero basada en una API REST, con intercambio de recursos mediante instrucciones semejantes a las de HTTP, lo que, como nombramos antes, facilita su integración con este protocolo y su interpretación.

2.1.6. CoAP vs MQTT

Por último, compararemos a CoAP con otro protocolo, también bastante interesante, como es MQTT. Aunque CoAP es un buen protocolo para la comunicación existen otros diferentes, que están diseñados para otro tipo de situaciones. El más conocido es MQTT, el cual es bastante diferente a CoAP. Mientras que CoAP se basa en una arquitectura de cliente y servidor, este en cambio utiliza el modelo de suscripción-publicación. En este modelo, los clientes se suscriben a un Broker, con el fin de recibir sus mensajes, los cuales son publicados por el Broker.

Entre sus mayores diferencias se encuentran:

- MQTT utiliza TCP frente a CoAP que usa UDP
- El modelo de comunicación desde x brokers con y clientes, a diferencia de CoAP, de un cliente con un servidor
- MQTT tiene un API diferente a CoAP, por lo tanto, no es RESTful, utilizando otro tipo diferente de mensajes
- Mientras que CoAP no tiene una estandarización, esta sí que se ha alcanzado con MQTT

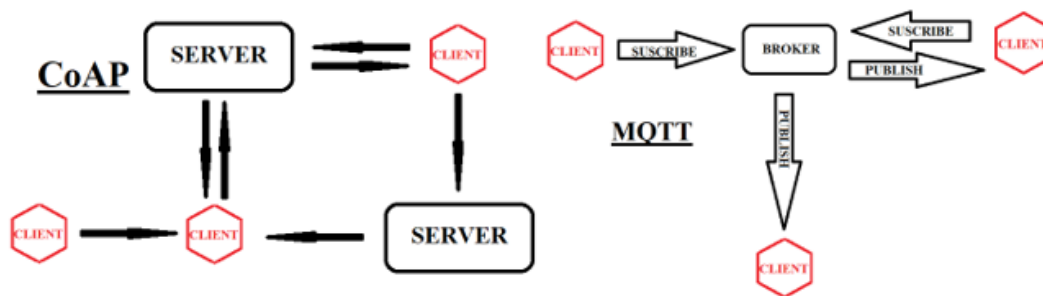


Fig. 2.1. Comparación de arquitectura de CoAP frente a MQTT

Como observamos en las diferentes arquitecturas de la Figura 2.1, en lo referente a CoAP, observamos dos servidores intercambiando mensajes con tres clientes, en los que dos están con una comunicación multiplexada de extremo a extremo con un servidor, mientras que el otro servidor se comunica con un cliente y recibe información de otro. Por último, observamos, como un cliente se comunica a través de otro cliente con el servidor.

Por el otro lado, analizando la arquitectura de MQTT, podemos observar la presencia de un *Broker*, al que los clientes se suscriben para leer la información, así como dicha información es publicada por el mismo *Broker* para que esté al alcance del cliente.

Como conclusión, MQTT está diseñado para usarse en una comunicación entre varios dispositivos a la vez mientras que CoAP está diseñado para una comunicación peer-to-peer.

2.2. OSCORE

El crecimiento exponencial de las tecnologías IoT ha traído muchas ventajas en numerosas industrias, como la electrónica, energías, biomédica, transporte, etc. Esto ha supuesto un crecimiento exponencial de las amenazas y ciberataques, que puede conllevar de fallos mínimos a situaciones catastróficas. Esto ha llevado a que la seguridad en el IoT sea una prioridad en las comunicaciones. Para ello se han llevado a cabo numerosos desarrollos con el fin de aplicar una capa de seguridad a este intercambio de recursos, es

por ello por lo que OSCORE fue especificado en la RFC 8613 [9], con el fin de proteger dichas comunicaciones.

OSCORE está orientado a ambientes muy restringidos, creado como extensión de CoAP, otorgando protección e integridad a dichas comunicaciones de extremo a extremo, en la cual protege parte del mensaje, que es la carga útil del mensaje, sus opciones y los códigos. Esto es debido a la naturaleza de bajo consumo de los dispositivos IoT, en la que proteger el mensaje completo podría suponer un aumento desmesurado de costes y efectividad en las comunicaciones.

2.2.1. OSCORE: La seguridad de CoAP

La comunicación CoAP puede ser de dos maneras: protegida o no protegida. Mientras que su valor por defecto es sin proteger, añadirle seguridad es necesario. La funcionalidad OSCORE se activa indicando la opción nueve en el mensaje como indicamos previamente en la Figura 2.3 donde se analizan las diferentes opciones de CoAP, en el campo *Options*, campo definido en la sección 2.1.2, donde se definían los diferentes campos del mensaje CoAP. OSCORE incorpora el formato CBOR (*Concise Binary Object Representation*) [10], que es un formato de mensaje diseñado para codificaciones extremadamente ligeras y mensajes de tamaño pequeño. Y el protocolo COSE (*CBOR Object Signing and Encryption*) [11] con el que abarca la funcionalidad de cifrado y derivación de claves a través de un contexto común en el que basar dichas claves.

Como protocolo de seguridad, OSCORE tiene numerosas funcionalidades aparte de proteger exclusivamente los mensajes CoAP. Permite proteger también intercambios de recursos CoAP-HTTP, de un extremo a otro de las comunicaciones. También certifica una comunicación segura entre proxies CoAP y proxies multiprotocolo (CoAP y HTTP), mencionados anteriormente en la sección 2.1.6 en la que se definía la implementación de Californium. A parte de soportar otras opciones de CoAP como pueden ser:

- *Observe* [12], en la que puedes obtener una representación de un recurso y su variación a lo largo del tiempo

- *No Server Response* [13], donde puede determinar el cliente al servidor su desinterés de forma explícita por todas las respuestas que le pueda proporcionar el servidor a una solicitud en particular, así como métodos comunes entre HTTP y CoAP como son PATCH, cuya función es la de realizar determinados cambios por parte del cliente en el servidor de recursos ya creados, y FETCH, en el que permite al cliente acceder a diferentes recursos del servidor de manera asíncrona.

Al ser OSCORE una opción de CoAP, las diferentes opciones aplicables a CoAP son también compatibles cuando la opción OSCORE es utilizada en el mensaje CoAP, sin ningún problema de compatibilidad, en cambio, en HTTP, para indicar la utilización de OSCORE, se crea un nuevo encabezado al mensaje HTTP. Esto permite derivar un mensaje HTTP-CoAP en un mensaje OSCORE y viceversa. Para ello, primero, se realiza un mapeado de dicho mensaje HTTP-CoAP en un mensaje simplemente CoAP, y, tras ello, con el fin de obtener un mensaje OSCORE a partir de dicho mensaje CoAP, este se traduce mediante la utilización del objeto COSE, con el que se proporciona los medios necesarios para la encriptación y protección del mensaje, finalizando el proceso.

Es por ello por lo que podemos determinar que OSCORE, protege intercambios de recursos que utilizan el api RESTful, entre los que se encuentran la solicitud de recursos y la respuesta a estos, la carga útil del mensaje, dejando sin proteger parte del mensaje CoAP, el Token, al este variar constantemente. Tampoco protege la capa de mensajería por donde se transmiten los mensajes CoAP, y es por eso por lo que OSCORE puede ser activado tanto en métodos de comunicación seguros (DTLS/UDP) como inseguros (UDP).

Capa de Aplicación		
Solicitudes	Respuestas	Señalización de la comunicación
OSCORE		
Capa del mensaje		
Capa de transporte (TCP, UDP, ...)		

Fig. 2.4. Mensaje CoAP (en gris) con la extensión OSCORE

El protocolo OSCORE usa un método de cifrado con características similares a otros métodos usados para mensajes más complejos. Utiliza claves que han sido compartidas de antemano a la comunicación, las cuales pueden ser derivadas de manera independiente o con un protocolo de para establecer las claves. También proporciona integridad en la comunicación, cifrado extremo a extremo y nos permite relacionar una solicitud de un recurso con la respuesta obtenida.

Como observamos en la Figura 2.4, observamos el flujo de protocolo, similar al de CoAP, de *Request* y *Response*, a diferencia del proceso inicial de derivación de las claves, a través de los diferentes contextos como abordaremos en la siguiente sección.

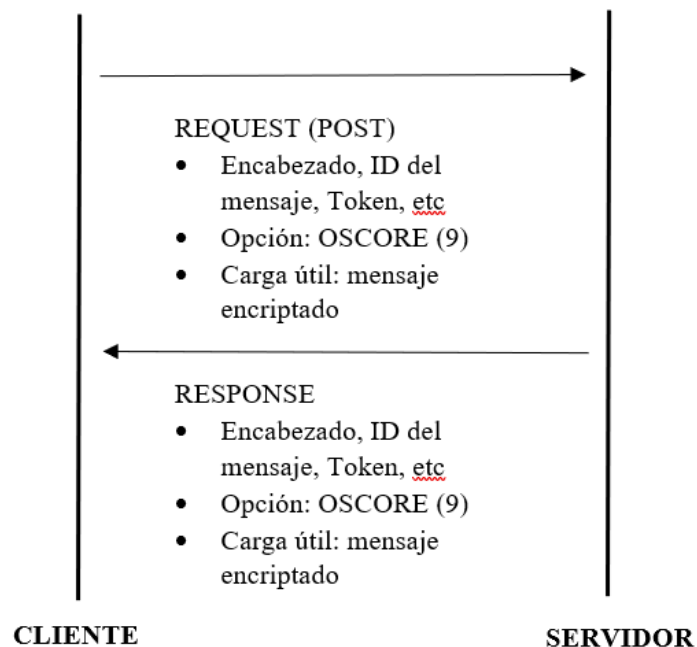


Fig. 2.5. Comunicación OSCORE

2.2.2. Contexto de seguridad de OSCORE

El contexto de seguridad se trata de todos los elementos necesarios para llevar a cabo la comunicación entre el servidor y el cliente operando con OSCORE. Este contexto de seguridad sirve para proteger los mensajes, verificar su integridad y autenticarlos con el fin de conseguir una comunicación segura de extremo a extremo, habiendo en cada uno de los puntos finales de la comunicación diferentes contextos de seguridad, el del emisor

y el del receptor. Ambos contextos, emisor y receptor, son inicialmente derivados de un contexto común a ambos.

Estos contextos son utilizados constantemente por el servidor y sus clientes, los cuales protegen los mensajes usando el contexto emisor y los verifica usando el contexto receptor, teniendo que usar el contexto correcto en cada acción, si no, el protocolo no funcionara. A pesar de sus similitudes, cada uno posee sus valores propios.

Como hemos nombrado antes, inicialmente solo encontramos el contexto común, el cual posee diferentes parámetros, como son:

- Algoritmo COSE AEAD (*COSE Authenticated Encryption with Associated Data*) es el algoritmo de cifrado de la información.
- Secreto maestro, secuencia de bits aleatorios de longitud variable de la que se derivan las claves de emisor y receptor, así como el vector de inicialización.
- Valor aleatorio del contexto común, secuencia de bits, que, junto al secreto maestro, es usada para derivar las claves y el IV. Es opcional en el sistema, pudiendo derivarse los diferentes parámetros del secreto maestro directamente.
- Vector de inicialización, secuencia de bits derivada del secreto maestro y la sal maestra, cuya longitud viene indicada en el algoritmo COSE AEAD.
- La función de derivación de la clave, basada en un código de autenticación de mensajes en clave-hash. De ella es derivada la clave del emisor, la clave del receptor y el vector de inicialización (IV) común a ambos.

Una vez establecido el contexto de seguridad, ninguno de los valores puede ser modificado, permaneciendo sus valores a lo largo de la comunicación.

Además del contexto común encontramos los contextos derivados, el emisor y el receptor, los cuales tienen gran parte de los componentes comunes, pero diferentes valores dependiendo de si el contexto es uno u otro. Estos parámetros son:

- El ID, cuya función es la de identificar cual es el contexto usado en ese momento, que nos permite garantizar que el *nonce*, el cual nos permite

identificar si un mensaje ha sido manipulado, es único. Dentro de un contexto, ya sea emisor o receptor, no podemos encontrar dos ID iguales.

- La clave, secuencia de bits, es simétrica, es decir, es igual entre un contexto emisor y receptor pertenecientes a una única comunicación

A parte de los parámetros nombrados podemos encontrar parámetros únicos dependiendo del contexto.

En la parte del emisor podemos encontrar una secuencia de números, utilizado como un vector de inicialización parcial con el que generar *nonces* únicos a partir del COSE AEAD, su función es la de proteger las solicitudes realizadas por el emisor, así como las notificaciones en las que se obtiene actualizaciones sobre los recursos ya solicitados. Por parte del receptor encontramos la ventana de reproducción, opción compatible solo si es un servidor, permite al mismo verificar las solicitudes recibidas.

Como con el contexto común, una vez establecidos los contextos emisor y receptor, no pueden ser modificados en toda la comunicación.

2.3. Funciones físicas no clonables (PUF)

Las funciones físicas no clonables podrían tratarse como el elemento identificativo de un dispositivo. Este procedimiento se aprovecha de las imperfecciones producidas durante la fabricación de memorias como SRAM (*Static Random Access Memory*), DRAM (*Dynamic Random Access Memory*), ReRAM (*Resistive Random Access Memory*), MRAM (*Magnetoresistive Random Access Memory*), EEPROM (*Electrically Erasable Programmable Read-Only Memory*), etc. para extraer los bits estables de cualquiera de ellas. Estas imperfecciones sirven como un concepto semejante a la huella dactilar, ya que es imposible crear dos memorias con las mismas imperfecciones, por lo que son bastante útiles para la derivación de llaves maestras, autenticación y más. Este mapa de la memoria es único por cada una, y es este detalle el que nos permite discernir con qué dispositivo estamos entablando una comunicación, y, además, al tratarse del

hardware de la memoria, es por ello por lo que supone un método muy simple para deducir cuál es el dispositivo. Esta característica es debida a las imperfecciones de la memoria [15].

Toda memoria, en el estado de *power up*, se inicia con unos valores. En un caso ideal, el valor al que se inicie debería ser totalmente aleatorio, es decir, en cada celda de la memoria debería haber un cincuenta por ciento de veces en las que su valor de inicio sea un lógico '1' y el otro cincuenta la celda tuviera el valor de un lógico '0'. Pero esto no se da, ya que, a la hora del proceso de manufacturación de las memorias, se introducen de manera no intencionada imperfecciones en sus elementos semiconductores. Esto provoca que ya no sea un cincuenta por ciento el número de veces que obtenemos cada valor, si no que empieza a tender hacia un lado u otro, pudiendo ser la mayoría el lógico '1' o el lógico '0'. Esta es la cualidad que se utiliza a la hora de formar un dato inherente al dispositivo. Esta tendencia de las celdas es evaluada repetidas veces en el inicio del dispositivo, con ello podremos extraer cuáles son sus bits más estables, es decir, los bits que tienen un mayor porcentaje de un valor de los dos posibles. Estos valores se extraen, pudiendo crear una secuencia de bits única para esa memoria.

2.3.1. Propiedades

Las PUFs deben cumplir determinadas propiedades para considerarlas como tal, estas propiedades también contribuyen a su establecimiento como sistema de seguridad, aunque, dichas propiedades pueden no siempre darse [36].

- **Única:** Una PUF debe ser única, es decir, no debe ser repetida entre dos dispositivos. Esto significa que, para una misma entrada en diferentes dispositivos, la salida de la PUF debe ser diferente siempre entre dispositivos.
- **Reproducible:** Una PUF debe ser siempre la misma. Siempre que se introduce una entrada en la PUF, debe devolver la misma salida, es por ello por lo que se seleccionan los bits estables a la salida, con el fin de obtener una salida fija.

- **Impredecible:** La salida de la PUF nunca debería ser precedida independientemente de la similitud de dos entradas, con el fin de que un atacante no pueda reproducir un modelo de la PUF.
- **No clonable:** El principio de las PUF es aprovechar las imperfecciones de la memoria con el fin de poder obtener determinados bits estables, pero estas imperfecciones no son provocadas y son totalmente aleatorias, lo que implica que ni siquiera el fabricante podría reproducir dichas imperfecciones, aunque se lo propusiera. Para ser una PUF perfecta tampoco debería poder replicar el modelo, aunque tuviera acceso a un número ilimitado de entradas y salidas. Cuantos más pares tenga, más complicado sería para un atacante poder descifrar el modelo.

2.3.2. Tipos de PUFs

2.3.2.1. Arbiter PUF

Este tipo de PUF se basa en la creación de dos caminos en paralelo dentro de un chip, regulados por un circuito que establece cuál de los caminos es completado antes. Estos circuitos son realizados de manera simétrica, pero debido a imperfecciones en la fabricación del chip, el retardo en ambos caminos será único, generando una respuesta imposible de deducir [36].

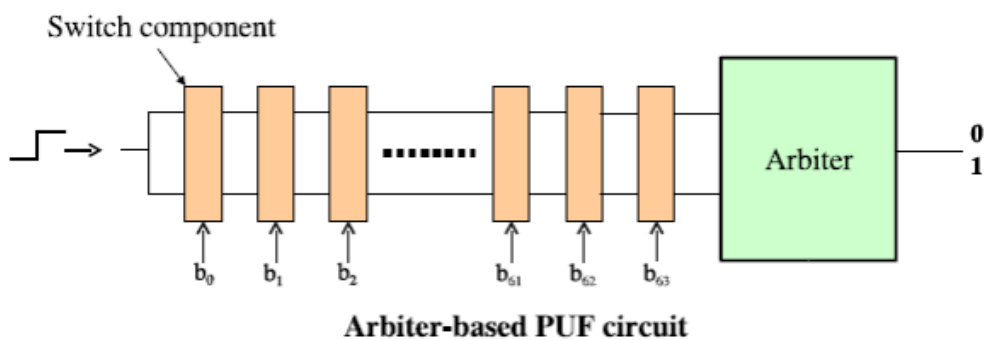


Fig. 2.6. Diagrama de un Arbiter PUF [36]

2.3.2.2. SRAM PUF

La SRAM posee ciertos valores de inicio, estos son obtenidos en el estado de inicio de la memoria, los valores de las celdas en la memoria varían en función de la predisposición

de tender al lógico '1' o '0', y es su preferencia por uno de los estados lo que nos permitirá una salida totalmente aleatoria e impredecible. Esta configuración se alcanza mediante la utilización de dos inversores conectados entre sí [36].

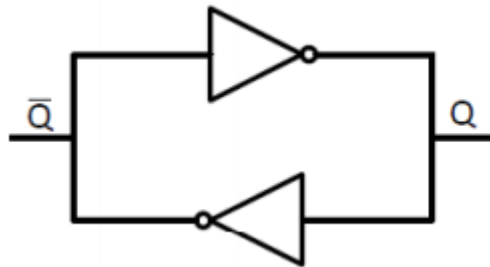


Fig. 2.7. Diagrama de una SRAM PUF [36]

2.3.2.3. Butterfly PUF

Utiliza el mismo concepto de la SRAM PUF, pero resuelve diversos inconvenientes que surgen con ella, en vez de depender del momento de encendido de la memoria para obtener los valores de los bits estables, estos son conservados, si poder sobrescribirse como en la PUF, también no depende de ninguna propiedad física, por lo que su valor es totalmente aleatorio [36].

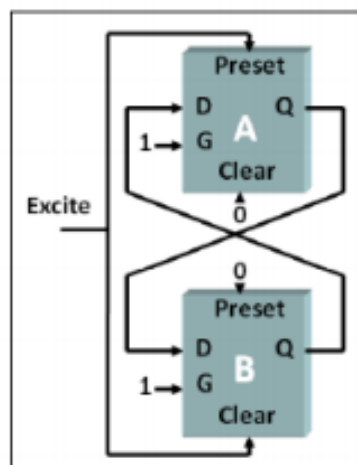


Fig. 2.8. Diagrama de una Butterfly PUF [36]

2.3.3. Autenticación mediante la PUF

El proceso de autenticación mediante una PUF es sencillo, consta de dos pasos en los que se desarrolla todo el proceso, primariamente la inscripción, para después realizar la autenticación.

2.3.3.1.Registro (*Enrollment*)

El *Enrollment* es el proceso mediante el cual un nuevo dispositivo se inscribe a un servidor. Primariamente el dispositivo (cliente) extrae su PUF correspondiente. Una vez extraído los bits estables, inicia la comunicación con el servidor, ya puede ser mediante sockets (conexión dedicada entre servidor y cliente) o mediante *endpoints* (el cliente se conecta a un servidor en una dirección que está constantemente escuchando). Envía esta información a un servidor, junto a un identificador del dispositivo, como puede ser un número de serie, para poder determinar a quién pertenecen esos bits. Este envío de bits se realiza en un entorno seguro, esto es así ya que, si un atacante tuviera acceso a esos bits de forma directa, podría deducir por su cuenta cual es la PUF. Cuando el servidor recibe la información, lo almacena, ya sea en un fichero o base de datos, con el nuevo dispositivo, esta información le corresponde a el dispositivo, y como vimos antes en las propiedades de las PUFs, es única, por lo que nos aseguraremos de que no pueda haber dos *Enrollments* semejantes.

2.3.3.2. Autenticación (Authentication)

El proceso de autenticación se realiza después del proceso de inscripción. En este caso el cliente quiere comunicarse con el servidor. Para ello, primariamente, el cliente le manda al servidor el identificador que había registrado en la inscripción, el cual, en caso de que el servidor compruebe que está en su lista de dispositivos inscritos, lanza un reto al cliente, que, una vez recibido por el cliente, analizando su propia PUF, determina la respuesta a dicho reto, y la reenvía al servidor.

Durante el mismo proceso, el servidor responde a su propio reto, con el cual obtiene una respuesta en particular, y, una vez recibida la respuesta del cliente, las compara ambas. En caso de que el cliente haya formulado una respuesta correcta, el servidor dará por concluida la autenticación con un resultado positivo, es decir, el servidor habrá sido capaz de dictaminar que el cliente es quien dice ser, en caso de que las respuestas no coincidan, y como una de las propiedades de la PUF es la capacidad de reproducirse exactamente igual, el servidor deducirá que el cliente no es quien dice ser y por tanto no se producirá la autenticación correcta.

Este intercambio podrá hacerse tanto en un entorno seguro como no seguro, lo cual es debido a que la respuesta no es la información literal de lo que incluye la función, si no la información formateada, como puede ser mediante el hash de la función. Este proceso se puede hacer incluso más seguro mediante la introducción de elementos aleatorios en las respuestas, como puede ser una combinación de los bits requeridos y una secuencia aleatoria.

2.3.3.3. Parejas Reto-Respuesta (*Challenge-Response*)

Los bits recibidos en el servidor son guardados, y, para cada dispositivo, son únicos. Esta información pertenece a las direcciones de memoria de la SRAM, la cual, por cada dirección, guarda un número de bits estables. Dichas parejas *Challenge-Response* no son un sistema uniforme, el servidor puede determinar diversas maneras de autenticarte, desde un byte que podría corresponder a una dirección de memoria como una sección entera de la memoria o incluso la memoria entera, es por ello, por lo que podemos determinar que la PUF es un concepto robusto dentro de la seguridad, al tener bastante margen de flexibilidad. También dicho par del reto y la respuesta pueden ser moldeado a gusto del programador, ya que puede ser configurado a gusto, desde enviar la información que te pide el servidor tal cual, lo cual podría ser bastante inseguro a, por ejemplo, hacer un XOR de tu secuencia con un número aleatorio proporcionado por el servidor, y tras ello mandar un HASH de lo obtenido. Es por ello donde brilla la PUF, al permitir múltiples maneras de realizar la misma tarea, y, por lo tanto, provocando una mayor complejidad a la hora de intentar romper la PUF.

2.3.4. Derivación de la clave maestra

Existen diversos mecanismos de almacenamiento de claves, estos suelen tratarse de una memoria, ya sea física o en la nube, donde son guardadas las claves, pero suelen ser bastante complejos. Esto puede ser debido a numerosos elementos, como la necesidad de proteger la memoria al contener información de crítica o la propia estructura de almacenamiento de claves. También, uno de los inconvenientes suele ser lo costosos que llegan a ser, tanto a nivel monetario como de esfuerzo, ya que supone implantar un

sistema complicado, al que un atacante no pueda acceder fácilmente, o adquirir un sistema ya creado para incorporarlo a tu memoria, pero dichos sistemas son muy caros. Es por ello por lo que la introducción de las PUFs como sistema de derivación de claves suponen una gran ventaja, así como un ahorro de costes y esfuerzo. En vez de almacenar la clave en una memoria, ésta es derivada durante el proceso, lo que además permite que sea usada en la ejecución de la comunicación pudiendo ser destruida tras ello, lo cual reduce el tiempo del atacante para deducir la clave.

Este proceso de derivación es bastante simple. Inicialmente se extraen los bits estables de la PUF, obteniendo una secuencia, la cual es necesaria pasarla por un código de corrección de errores, *Error Correction Code* (ECC). Esto es necesario debido a que la presencia de diferentes interferencias como puede ser el ruido puede provocar la presencia de diversos bits incorrectos. Este código entonces produce una información de ayuda, la cual es pública, por lo que no debería ser posible derivar la clave a través de esta información. También, mediante el ECC, es generada la clave con la que poder realizar la comunicación. Una vez finalizada la clave es destruida. Esta es la primera fase del proceso, la cual es denominada fase de generación o inicialización.

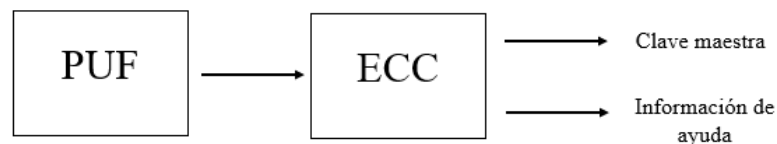


Fig. 2.9. Diagrama de generación inicial de la clave maestra

Una vez realizada esta primera generación de la clave, ya se posee la información de ayuda necesaria para poder volver a generar dicha clave. Cuando sucede esto, se entra en la siguiente y última fase, la de réplica de la clave. Primeramente, se vuelve a generar la salida de la PUF, la cual es procesada de nuevo por el código de corrección de errores, pero a diferencia de la primera fase, en este caso se utiliza la información de ayuda para corregir la salida del ECC. Gracias a esto podemos reproducir la clave inicial y con ello dar por finalizado el proceso de recuperación de la clave.

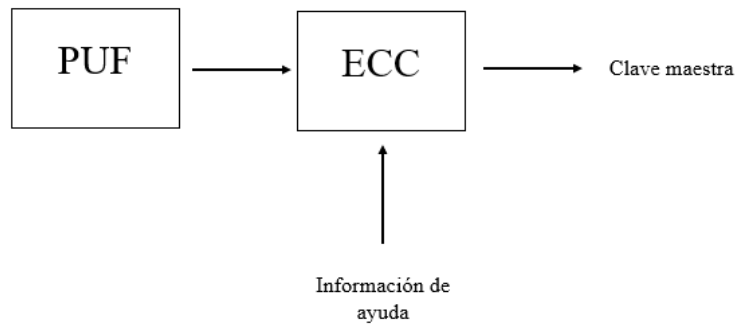


Fig. 2.10. Diagrama de reproducción de la clave maestra

2.4. Trabajos relacionados

El tema abordado en este proyecto es relativamente reciente, por lo que existen otras propuestas de investigación relacionados con autenticación basada en PUF, aunque no relacionados con OSCORE, como son:

- *PUF-based authentication*, el cual analiza las distintas amenazas y posibles ataques de una PUF, realizando diferentes experimentos en una FPGA Xilinx [37].
- *PUF Based Authentication Protocol for IoT*: extiende un poco más el concepto de una PUF, desarrollándolos algoritmos de manera más visual y práctica de autenticación y derivado de la clave.

Por otra parte, para la gestión de claves por parte de OSCORE, lo cual no forma parte de la especificación existen las siguientes propuestas, algunas de las cuales se están estandarizando dentro del IETF:

- *Application Layer Key Establishment for End-to-End Security in IoT*: en la que mediante la utilización de Compact EDHOC (*Ephemeral Diffie-Hellman Over COSE*), se extraen los parámetros del protocolo *core* de EDHOC, para autenticación mediante claves efímeras, solo pudiéndose usar en una sesión, para entornos restringidos [39].
- *Key Management for OSCORE Groups in ACE*: esta propuesta se centra en la gestión de claves para grupos, en la cual se delega la responsabilidad de inscribir y autenticar nuevos clientes a un servidor independiente de Gestión del Grupo de los nuevos clientes [40].

- *An Application-Layer Approach to End-to-End Security for the Internet of Things:* mediante la introducción de un dispositivo de confianza en el que se almacena la información del contexto, tanto el cliente como el servidor acceden a este dispositivo para reclamar dicha información [41].

Capítulo 3

Descripción del sistema

El sistema propuesto se divide en tres partes. La primera trata de la generación de la PUF en la Raspberry Pi. En la segunda se define el proceso de inscripción (*enrollment*) a partir de los datos obtenidos a la salida de la PUF, así como la derivación de la clave maestra. Para acabar, se extenderá la implementación de Californium para desplegar un cliente, en una Raspberry Pi, y un servidor en una máquina virtual, utilizando como clave maestra la clave generada en el apartado dos.

La arquitectura de este proyecto se conforma de un dispositivo IoT, en este caso la Raspberry Pi, que está conectado a una memoria SRAM de la que se extraen los bits estables. Estos bits entonces se envían como cliente a un servidor, alojado en una máquina virtual en un ordenador, completando la estructura del sistema. Dicho servidor permitirá realizar la inscripción y autenticación del dispositivo IoT a partir de los bits estables, y tras ello, un nuevo servidor de comunicación también alojado en la misma máquina virtual, con el que, tras la derivación de los diferentes contextos de comunicación, se realiza la comunicación entre el dispositivo IoT y el servidor de la máquina virtual.

3.1. Generación de la PUF

En este paso utilizaremos la Raspberry Pi 3 Model B como dispositivo IoT, también como memoria SRAM utilizaremos el modelo de circuito integrado 23LC1024, así como los diferentes cables y un condensador de 10nF. Ambos se comunicarán mediante un cableado para llevar a cabo la comunicación de ambos, realizándose las conexiones sobre una *protoboard*.

3.1.1. Raspberry Pi 3

La Raspberry Pi se trata de un ordenador del tamaño de una tarjeta, cuya finalidad suele ser la de crear circuitos electrónicos con los que puede realizar diferentes operaciones. Desde ella se realiza el programa de lectura de la SRAM. Como se puede observar, en su lateral derecho donde encontramos los pines, se encuentran los puertos de entrada y salida, GPIO. De los que podemos observar en la Figura 3.1., utilizaremos para este proyecto:

- 3v3 power (3.3V)
- GPIO 10 MOSI (*Master out – Slave in*)
- GPIO 9 MISO (*Master in – Slave out*)
- GPIO 11 SCLK (*SPI Clock*)
- GPIO 17 Manual CS (*Manual Chip Select*)
- GROUND (0V)

Estos pines son necesarios para realizar la comunicación y la lectura de los bits estables desde la SRAM, para ello utilizaremos la interfaz periférica serial, SPI (Sección 3.1.3).

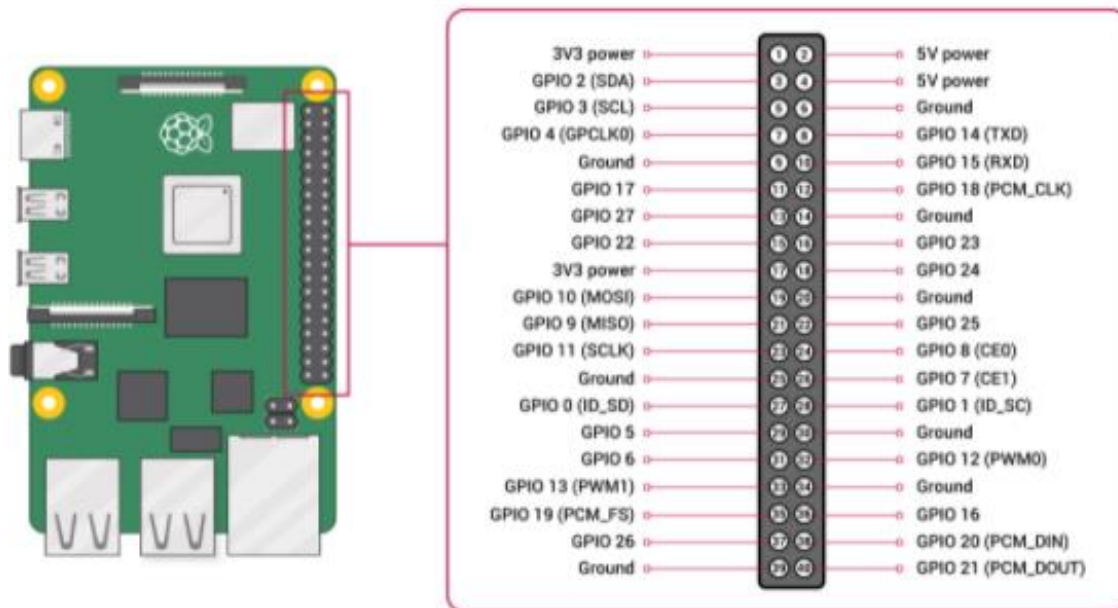


Fig. 3.1. GPIO de la Raspberry

En este proyecto solo serán conectados los pines nombrados anteriormente, el resto de los pines permanecerán en un estado ideal, también a la hora de interactuar con la interfaz, un teclado y ratón fueron enchufados en los puertos USB, al igual que un cable HDMI con el que poder proyectar la imagen en un monitor.

3.1.2. Circuito integrado 23LC1024

La elección inicial como dispositivo hardware fue la utilización de la propia DRAM que está instalada en la Raspberry Pi, con la utilización de una librería ya existente en Github [14], pero surgieron diversas complicaciones.

Para empezar, mientras que hay mucha información acerca de la comunicación entre una Raspberry y una SRAM, pero no hay mucha entre la comunicación entre una Raspberry y una DRAM. A pesar de eso fue seleccionada como opción, ya que era complicado adquirir una memoria SRAM. Este método de generar la salida de la PUF se trataba de la compilación del módulo del *kernel* con el fin de extraer los bits de la DRAM durante el inicio, pero cuando se inició el proceso de integración de este sistema, empezaron a surgir diferentes errores, de compilación de los módulo del *kernel* y de falta de archivos, por lo

que, una vez contactado el autor del repositorio, fue aclarado que dicho proyecto nunca fue acabado, es por ello que se cambió el enfoque de esta parte del proyecto para derivar la clave de una SRAM, un circuito integrado independiente.

Como dispositivo hardware sobre el que leer los bits ha sido escogido el circuito 23LC1024. Esto se trata de una memoria SRAM de ocho puertos, los cuales son:

Name	Function
\overline{CS}	Chip Select Input Pin
SO/SIO1	Serial Output/SDI/SQI Pin
SIO2	SQI Pin
Vss	Ground Pin
SI/SIO0	Serial Input/SDI/SQI Pin
SCK	Serial Clock Pin
HOLD/SIO3	Hold/SQI Pin
Vcc	Power Supply Pin

Fig. 3.2. Diagrama de la SRAM

Estos puertos son conectados con los pines de la Raspberry Pi [15], las conexiones son:

- $V_{cc} \rightarrow 3v3$ power (3.3V)
- SI/SIO0 \rightarrow GPIO 10 MOSI (*Master out – Slave in*)
- SO/SIO1 \rightarrow GPIO 9 MISO (*Master in – Slave out*)
- SCK \rightarrow GPIO 11 SCLK (*SPI Clock*)
- $\overline{CS} \rightarrow$ GPIO 17 Manual CS (*Manual Chip Select*)
- $V_{ss} \rightarrow GROUND$ (0V)
- $\overline{HOLD} \rightarrow V_{cc}$ (3.3V)

Dichas conexiones serán explicadas más a detalle en la sección 3.1.4, mientras que su funcionalidad será descrita en la sección 3.1.3. Para el montaje del circuito utilizaremos el diagrama de la Figura 3.3, haciendo solo uso de un condensador, al no ser requerido, para este caso, resistencias de *pull-up*. En este circuito utilizaremos unas condiciones específicas:

- El que *chip select* tendrá de input un lógico ‘0’. Esto es así con el fin de que, a la hora de proceder a la lectura de bits, no interfiera ni produzca errores.

- La señal de *hold* deberá poseer un lógico ‘1’ durante toda la lectura, en caso de no ser así, el circuito integrado interpretará que no se deben leer el input que pueda llegar ni mandar ningún output.
- El *Clock* será un pulso que operará a una frecuencia de 100 KHz, pudiendo operar a una frecuencia máxima de 20 MHz, por ello siendo un periodo de 10 μ s y, por tanto, un mismo tiempo de subida y bajada del pulso de 5 μ s.

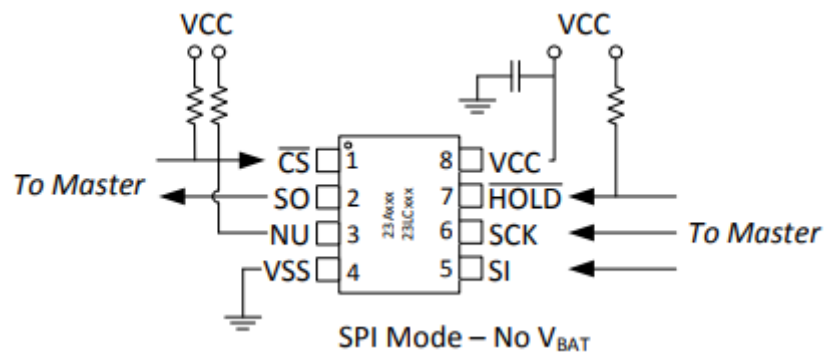


Fig. 3.3. Diagrama de la SRAM

3.1.3. Comunicación Raspberry Pi – SRAM a través de SPI

La lectura y escritura de los valores de la SRAM, será realizada a través del protocolo SPI, cuyas siglas corresponden con Interfaz periférica serial [16]. Mediante este protocolo se puede realizar un manejo de los datos dentro de la SRAM de una forma eficiente, al no requerir de muchos pines para llevar a cabo la comunicación, así como rápida, al poder operar a una frecuencia alta.

El protocolo SPI define una jerarquía, con la que podemos identificar qué dispositivo comunica las órdenes, definido como maestro o *master*, pudiendo sólo implementar uno por comunicación en el protocolo SPI, y el esclavo o *slave*, pudiendo ser uno o varios dependiendo de la configuración implementada. Dicho componente recibe dichas órdenes y las acata.

El protocolo consiste en mandar señales de reloj desde el dispositivo al circuito integrado. Durante estos periodos del reloj, se mandarán o recibirán de manera síncrona, la información binaria de entrada o salida de la memoria. Esta lectura y escritura se podrá configurar, pudiendo realizarse durante el flanco de subida de la señal de *Clock*, así como

de bajada. La comunicación tendrá lugar a través del *bus* (túnel) SPI, y puede realizar cuatro operaciones diferentes.

- *Clock*: como ya hemos mencionado anteriormente, se produce una señal de reloj con la que poder regular el resto de las tareas. Esta configuración permite diferentes opciones, pudiendo sincronizar el resto de las señales mediante un flanco de subida o de bajada, así como en combinación con un retraso de la señal.
- *MOSI (Master in – Slave Out)*: Este modo de operación consiste en la lectura de la información recibida desde el esclavo, es decir, consiste en un puerto de entrada, el cual recibe e interpreta la información de salida del esclavo.
- *MISO (Master out – Slave Out)*: Opuesto al anterior, este modo de operación consiste en la transmisión de información al esclavo, es decir, se trata de un puerto de salida del *master*, cuya salida es recibida por el puerto de entrada del esclavo.
- *Chip Select*: esta operación permite al maestro seleccionar a que esclavo desea transmitir la información, en caso de que el sistema conste de diversos esclavos, para ello activa el *flag* del *CS* del correspondiente esclavo. Este *flag* consiste en un lógico '0'.

SPI no es el único protocolo de comunicación entre dispositivos y circuitos integrados, pero es uno de los mejores, esto es debido a su capacidad, *Full Duplex*, comunicación simultánea de ambos canales, el canal de salida y el de entrada, también posee una gran configurabilidad, como lectura bit a bit, por páginas o secuencias, posee una implementación sencilla y compacta, utilizando un número reducido de pines para realizar la comunicación, y una intercambio de información de mayor velocidad que otros protocolos. A pesar de eso también tiene sus aspectos negativos. Como podría ser la falta de visibilidad del *master*, es decir, podría estar escuchando y transmitiendo información sin darse cuenta de que no hay ningún esclavo conectado o escuchando, o, el poder solo poseer un *master*, para regular las comunicaciones, limitación no encontrada en otros protocolos.

3.1.4. Diseño del circuito de comunicación entre el dispositivo y la SRAM

El circuito propuesto es la combinación de las configuraciones propuestas en las secciones 3.1.1 y 3.1.2 para los respectivos dispositivos, la Raspberry Pi y el circuito integrado 23LC1024. Como observamos en la Figura 3.4, se trata de un circuito simple. En dicho circuito podemos distinguir los siguientes componentes:

- Cable ROJO → Vcc con la línea Vcc de la *protoboard*, de la que derivaran las conexiones a *hold* y al puerto Vcc de la SRAM, así como la pata Vcc del condensador, el cual posee un valor de 10 nF.
- Cable AZUL → *Ground* con la línea de tierra de la *protoboard*, de la que derivaran las conexiones al puerto Vss de la SRAM, NU, puerto no usado y por ello conectado a tierra, y a la pata de tierra del condensador.
- Cable VERDE → Conexión entre el pin MOSI de la Raspberry con el puerto SI de la SRAM.
- Cable NARANJA → Conexión entre el pin MISO de la Raspberry con el puerto SO de la SRAM.
- Cable BLANCO → Conexión entre el pin SCLK de la Raspberry con la entrada de la señal del reloj de la SRAM.
- Cable GRIS → Conexión del *chip select* de la SRAM con un puerto de *General Output* de la Raspberry, cuya función ha sido configurada manualmente para simular la selección de dicha SRAM para la comunicación.

El diseño propuesto en la ficha [17] del circuito integrado 23LC1024 (Figura 3.3.) define la posibilidad de incluir resistencias *pull up*, las cuales no han sido necesitadas en el proyecto, a diferencia del condensador de desacoplo de 0.1 μ F, con el fin de proporcionar una señal de Vcc estable sin ruido.

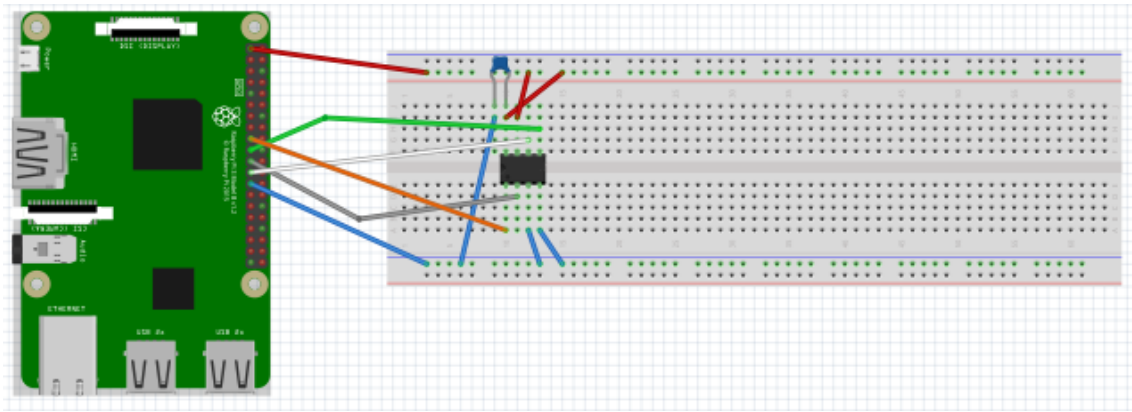


Fig. 3.4. Conexión Raspberry-SRAM con condensador de 0.1 μ F

3.2. Inscripción y derivación de la clave para autenticación

Para la segunda y tercera parte del proyecto necesitaremos el apoyo de un dispositivo donde lanzar los respectivos servidores, tanto el servidor de autenticación con el que terminar derivando la clave, como el servidor de comunicación con mensajes mediante el protocolo OSCORE.

Para ello cada cliente es lanzado en la Raspberry, donde, de manera correspondiente, se lanzará un cliente que tratará de autenticarse frente a un servidor, así como un cliente siguiendo el mismo protocolo OSCORE, con el que realizar la comunicación.

Con el fin de utilizar un entorno semejante al sistema operativo usado en Raspberry, fue seleccionado como sistema operativo, ejecutándose en una máquina virtual, Ubuntu versión 20.04.2.0, al poseer numerosas semejanzas con el sistema operativo perteneciente a la Raspberry, Raspbian.

La inscripción del dispositivo Raspberry y su posterior autenticación se realiza con un servidor, ejecutándose en la máquina virtual. El servidor implementa los protocolos de inscripción y autenticación descritos a continuación y utiliza como repositorio el sistema de archivos. El servidor se ejecuta y escucha en el puerto 5000 a cualquier cliente que quiera inscribirse o autenticarse.

En caso de que el dispositivo deba inscribirse, el dispositivo se someterá al protocolo de inscripción, en el cual deberá enviar aquellos bits estables que haya generado como salida de su PUF. Estos bits son registrados por el servidor en un almacén físico dentro de su memoria, donde los clasifica en función de un valor identificativo del cliente, en este caso, se seleccionó el número de serie. De esta manera el servidor posee la información necesaria para poder identificar al cliente en el futuro.

Una vez completada la inscripción, el cliente podrá autenticarse las veces que quiera. En este caso, el servidor no reclamará los bits estables, ya que sería un riesgo de seguridad, ya que entonces cualquier atacante podría escuchar la conversación y extraer dichos bits, con lo que podría pasarse por dicho cliente. En vez de ello, el servidor realizará retos o *challenges* que constituirán en diferentes pruebas a someter al cliente con el fin de asegurar que es el. Estas varían en función del listado de los bits de la memoria del cliente, pueden ser en función de una dirección de los bits estables, un conjunto o toda la memoria. En este caso se seleccionó 4096 bytes, ya que es el máximo que permite la memoria sin cambios en el *kernel*. Estos bits entonces son sometidos a un protocolo de autenticación, un proceso acordado previamente, en nuestra aplicación, dichos bits son la entrada de una operación XOR entre dichos bits con una secuencia aleatoria, que, tras ejecutar dicho procedimiento, se realiza un hash del resultado. Este proceso es ejecutado tanto por el servidor como el cliente, obteniendo dos resultados a dicho reto por parte de ambos. Dicho resultado del cliente es entonces enviado al servidor, que lo comparará con su propio resultado, y con ello dictaminando si es el cliente quien dice ser, o en caso de que no, comunicar que la autenticación ha sido fallida.

3.3. Comunicación Cliente/Servidor mediante OSCORE

Esta última fase del sistema mantiene el mismo concepto de la fase anterior, teniendo el cliente CoAP con soporte de OSCORE desplegado en una Raspberry Pi, y el

servidor escuchando desde la máquina virtual, ubicada en un ordenador desde el puerto 5683.

Como resultado del proceso de autenticación previo, se deriva el contexto de seguridad de OSCORE que permite establecer una comunicación segura entre ambos dispositivos. En el contexto de seguridad se recoge la información del secreto maestro, así como valor aleatorio para proceder a su construcción. En nuestro caso, el procedimiento para recoger la información del secreto maestro ha sido modificado con el fin de añadirle una capa de seguridad. En vez de estar escrito en el código o leído desde un fichero con un contenido fijo, dicho secreto maestro será el hash de los bits calculado en la fase anterior, por ello será diferente en cada sesión, debido que, aunque los bits estables sean constantes a través de las diferentes sesiones, el valor aleatorio que se aporta a la hora de realizar el XOR y después el hash, permite que el valor de salida, y por ello, el del contexto maestro sea diferente en cada sesión. A partir de este contexto se deriva la clave con la que tanto el cliente como el servidor entablarán la comunicación.

Con esta fase se pretende aportar el elemento de comunicación entre el dispositivo IoT, la Raspberry Pi, y el servidor, en este caso el ordenador, codificada mediante el protocolo OSCORE, con el fin de cifrar las comunicaciones y con ello evitar que un atacante pueda extraer la información

Capítulo 4

Implementación del sistema

La implementación del sistema se ha realizado utilizando dos lenguajes de programación, teniendo en cuenta las diferentes opciones propuestas. Para la generación de la PUF se utilizó Python, ya que posee una librería fácil de implementar y sencilla de entender. En cambio, tanto para el proceso de Registro/Autenticación como para la comunicación OSCORE se utilizó Java, acorde con la implementación elegida para desarrollar este proyecto, Californium.

A continuación, se hará una explicación detallada de la implementación del proyecto.

4.1. Spidev: Librería de comunicación SPI para generar la PUF

Spidev se trata de una librería licenciada por el MIT [18]. Esta librería está programada en Python y trata de un módulo para realizar el intercambio de información con dispositivos que admitan el protocolo SPI. Es por eso por lo que se ha derivado el

código de la generación de la salida de la PUF usando este protocolo de comunicación, el cual será explicado a continuación por partes:

4.1.1. Importación de librerías

La implementación de nuestra PUF requiere diversas librerías las cuales son:

```
1. import spidev
2. import RPi.GPIO as GPIO
3. import os
```

Estas librerías nos permiten desarrollar el código con el que configurar nuestra PUF.

- Spidev: como hemos nombrado anteriormente, se trata de la librería con la que realizar la comunicación SPI. Al no ser una librería nativa de Python es necesario instalarla antes, lo cual se puede hacer desde la página [18] o mediante la introducción del comando `pip install spidev`. Tras ello, tendremos acceso a las múltiples funciones que ofrece.
- GPIO: nos permite modificar la funcionalidad y el uso de los puertos GPIO de la Raspberry Pi, con diferentes configuraciones y valores de dichos puertos.
- OS: esta librería ha sido importada para implementar la función de leer y escribir archivos en nuestro proyecto.

4.1.2. Inicialización de variables

Tras importar las librerías se procede a la inicialización de las variables a implementar en la lectura de la SRAM:

```
4. spi = spidev.SpiDev()
5. spi.open(0,0)
6. spi.max_speed_hz = 100000
7. spi.mode = 0b00
8. spi.no_cs = True

9. read = 0x03
10. GPIO.setmode(GPIO.BCM)
```



```
11. GPIO.setup(17,GPIO.OUT)
12. GPIO.output(17,GPIO.HIGH)

13. GPIO.output(17,GPIO.LOW)

14. spi.writebytes([read,0x00,0x00,0x00])
15. data = spi.readbytes(5000)
16. stableBits = []
```

Esta sección del código inicia con la inicialización de un objeto Spidev (línea 4) para después abrir mediante la línea 5 el bus 0 y la comunicación con el dispositivo 0. Esto es así al solo utilizar un esclavo, en caso de que fueran más, hubiera sido necesario abrir un segundo canal.

Tras ello se establece la frecuencia del reloj, que determinará la frecuencia a la que se ejecutarán todas las operaciones en el sistema. Ha sido seleccionado 100000 Hz, ya que como se determinó en algunas pruebas, a frecuencias más altas el circuito tendía a indicar solo un 0 en todas las posiciones, y a frecuencias más bajas obtenemos los mismos valores de los bits y empieza a ralentizarse la ejecución del programa en gran medida.

Como modo de operación, se seleccionó el 0b00, el correspondiente a una lectura byte a byte, lo cual es lo que requerimos al necesitar evaluar determinadas posiciones de la memoria de longitud de un byte cada una.

Por último, hemos seleccionado la opción de *no chip select*, ya que el *chip select* se implementó de manera manual, mediante la utilización de la librería GPIO. Para ello establecemos el PIN 17 como *output*, y simulamos la misma señal que la Figura 4.1, iniciándose en *high* (lógico ‘1’), para más tarde enviar un ‘0’ a través de ese pin (línea 11), y así “seleccionar” el dispositivo con el que interactuar (solo hay uno por lo que este paso se podría mantener a ‘0’ desde el principio), y una vez terminada la comunicación lo volveremos a establecer como ‘1’.

Una vez configurado el canal SPI de comunicación se procede a una configuración de la operación a realizar. Ésta será la de lectura de bits. Para ello se utiliza la línea `spi.writebytes`, la cual se compone de una secuencia de 32 bits. Esta función es

necesaria para realizar cualquier operación en la memoria, ya que se necesitará para inicialmente establecer un valor a los registros y con ello indicar una operación o funcionalidad en concreto.

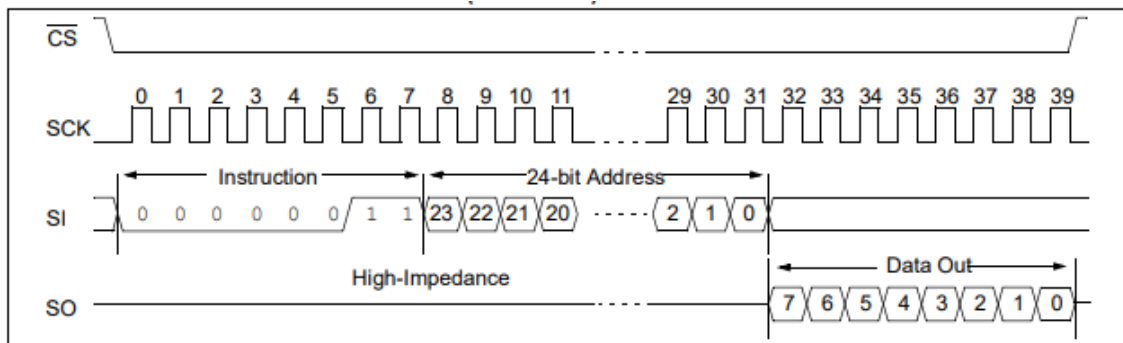


Fig. 4.1. Secuencia de lectura byte a byte

Como observamos en la Figura 4.1, esta secuencia se compone de dos partes. Primero, podemos ver la instrucción en forma de 8 bits, habiendo diversos tipos como lectura de la SRAM o escritura en la SRAM, correspondiendo al modo de operación utilizado, entre los que encontramos el que es implementado en este sistema 0x03 (línea 9), que corresponde a indicar que estamos reclamando una operación de lectura. Tras ello se establecerán los 24 bits de la dirección en la SRAM. Esta función `spi.writebytes` recibe de argumento primeramente la instrucción *read* (0x03), para después acompañarlos con la dirección inicial desde donde se quiere leer, en este caso 0x000000 (en hexadecimal). Tras ello se extraen los bits de salida, en la Figura 4.1 se establece como *Data out*, todo ello de forma síncrona con el *Clock*. En este caso, por limitaciones del protocolo SPI en el propio *kernel* de la memoria, solo se podía realizar una lectura de hasta 4096 bytes, lo que representa hasta la dirección de memoria 0x001000 de la SRAM, pero si se anula esa limitación, se podría llegar a leer toda la memoria, que correspondería con las direcciones desde 0x000000 hasta 0x01FFFF. Esto es así, ya que esto abarcaría todas las direcciones de memoria desde el byte 0 al 128K, todas las palabras de la memoria.

4.1.3. Extracción inicial de los bits

Con esta funcionalidad, primero comprobamos si está el archivo, en caso de que resulte que no procederemos a crearlo, mediante el método *open* creamos el archivo con permisos de escritura. Una vez creado extraemos los valores de la lista *data*, la cual contiene los datos leídos mediante la función *readbytes* previamente mencionada. A la vez que los valores son extraídos también son formateados con el fin de obtener una secuencia binaria de 8 bits por línea. Esto es necesario debido a que Python detecta los ceros a la izquierda y los elimina, por lo que cuando comparas bytes en la misma *address*, no estas comparando bit a bit ambas secuencias. El código que implementa esta secuencia se muestra a continuación:

```
17. if(os.path.isfile("/home/pi/Documents/ReadV2/ReadSRAM/pufUNF.txt")
    == False):
18.     f = open("pufUNF.txt", "w")
19.     for byteRead in data:
20.         f.write('{0:08b}'.format(byteRead) + "\n")
21.
22.     f.close()
```

4.1.4. Detección de bits estables

Una vez certificado que ya había sido creada una PUF, el programa accederá a esta segunda parte.

```
23. else:
24.     f = open("pufUNF.txt", "r")
25.     for i in range(len(data)):
25.         readBytes = '{0:08b}'.format(data[i])
27.         stableBytes = f.readline()
28.         stableSeq = ""
29.         for j in range(8):
30.             # readBit = (readBytes & (1<< j)) >> j
31.             # stableBit = (stableBytes & (1<< j)) >> j
32.             readBit = readBytes[j]
33.             stableBit = stableBytes[j]
34.             if (readBit == stableBit):
35.                 stableSeq = stableSeq + stableBit
```

```

34.             else:
35.                 stableSeq = stableSeq + "X"
36.             stableBits.append(stableSeq)
37.         f.close()
38.         f = open("pufUNF.txt", "w")
39.         for seq in stableBits:
40.             f.write(seq + "\n")

```

En esta sección se ejecutará después de reiniciar la Raspberry, en ella identificará que ha sido creado previamente el archivo y lo formateará como con anterioridad a una secuencia de 8 bits. A la par, se abrirá el archivo con la función *open*, pero en este caso, el archivo se abrirá con solo permisos de lectura. Tras esto, ambas secuencias serán analizadas bit a bit, byte a byte, entre el archivo original y los nuevos bits leídos, guardando dicha información en una lista de bytes.

Para ello, cuando dicha comparación resulte positiva, se guardará el valor del bit evaluado, dentro en su celda correspondiente a su *address*. En caso de que la comparación salga fallida, se almacenará dicho bit como una “X”, esto nos permitirá distinguir los bits estables de los inestables.

Una vez realizada la comparación completa, reabrimos el archivo original con permisos de escritura, lo cual sobrescribirá la información con la nueva información de los bits actualizada, la cual se irá refinando cuantas más veces se realice el procedimiento.

4.1.5. Creación de la salida de la PUF

Una vez finalizado el proceso, para acabar, se formateará la lista de bytes en una secuencia.

```

f1 = open("pufUNF.txt", "r")
f2 = open("puf.txt", "w")

completeSeq = ""

for byte_to_read in f1:
    for bit in range(0,8):
        # bit_to_read = (byte_to_read & (1<< bit)) >> bit

```

```
        bit_to_read = byte_to_read[bit]
        if (bit_to_read != "X"):
            completeSeq = completeSeq + bit_to_read
f2.write(completeSeq)

f1.close()
f2.close()
```

Para llevar a cabo eso se creará un nuevo archivo o abrirá uno existente, con permisos de escritura, mientras que se abrirá el archivo inicial con permisos de lectura. Tras ello se iterará a lo largo de todo el documento añadiendo los bits estables a la secuencia, para ello descartaremos todos aquellos bits que sean “X”, finalmente escribiendo dicha secuencia en el archivo que será interpretado por el cliente de autenticación.

4.2. Inscripción y autenticación de la Raspberry

La inscripción de un nuevo dispositivo o la autenticación de uno ya inscrito se realiza de manera íntegra en Java, esto es así debido a la facilidad de implementar la funcionalidad que precisamos de manera sencilla. Para ello, se ha implementado un sistema de *sockets*, cliente-servidor, y con el objetivo de crear una comunicación segura, necesaria para la segunda parte, la autenticación, se han utilizado sockets seguros , TLS (*Transport Layer Security*) [19], , asegurando que la información y los recursos que se encuentren en dicho flujo de la comunicación no sean modificados, así como evitar que otros dispositivos se interpongan en la comunicación, es decir, que cumplan los principios de autenticación, confidencialidad e integridad.

En la Figura 4.2 y Figura 4.3 se pueden ver las funciones que describiremos a continuación, siendo las clases que empiezan por *Library*, las librerías del cliente y servidor, y las otras dos clases son las clases ejecutables, que se basan en su librería correspondiente.

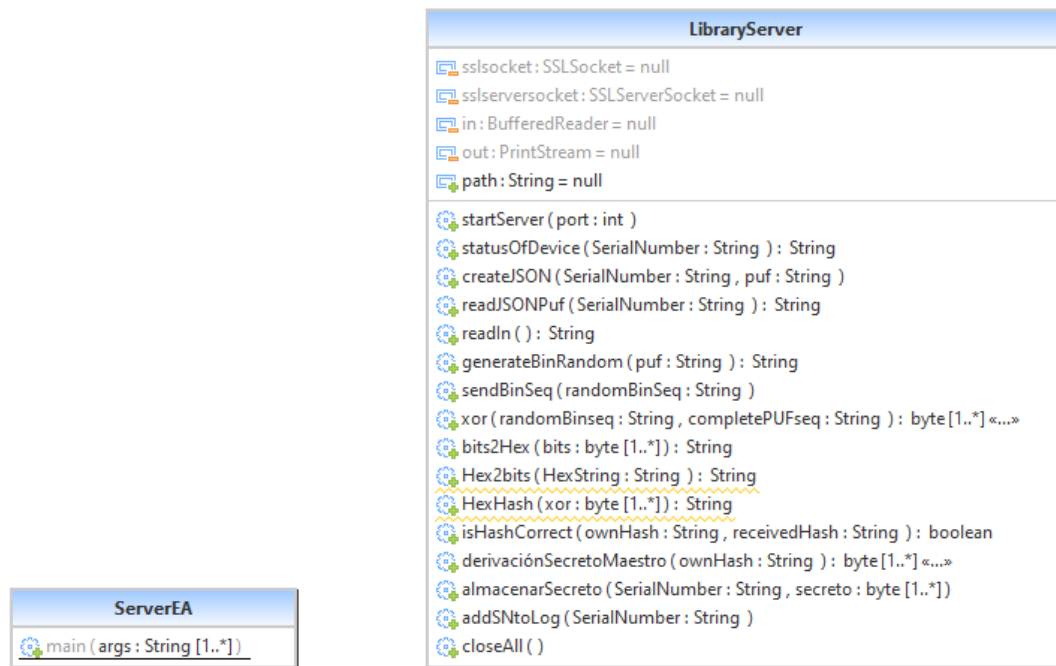


Fig. 4.2. Diagrama de funciones del servidor (E y A)

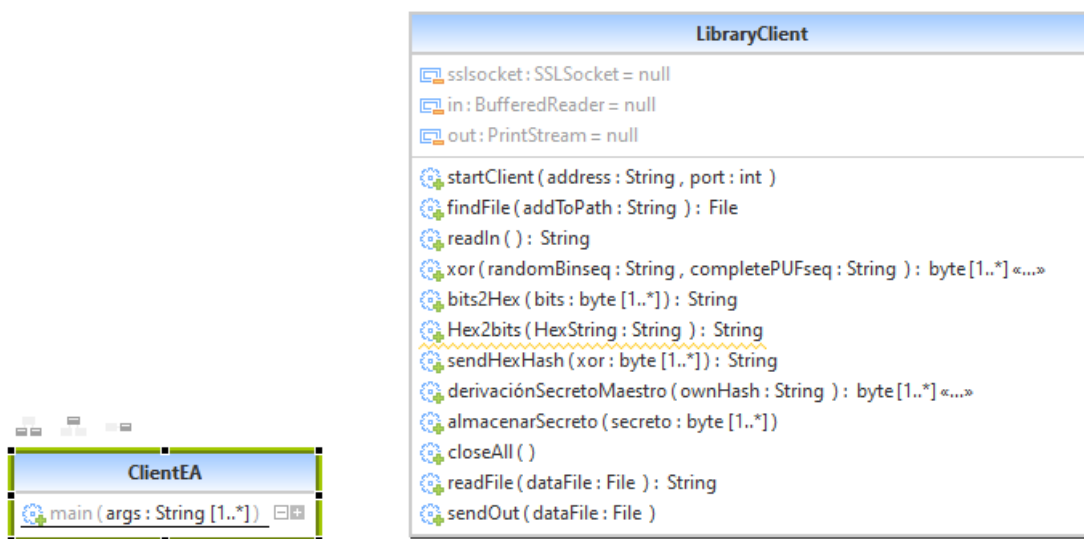


Fig. 4.3. Diagrama de funciones del cliente (E y A)

4.2.1. Implementación de *sockets* TLS

Tanto la inscripción como la autenticación se realizan mediante unos sockets TLS, que son la versión más reciente de los certificados SSL (*Secure Sockets Layer*), con los que se puede cifrar la información compartida entre el cliente y el servidor. Esta

comunicación se realiza mediante un cifrado simétrico de la información y la utilización de certificados, que son los que permiten identificar qué dispositivo.

En esta implementación encontramos dos archivos que incluyen los certificados [20], el *server.pem* (PEM: *Privacy Enhanced Mail* [21]) que se trata de un certificado que incluye las claves, así como otros certificados, como los certificados raíz, que identifican la autoridad que emite los certificados, y el *server.pkcs12* (PKCS12: *Public Key Cryptography Standard # 12* [22]), que es igual que el *server.pem*, con la variación de estar encriptado, y solo poder descifrarse con la clave, convirtiéndose entonces en un archivo *.pem*.

También encontramos el archivo *client.trust* [23], donde se almacena el ancla de confianza, que certifica si el dispositivo al que se conecta es de confianza, así como la conexión entre los dispositivos. Esta ancla de confianza es la encargada de verificar las firmas digitales y las rutas de certificación.

Para crear el socket entonces primeramente establecemos las diferentes propiedades, en caso de ser el cliente indicamos que su almacén de confianza, donde encontramos el ancla de confianza, se encuentra en el archivo *client.trust*, de la misma manera, por parte del servidor, indicamos que su almacén de claves, con las diferentes claves a usar en la comunicación. Estas propiedades entonces las establecemos por defecto para un futuro cliente y servidor, y sobre ellas, creamos el cliente, a partir de la dirección a la que conectarse y el puerto, y el servidor, a partir del puerto para abrir la conexión, a la vez que creamos los canales de comunicación de salida y, de entrada.

4.2.2. Selección de la funcionalidad

El sistema se compone de dos diferentes funcionalidades, la inscripción del dispositivo, el modo de operar cuando el dispositivo no ha sido reconocido, y la autenticación, cuando el dispositivo ha sido reconocido, por lo que se quiere comprobar que se trata de cliente.

Esta comprobación se ha implementado mediante el reconocimiento o no del número de serie de la Raspberry Pi y su extracción se ha realizado mediante Python, al ser bastante más simple que en Java.

```
# Extract serial from cpuinfo file
cpuserial = "0000000000000000"
try:
    f = open('/proc/cpuinfo', 'r')
    for line in f:
        if line[0:6]=='Serial':
            cpuserial = line[10:26]
    f.close()
except:
    cpuserial = "ERROR0000000000"

f = open("Serial Number.txt", "w")

f.write(cpuserial)

f.close()
```

Como observamos en el código, el número de serie es de 16 dígitos alfanuméricos, los cuales se observan en el archivo `cpuinfo` en el directorio `/proc` de la Raspberry Pi. En dicho archivo encontramos diferentes datos de la CPU, siendo el que nos importa aquel con el nombre del campo “Serial:”. Dicho resultado es después escrito mediante el método `open` con la opción de `write`, a diferencia que con el otro archivo que era `read`.

4.2.3. Inscripción del dispositivo

El proceso de inscripción se encuentra implementado en Java, y es el más sencillo, al ser una simple acción de envío de información y catalogación de dicha información por parte del servidor. A continuación, veremos el código del cliente:

En él, el cliente primeramente lee el estado en el que está, y al ser una primera ocasión, el servidor le indicará que está en la fase de inscripción. Por ello, el cliente, lee el archivo `puf.txt` en el que está generada la salida de la PUF y la envía al servidor. Tras ello su función en la inscripción se ha terminado por lo que simplemente indica el final del proceso. Por otro lado, en la parte del servidor, como observamos en el código:

Primeramente, usando el método `statusOfDevice(SerialNumber)`, determina si encuentra un registro del dispositivo, en caso de que no, cómo se trataría de la inscripción, crea un directorio cuyo nombre será el número de serie del dispositivo cliente, con ello podrá de

forma rápida acceder a la información del determinado dispositivo más adelante. Este asocia el ID del dispositivo, en nuestro caso como hemos mencionado el número de serie, con dicha secuencia de bits, y tras ello crea con ambos campos un archivo JSON, y lo guarda en el directorio correspondiente al número de serie.

```
JSONObject jsonObj = new JSONObject();
jsonObj.put("puf", puf);
jsonObj.put("SerialNumber", SerialNumber);
FileWriter pufFile = new FileWriter(completePath);
pufFile.write(jsonObj.toJSONString());
```

Como observamos en el código de arriba, para crear un archivo JSON, importamos la librería de JSON, y por ello creamos un objeto JSON, y usando el método *put* que posee de argumentos el nombre del campo y su correspondiente valor, introducimos en el objeto, el número de serie y la secuencia de salida de la PUF. Por último, tras crear un objeto de escritura e indicar el *path* en el que se escribirá dicho documento JSON, se utilizará el método *write* para crear dicho documento a partir de los campos incluidos en el objeto JSON.

4.2.4. Autenticación del dispositivo

En este caso la autenticación es más compleja que la inscripción, esta conlleva una serie de pasos más que la inscripción. Dicho código, al igual que la inscripción, se ha creado usando Java, al igual que diversas librerías en las que apoyarse. A continuación, se analizará la sección de autenticación del cliente:

```
if(status.equals("authentication")) { //Authentication
    System.out.println("Dispositivo identificado, autenticación en curso");
    String randomBinSeq = Client.readIn();
    String puf = Client.readFile(Client.findFile("puf.txt"));

    Client.sendHexHash(Client.xor(randomBinSeq, puf));
    System.out.println("Autenticación completada");
}
```

En él, el cliente lee la secuencia de entrada proporcionada por el servidor. Esta secuencia posee una longitud igual a la secuencia de bits estables, variando en función de los diferentes bits estables de las diferentes memorias. Tras ello, encuentra primeramente el archivo de la PUF correspondiente a partir del número de serie, que como determinamos previamente, coincide con el nombre del directorio donde encontrar dicho archivo. Una vez extraídas ambas secuencias se procede a hacer un XOR de ellas, con las que se extrae una única secuencia, y, para acabar el proceso, se hace un *Hash* del XOR. Para ello primeramente se invoca el *MessageDigest*, que se trata de una función criptográfica de una longitud de salida determinada por el algoritmo a usar, en nuestro caso “SHA-256”, la cual se compone de 256 bits, es decir, 32 bytes. Para una más rápida comparación, dicho hash es entonces convertido de binario a hexadecimal.

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hashXoR = digest.digest(xor);
hexHashStr = bits2Hex(hashXoR);
```

Dicho *hash* se manda al servidor, el cual analizaremos a continuación:

```
if(status.equals("authentication")) { //Authentication
    System.out.println("Dispositivo identificado, autenticación en
    curso");
    String puf = Server.readJSONPuf(SerialNumber);
    String randomBinSeq = Server.generateBinRandom(puf);
    Server.sendBinSeq(randomBinSeq);

    String ServerHash = Server.HexHash(Server.xor(randomBinSeq, puf));
    String ClientHash = Server.readIn();
    boolean isCorrect = Server.isHashCorrect(ServerHash, ClientHash);
    if(isCorrect == true) {
        byte[] secretoMaestro
        Server.derivaciónSecretoMaestro(ServerHash);

        Server.almacenarSecreto(SerialNumber, secretoMaestro);
        Server.addSNtoLog(SerialNumber);
        System.out.println("Secreto derivado y almacenado, cierre
de conexión");
        Server.closeAll();
    }
}
```

```
}  
}
```

El servidor primeramente extrae el valor de la PUF del archivo JSON, para ello hace uso del número de serie con el que encontrar dicha PUF, que nos permite definir el *path* en el que está definido el archivo y tras ello extraer con el método *get*, el valor del campo PUF.

Una vez extraída dicha secuencia se analiza su longitud, y mediante la función *generateBinRandom*, obtenemos una secuencia aleatoria con la misma longitud que la PUF, la cual es enviada al mismo tiempo al cliente. Tras ello, procedemos como antes a hacer un XOR de la secuencia aleatoria y un hash del XOR obtenido, de la misma manera que el método del cliente.

Tras ello, el servidor comprobará ambos *hashes*, y en caso de que la comparación sea fructífera, lo indicará al cliente, pudiendo entonces derivar el secreto maestro del contexto común, creando un archivo JSON con dicha secuencia, y realizar la comunicación OSCORE.

Por último, para llevar un *log* de los dispositivos que han sido autenticados, se añadirá una entrada al archivo *log* de las autenticaciones, el cual estará almacenado en el servidor. Dicha entrada se comprenderá de la hora y fecha a la que se autenticó el dispositivo, así como su número de serie, algo que será necesario para que pueda iniciar una comunicación OSCORE con dicho dispositivo.

4.3. Comunicación OSCORE entre dispositivos

Para la implementación de la última sección del proyecto, como ya ha sido mencionado anteriormente, se trabaja con la librería Californium. Para ello se utilizó las librerías **californium-core** y **cf-oscore**. Inicialmente, se procedió a testear el servidor y cliente CoAP implementados: *HelloWorldClient.java* y *HelloWorldServer.java*, respectivamente. Éstos, servidor y cliente, se extendieron para crear un recurso de comunicación de forma continuada denominado *ComsResource*, que establecerá los *endpoint* entre los que se conectarán servidor y cliente. En la Figura 4.3 observamos el diagrama de funciones que abarcan los códigos del servidor y cliente CoAP con soporte

de OSCORE, y en la Figura 4.4, observamos cómo se conectan estas clases con el resto del paquete. Se ha reducido esta última Figura a las conexiones inmediatas ya que no se ve con claridad el diagrama completo.

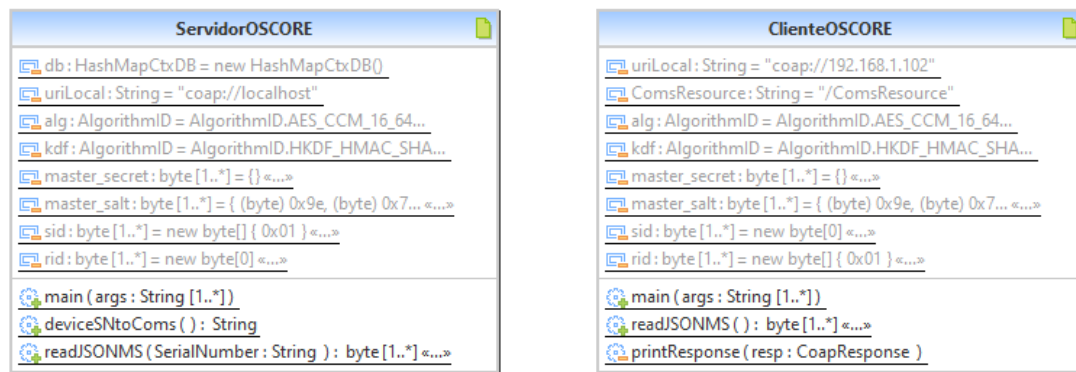


Fig. 4.4. Diagrama de funciones del servidor y cliente

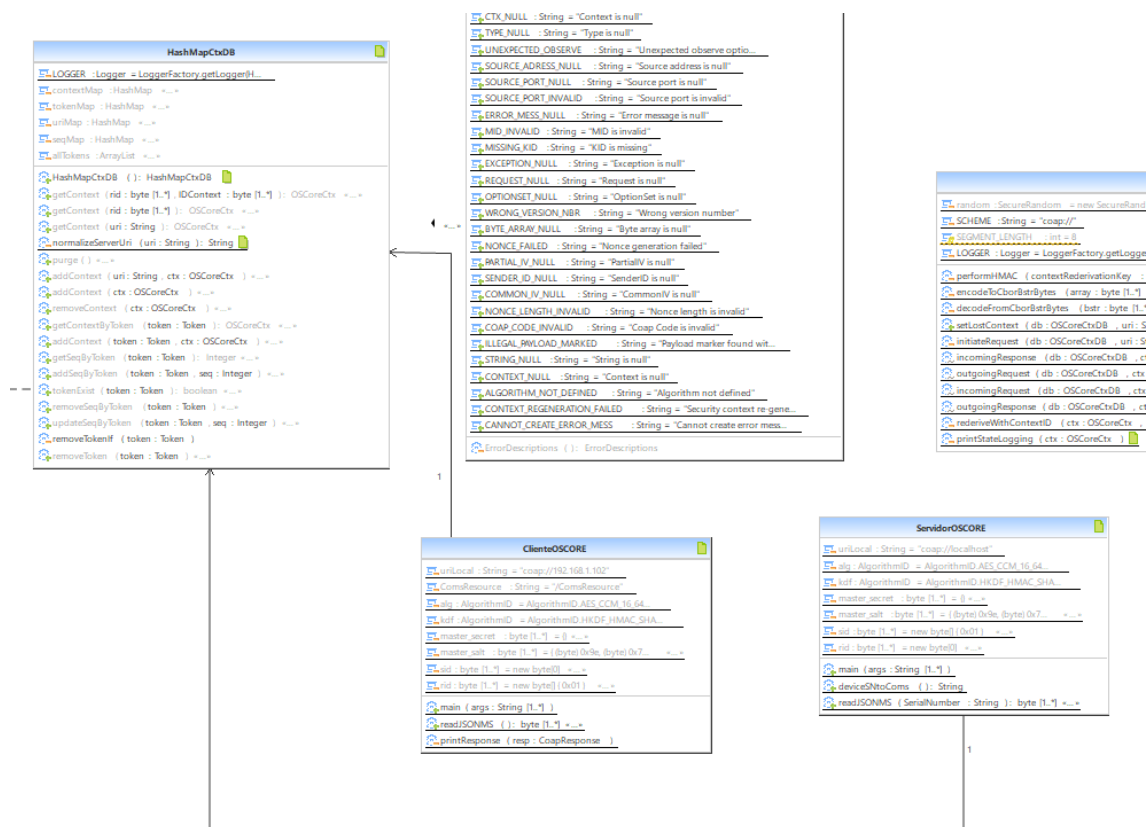


Fig. 4.5. Diagrama de clases del servidor y cliente

Sobre los ejemplos se ha modificado el soporte de la seguridad de OSCORE, a través del contexto de seguridad, cuyo secreto maestro se obtiene de la PUF. En vez de encontrarse

hardcodeado en el código como un array de bytes, este se obtiene mediante la derivación del hash obtenido en la autenticación, por ello tanto el cliente como el servidor se derivan de manera independiente, evitando un posible *eavesdropping*, al ser contenido potencialmente delicado.

Dicho contexto se deriva de manera diferente en el cliente y en el servidor, por parte del servidor:

```
String SerialNumber = deviceSNtoComs();
master_secret = readJSONMS(SerialNumber);

OSCoreCtx ctx = new OSMCoreCtx(master_secret, false, alg, sid, rid, kdf, 32,
master_salt, null);
```

Inicialmente se extrae el número de serie del archivo .log creado anteriormente, leyendo la última entrada de este mediante la función *deviceSNtoComs()*, y una vez extraído dicho número se hace una búsqueda del archivo “SecretoMaestro.json” correspondiente a dicho dispositivo, donde se encontrará dicho secreto, y, tras ello, servirá como input para crear el contexto maestro.

Por parte del cliente, es mucho más sencillo, al no necesitar ningún valor para encontrar dicho archivo por lo que simplemente hará una búsqueda de dicho archivo como se mostrará a continuación para extraer el secreto maestro e introducirlo en el contexto.

```
master_secret = readJSONMS();

OSCoreCtx ctx = new OSMCoreCtx(master_secret, true, alg, sid, rid, kdf, 32,
master_salt, null);
```

Una vez establecido el contexto, es el turno de crear el canal de comunicación, para ello estableceremos un recurso para llevar a cabo la comunicación, como se observa abajo, el cual recoge el valor del escáner configurado para leer el *input* del teclado, y establecer ese valor como la cara útil del mensaje, para después de eso realizar un *Response* a la solicitud GET realizado por el cliente.

```
OSCoreResource ComsResource = new OSMCoreResource("ComsResource", true) {
    Scanner sc = new Scanner(System.in);
```

```

@Override
public void handleGET(CoapExchange exchange) {
    System.out.println("Accessing ComsResource
resource");

    String data = sc.nextLine();
    Response r = new Response(ResponseCode.CONTENT);

    r.setPayload(data);
    exchange.respond(r);
    if(data.equals("Over")) {
        server.destroy();
    }

}

};

```

En el lado del cliente, como hemos mencionado anteriormente, se realizará un *Request*. Tras ello tras recibir el determinado *Response* al *Request*, imprimirá el resultado por pantalla, componiéndose de dicho resultado, el Response Code y el Response Text. Dicha configuración la observamos a continuación.

```

Request r = new Request(Code.GET);
CoapResponse resp = c.advanced(r);
printResponse(resp);

String data = "";

r = new Request(Code.GET);
r.getOptions().setOscore(new byte[0]);
while (!resp.getResponseText().equals("Over")) {
    if(!resp.getResponseText().equals("Over")) {
        resp = c.advanced(r);
        printResponse(resp);
    }
}
c.shutdown();

```

Tras ello, la configuración ha terminado, procediendo a una integración de todas las partes mencionadas en un sistema entero.

4.4. Integración de todos los subsistemas

Para la integración de los subsistemas inicialmente se consideró la idea de integrar todas las funcionalidades en un solo punto. Pero, aunque podría reducir algunos procedimientos intermedios, añadía una complejidad innecesaria, ya que el resultado de dicha integración era la de un sistema caótico. Por ello, se decidió crear tres partes interconectadas pero independientes, lo cual aporta una gran claridad al proyecto y permite analizar cada evento de forma aislada.

Para ello, inicialmente se **genera la PUF**, cuyo resultado será almacenado en un fichero de texto, y, como hemos dicho anteriormente, en el estado de **inscripción** del sistema, el contenido de archivo será enviado al servidor, que creará un nuevo directorio en su librería de dispositivos, y almacenará el archivo JSON de dicho dispositivo en el directorio. En la siguiente sesión, que corresponde a la de **autenticación**, dicho archivo será localizado y será utilizado para verificar la identidad del dispositivo cliente, a lo que, en caso de una comparación positiva, se registrará el secreto maestro que ha sido derivado de la salida de la autenticación como el hash de la PUF con una información de ayuda, la cual se trata de una secuencia binaria aleatoria al alcance del cliente. Y, apoyándose el servidor en el archivo .log para localizar el dispositivo que acaba de autenticarse, localizará dicho secreto el cual será utilizado para configurar el contexto maestro de **comunicación del servidor y cliente OSCORE**, completando la conexión, y con ello, se completa la **IMPLEMENTACIÓN DE UN MECANISMO DE GESTIÓN DE CLAVES PARA OSCORE**.

Capítulo 5

Pruebas del sistema

En este capítulo haremos una simulación del proyecto entero. Para ello primero realizaremos diferentes pruebas a los diversos componentes que componen el proyecto, para después finalizar haciendo una reproducción global de la funcionalidad entera.

Las diferentes pruebas que realizar son:

- **En la primera prueba, Generación de la PUF**, evaluaremos paso a paso el procedimiento de generar una PUF, desde la primera lectura de la memoria hasta la obtención de los bits estables.
- **En la segunda prueba, Registro del dispositivo**, procederemos a documentar el registro del dispositivo cliente, la Raspberry Pi, frente al servidor, el ordenador, mediante el número de serie de los dispositivos y los bits estables extraídos en el segundo apartado.
- **En la tercera prueba, Autenticación del dispositivo**, volveremos a retomar la comunicación entre el servidor y el cliente, y observaremos como ambos dispositivos, cliente y servidor, realizan las operaciones correspondientes para verificar la identidad del servidor.
- **En la cuarta y última prueba, Comunicación segura usando OSCORE y la clave maestra derivada**, finalizaremos las pruebas particulares, derivando

inicialmente la clave maestra haciendo uso de los datos extraídos de la PUF y los resultados posteriores, para introducirlos en el contexto de la clave, y con ello una vez configurada dicha clave, realizaremos una comunicación mediante el protocolo OSCORE

Una vez realizadas todas las pruebas particulares, realizaremos una prueba general de la funcionalidad del proyecto entero, simulando dichas partes de manera integrada, creando finalmente **un sistema de gestión de claves para la comunicación OSCORE**.

5.1. Primera prueba: Generación de la PUF

Esta prueba se realizará de forma integran en la Raspberry Pi, haciendo uso de la librería Spidev como nombramos antes. Para ello inicialmente procederemos a realizar una primera lectura de los bits de la SRAM mediante la ejecución del archivo **puf.py**. Este archivo se compila y ejecuta introduciendo el comando **Python puf.py**, que como respuesta nos crea un archivo denominado **pufUNF.txt**.

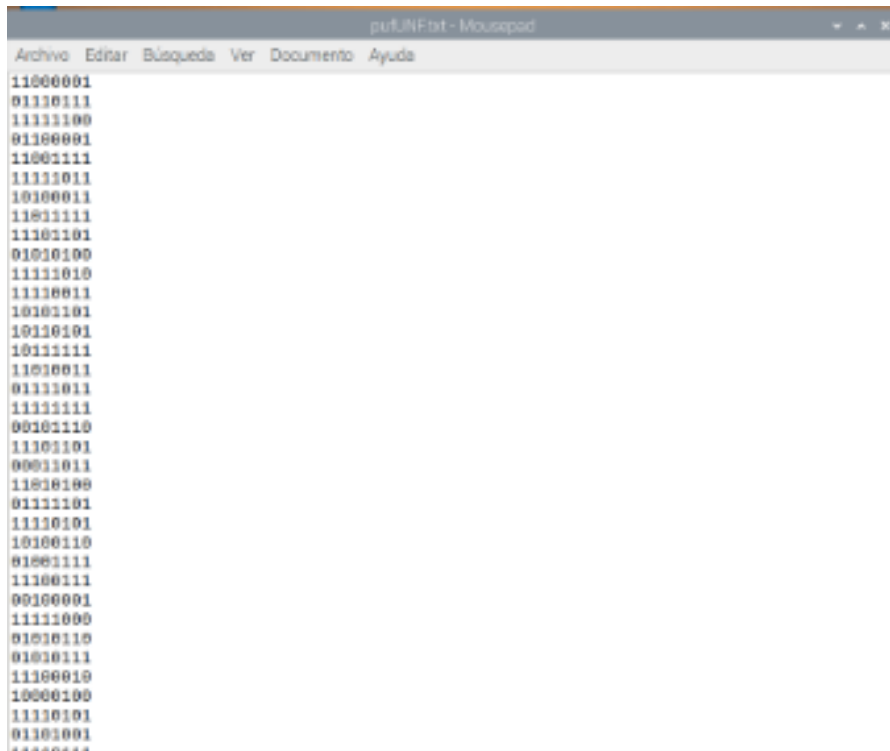


Fig. 5.1. Lectura inicial de los bits

Como observamos en la Figura 5.1., hemos recibido una lista de 4096 bytes, la capacidad máxima a leer de la SRAM sin la modificación del *kernel* del dispositivo. Estos bytes, al ser obtenidos de imperfecciones aleatorias de los materiales semiconductores de la SRAM, también son aleatorios, no encontrando ninguna relación entre ellos.

El archivo, denominado pufUNF.txt, recibe ese nombre debido a que, una vez creado la lista de bits, se trata de la entrada de un sistema implementado con el fin de convertir, esta lista de bits en una secuencia única, como podemos comprobar en la Figura 5.2.

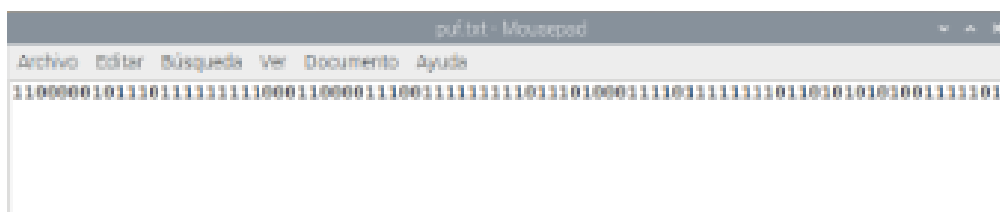


Fig. 5.2. Secuencia de bits

Una vez registrados ambos archivos, se procede a apagar la Raspberry, y tras ello se reinicia. Esto es necesario para alcanzar un nuevo estado de *power-up* nuevo, es decir, tendremos una nueva lista de bytes. Tras ello realizaremos la comparación de los nuevos bytes, extraídos en una nueva lectura de la SRAM, y los comparamos con los bytes del anterior estado, es decir, de la lista registrada en el archivo pufUNF.txt.

De esta nueva lectura encontraremos algunos bytes que han conservado su estado, los cuales simplemente añadiremos a la secuencia y otros, los cuales han cambiado y por tanto son inestables. Estos últimos serán marcados con una "X". Una vez realizada la comparación de un byte procedente de la lectura de la SRAM con una entrada del archivo pufUNF.txt, se registra el resultado en una lista local, en la que se llevara un registro de todos los resultados de la comparación. Tras realizar una comparación completa de los datos leídos de la SRAM con todos los registros del archivo pufUNF.txt, se sobrescribe el nuevo estado en el archivo pufUNF.txt, de esta manera en cada encendido este archivo tendrá de inicio la información del estado que le precede.

El objetivo es realizar este procedimiento, primero, encendido del dispositivo, segundo, comparación uno a uno de los bytes leídos en la SRAM con los del archivo pufUNF.txt, tercero, asignación del propio valor en la nueva secuencia de bytes en caso de que el

resultante de la comparación sea mismo valor o “X” en caso de que el resultado sea distinto valor, cuarto, reescribir el archivo pufUNF.txt con la nueva lista de valores, y, por último, apagar el dispositivo para proceder a encenderlo de nuevo.

Como observamos en las figuras siguientes podemos ver una funcionalidad extra en la que nos permite contar el número de iteraciones del código, así como saber si hemos alcanzado la estabilidad de bits deseada, a la vez en la propia terminal, el número de bytes estables alcanzado.

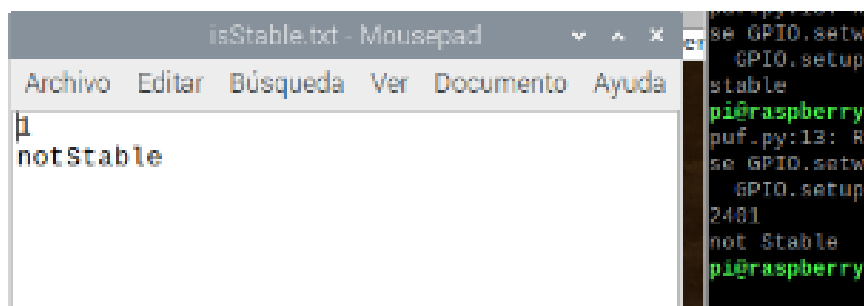


Fig. 5.3. Primera iteración del código

Como observamos en la Figura 5.3. se nos indica con el 1 que estamos en la primera iteración, así como que la lista de bytes no es estable (*notStable*), y, por último, se nos indica que hemos alcanzado 2041 bytes estables. Una vez hagamos de nuevo el mismo procedimiento, entraremos en la segunda iteración, de la que podremos obtener nuevos valores:

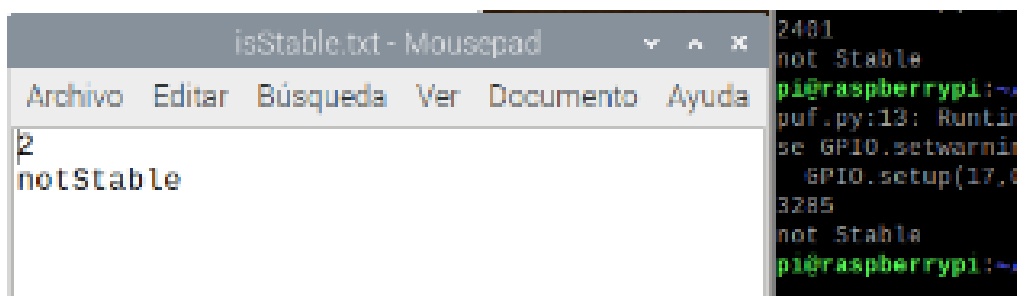
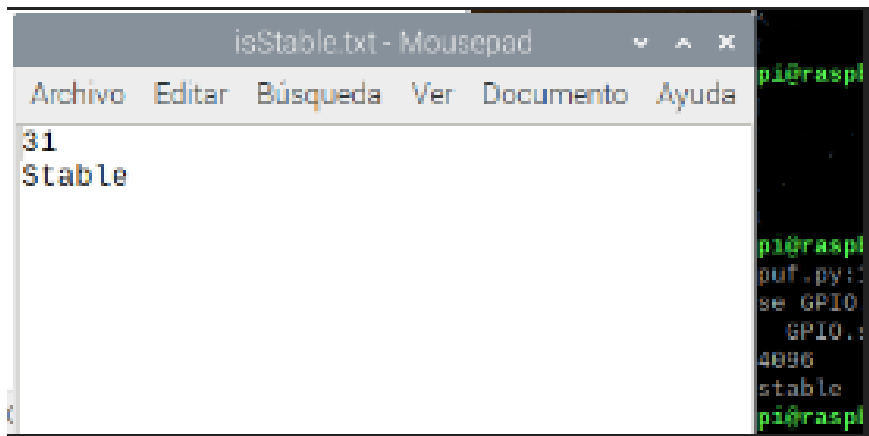


Fig. 5.4. Segunda iteración del código

De la Figura 5.4. podemos recoger nuevos valores, el 2 para indicarnos que nos encontramos en la segunda iteración del código, así como el registro de 3285 bytes

estables. Entonces repetiremos el proceso reiteradas veces hasta que se indique que hemos alcanzado la estabilidad deseada, como observamos en la Figura 5.5.

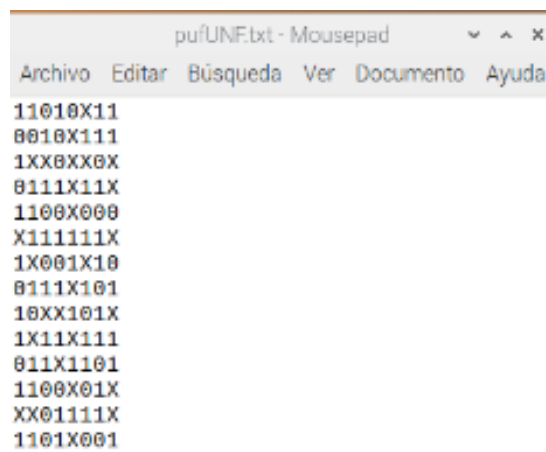


```
isStable.txt - Mousepad
Archivo Editar Búsqueda Ver Documento Ayuda
31
Stable

pi@raspl
puf.py:
se GPIO
GPIO:
4096
stable
pi@raspl
```

Fig. 5.5. Última iteración del código

Como observamos en esta última iteración, podemos comprobar que hemos llegado a la última iteración, la número 31, al haber registrado los 4096 bits estables. En esta iteración encontraremos que aquellos bits estables se representarán como su propio valor y aquellos bits inestables se representarán como una “X” (Figura 5.6).



```
pufUNF.txt - Mousepad
Archivo Editar Búsqueda Ver Documento Ayuda
11010X11
0010X111
1XX0XX0X
0111X11X
1100X000
X111111X
1X001X10
0111X101
10XX101X
1X11X111
011X1101
1100X01X
XX01111X
1101X001
```

Fig. 5.6. Representación de la tabla clasificada como bits estables o inestables

Con los resultados obtenidos de las diferentes iteraciones podemos realizar un gráfico de la evolución de la estabilidad de los bytes (Figura 5.7). En este gráfico observamos que, en un principio, el registro de los bytes estables es exponencial, hasta alcanzar una mayor estabilidad hacia la iteración 10, llegando incluso a descender el número de bytes estables

registrados en algunas iteraciones como las iteraciones 23 o la 25. Y como observamos alcanzamos los 4096 bytes estables en la iteración 31.

Podemos observar que la gráfica tiene parecido a una gráfica de un logaritmo, teniendo una gran subida en las primeras iteraciones y estabilizándose a partir de ello.

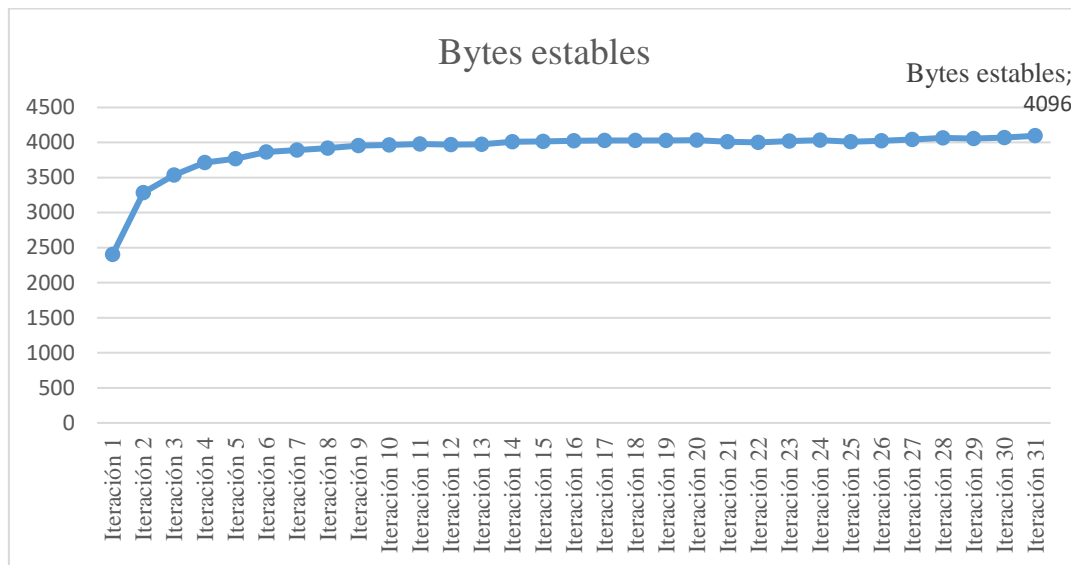


Fig. 5.7. Evolución del algoritmo hasta alcanzar la estabilidad

Como podemos observar en la gráfica, esta forma se podría trasladar a cualquier PUF, ya que, analizando la gráfica, podemos deducir que al principio una gran cantidad de bits inestables son detectados, aquellos que tienen aproximadamente un 50% de posibilidades de ser o un lógico “0” o un lógico “1”. Una vez descartados estos bits, las posibilidades de que el bit se presente como inestable decrece, ya que empieza a tender más hacia uno de los valores, de ahí que la gráfica se asemeje como a la de un logaritmo.

5.2. Segunda prueba: Inscripción y autenticación del dispositivo

Esta sección inicialmente se realizó en un entorno de *testing*, con los resultados de la prueba anterior extraídos en una carpeta creada para pruebas, y con todas las

conexiones en local, simulando un servidor y un dispositivo cliente, ambas en el entorno Eclipse.

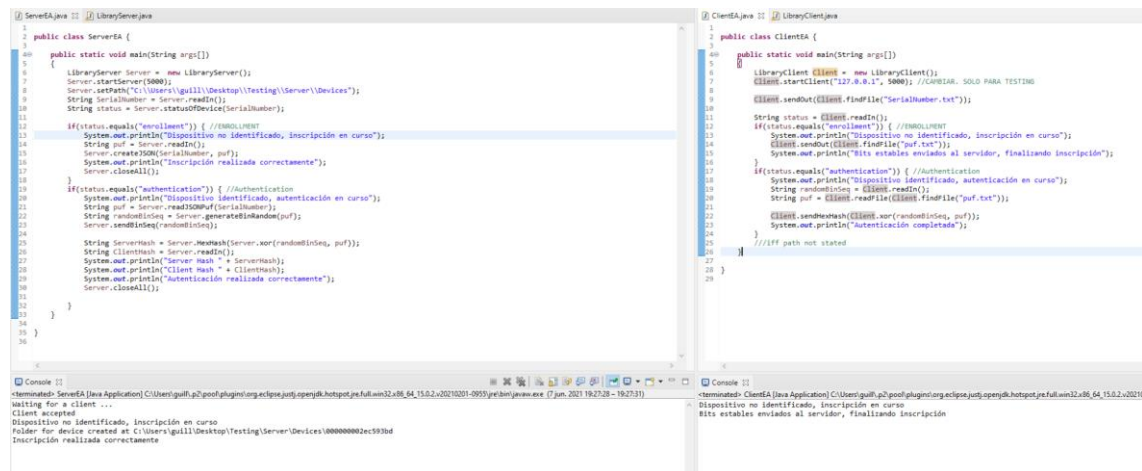


Fig. 5.8. Inscripción del dispositivo en local

Como observamos en la Figura 5.8., tanto el cliente como el servidor se encuentran en la misma ventana. El cliente tiene determinado que debe conectarse al “127.0.0.1”, es decir, el *localhost*. Observando la figura deducimos que estamos en el entorno de inscripción, a diferencia del caso de la Figura 5.9., en el que nos encontramos en la autenticación. De la Figura 5.8. también observamos como nos indica que la carpeta ha sido creada. Un pequeño detalle del código de pruebas es que en la Figura 5.9 observamos ambos *Hash*, tanto el creado por el propio servidor como el recibido por el cliente, algo que solo se programó por fines visuales, y cuya implementación quedó solo en el área de pruebas.

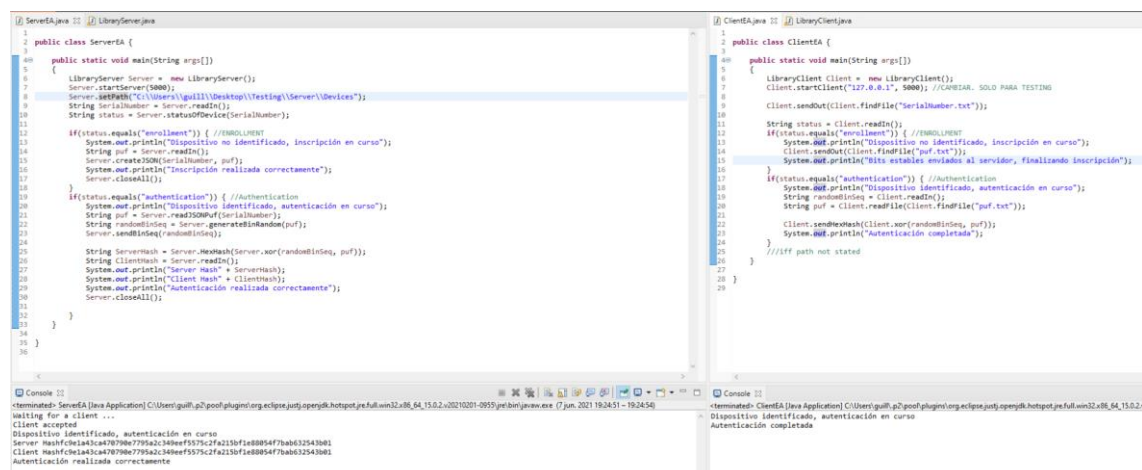


Fig. 5.9. Autenticación del dispositivo en local

Una vez certificado su funcionamiento en el área de pruebas, se debía trasladar esto al entorno del proyecto, siendo el cliente ejecutado en la Raspberry Pi, y el servidor en la máquina virtual Ubuntu. Para este traslado se tuvieron que hacer diversas modificaciones del código, las más simples, como la modificación de los *path* registrados en el código por otros, esto es debido a que las pruebas se hicieron en Windows, pero el entorno real sería en Ubuntu y Raspbian, con la ventaja de que Raspbian se asemeja a Ubuntu en muchos aspectos, y uno de ellos es la estructura de los archivos, por lo que la traducción de los mismos y la coherencia de las rutas entre cliente y servidor no fue complicada.

Pero si hubo un gran inconveniente, y es que para trasladar tanto el cliente como el servidor fuera de Windows eran necesarios exportarlos como *runnable JAR*, es decir, archivos Java ejecutables. El problema surgió entonces con los archivos del almacén de llaves (*server.jks*), por parte del servidor, y el ancla de confianza (*client.trust*), por parte del cliente. Ambos son archivos que leer dentro del programa, y con ello se establecía una de las propiedades de la fábrica del servidor TLS en la que determinabas el *path* en el que debían buscar tanto cliente como servidor, el problema surgió con la exportación, ya que JAR comprime los archivos, pero los guarda como recursos no como archivos, por lo que, al ejecutar el programa desde el JAR, este no funcionaba, lanzando una excepción indicando que no existía el archivo (Figura 5.10.). Esto era un grave problema, ya que, al no encontrar los archivos necesarios para crear los contextos, tanto el servidor como el cliente fallaban a la hora de su creación, y, por tanto, el impacto era muy grande, al no poder ni siquiera autenticar el dispositivo y, por tanto, paralizando el resto de las pruebas que dependían de una autenticación correcta.

```
Exception in thread "main" java.lang.reflect.InvocationTargetException
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:78)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:567)
    at org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoader.java:51)
Caused by: java.lang.RuntimeException: java.net.SocketException: java.security.NoSuchAlgorithmException: Error constructing implementation (algorithm: Default, provider: SunJSE, class: sun.security.ssl.SSLContextImpl$DefaultSSLContext)
    at libraryClient.startClient(LibraryClient.java:38)
    at ClientEA.main(ClientEA.java:8)
    ... 5 more
Caused by: java.net.SocketException: java.security.NoSuchAlgorithmException: Error constructing implementation (algorithm: Default, provider: SunJSE, class: sun.security.ssl.SSLContextImpl$DefaultSSLContext)
    at java.base/javax.net.ssl.DefaultSSLSocketFactory.throwException(SSLSocketFactory.java:266)
    at java.base/javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:288)
    at libraryClient.startClient(LibraryClient.java:34)
    ... 6 more
Caused by: java.security.NoSuchAlgorithmException: Error constructing implementation (algorithm: Default, provider: SunJSE, class: sun.security.ssl.SSLContextImpl$DefaultSSLContext)
    at java.base/java.security.Provider$Service.newInstance(Provider.java:1988)
    at java.base/sun.security.jca.GetInstance.getInstance(GetInstance.java:236)
    at java.base/sun.security.jca.GetInstance.getInstance(GetInstance.java:164)
    at java.base/javax.net.ssl.SSLContext.getInstance(SSLContext.java:184)
    at java.base/javax.net.ssl.SSLContext.getDefault(SSLContext.java:110)
    at java.base/javax.net.ssl.SSLSocketFactory.getDefault(SSLSocketFactory.java:83)
    at libraryClient.startClient(LibraryClient.java:32)
    ... 6 more
Caused by: java.security.KeyManagementException: problem accessing trust store
    at java.base/sun.security.ssl.SSLContextImpl$DefaultManagerHolder.<init>(SSLContextImpl.java:942)
    at java.base/sun.security.ssl.SSLContextImpl$DefaultSSLContext.<init>(SSLContextImpl.java:1112)
    at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:78)
    at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
    at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:499)
    at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:480)
    at java.base/java.security.Provider.newInstanceUtil(Provider.java:155)
    at java.base/java.security.Provider$Service.newInstance(Provider.java:1893)
    ... 12 more
```

Fig. 5.10. Pantalla de error del JAR

Para evitar el problema inicialmente se exportó el JAR, pero almacenando en local los certificados e indicando la ruta donde estaban almacenados en local. Esto conlleva problemas de seguridad, ya que el cliente se haría con el certificado como tal, y por tanto un atacante podría hacerlo también y pasarse por el cliente.

Tras descartar esta opción, aunque el código funcionase, se decidió darle una vuelta al código, y reconfigurar la manera en la que localizaba dicho recurso, para ello se utilizó la función *getResourcesAsStream*, a diferencia de la original *getResources(file).getFile()*. Esta función permitía tratar el archivo con una corriente de bits. Tras ello, se creaba un archivo temporal, que se autodestruiría al final de la sesión, en la que se recogería dicha corriente de bits y se almacenaría dentro del archivo temporal. Una vez hecho esto se recoge el *path* del nuevo archivo, y de esta manera, solucionando el problema. Para concluir dicha prueba se probaron los sistemas creados, resultando en éxito.

```
tfg@tfg-VirtualBox:~/Desktop$ java -jar ServerEA.jar
Waiting for a client ...
Client accepted
Dispositivo no identificado, inscripcion en curso
Folder for device created at /home/tfg/Devices/000000002ec593bd
Inscripcion realizada correctamente
tfg@tfg-VirtualBox:~/Desktop$ java -jar ServerEA.jar
Waiting for a client ...
Client accepted
Dispositivo identificado, autenticacion en curso
Autenticación correcta.
Jun 16, 2021 8:32:23 PM LibraryServer addSNtoLog
INFO: 000000002ec593bd
Secreto derivado y almacenado, cierre de conexion
```

Fig. 5.11. Inscripción y autenticación (Servidor)

```
pi@raspberrypi:~/Desktop/EnrollmentAndAuthentication $ java -jar ClientEA.jar
Client connected
Dispositivo no identificado, inscripción en curso
Bits estables enviados al servidor, finalizando inscripción
pi@raspberrypi:~/Desktop/EnrollmentAndAuthentication $ java -jar ClientEA.jar
Client connected
Dispositivo identificado, autenticación en curso
Autenticación correcta
Secreto derivado y almacenado, cierre de conexión
```

Fig. 5.12. Inscripción y autenticación (Cliente)

Analizando ambas figuras (Figura 5.10. y 5.11.), podemos observar cómo inicialmente se realiza la inscripción, y se crea en el servidor la carpeta con el número de serie del cliente.

Tras ello se cierra la sesión, para volver a iniciarse mismo cliente y servidor, pero en este caso el dispositivo si es reconocido, por lo que inicia la autenticación, resultando correcto y por ello, por parte del servidor, añade el número de serie al archivo *.log* con el que llevar el seguimiento de a qué dispositivo contactar, y por parte del cliente, deriva el secreto maestro que será utilizado para el contexto maestro más adelante, y tras ello cierran la conexión. También se ha hecho la prueba de que resultaría si el dispositivo no fallara en la autenticación, cuyo caso sería si el hash recibido fuera erróneo. El resultado fue también correcto, como observamos en este caso en las Figuras 5.12 y 5.13, en donde el servidor corta la comunicación sin añadir el dispositivo al *.log*, y el cliente no deriva el secreto maestro.

```
at ServerEA.Math(ServerEA.java:9)
tfg@tfg-VirtualBox:~/Desktop$ java -jar ServerEA.jar
Waiting for a client ...
Client accepted
Dispositivo identificado, autenticacion en curso
Autenticación incorrecta.
```

Fig. 5.13. Autenticación incorrecta (Servidor)

```
pi@raspberrypi:~/Desktop/EnrollmentAndAuthentication $ java -jar ClientEA.jar
Client connected
Dispositivo identificado, autenticación en curso
Autenticación incorrecta
pi@raspberrypi:~/Desktop/EnrollmentAndAuthentication $ java -jar ClientEA.jar
```

Fig. 5.14. Autenticación incorrecta (Cliente)

5.3. Tercera prueba: Comunicación OSCORE-CoAP

Esta última prueba aborda el establecimiento de la conexión entre el cliente, la Raspberry Pi, y el servidor, la máquina virtual Ubuntu. Una vez establecida la comunicación, se observó que los mensajes de *debug* se imprimían de forma caótica por pantalla como observamos en la Figura 5.10, por ello, en el resultado final, se eliminó la dependencia que imprimía dicho contenido por pantalla, lanzando un *warning*, que no interfería en la funcionalidad final y aportaba mucha más claridad.

```

C:\Windows\System32\cmd.exe - java -jar ServidorOSCORE.jar
19:01:02.181 [CoapServer(main)#2] DEBUG org.eclipse.californium.core.network.stack.ReliabilityLayer - Exchange[R2] send response null-2.05 MID= -1, Token=null, OptionSet={}, "hola"
19:01:02.183 [CoapServer(main)#2] DEBUG org.eclipse.californium.core.network.Exchange - Exchange[R2, complete]
19:01:02.183 [CoapServer(main)#2] DEBUG org.eclipse.californium.core.network.Exchange - Remote Exchange[R2, complete] completed ACK-2.04 MID=23022, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x}, ea e8 ec ae 52 82 51 63 65 53 1b 38 1d 54 1
19:01:02.188 [UDP-Sender-0.0.0.0/0.0.0.0:5683[0]] DEBUG org.eclipse.californium.elements.UDPConnector - UDPConnector (Thread[UDP-Sender-0.0.0.0:5683[0],5,Californium/Elements]) sent 28 bytes to 192.168.1.58:51328
19:01:02.204 [UDP-Receiver-0.0.0.0/0.0.0.0:5683[0]] DEBUG org.eclipse.californium.elements.UDPConnector - UDPConnector (0.0.0.0:5683) received 38 bytes from 192.168.1.58:51328
19:01:02.204 [CoapServer(main)#4] DEBUG org.eclipse.californium.core.network.deduplication.SweepDeduplicator - add exchange for KeyMID[192.168.1.58:51328-23023]
19:01:02.205 [CoapServer(main)#1] INFO org.eclipse.californium.oscore.RequestDecryptor - Removes E options from outer options which are not allowed there
19:01:02.205 [CoapServer(main)#1] INFO org.eclipse.californium.oscore.OptionJuggler - Removing inner only E options from the outer options
19:01:02.206 [CoapServer(main)#1] DEBUG org.eclipse.californium.oscore.ContextRederivation - Context re-derivation not initiated due to it being disabled for this context
19:01:02.207 [CoapServer(main)#1] DEBUG org.eclipse.californium.core.network.Exchange - Exchange[R3] replace CON-POST MID=23023, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload by CON-GET MID=23023, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload
19:01:02.207 [CoapServer(main)#1] DEBUG org.eclipse.californium.core.network.deduplication.SweepDeduplicator - Sweep run took 0ms
19:01:02.207 [CoapServer(main)#1] DEBUG org.eclipse.californium.core.network.deduplication.SweepDeduplicator - Sweep run took 0ms

C:\Windows\System32\cmd.exe - java -jar ClienteOSCORE.jar
work.Exchange - Exchange[L2, complete]
19:01:02.191 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore - coap removing Exchange[L2, complete] for token KeyToken[192.168.1.58:5683-1C8A082B4BF78D89]
19:01:02.192 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore - coap removing Exchange[L2, complete] for MID KeyMID[192.168.1.58:5683-23022]
19:01:02.192 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.Exchange - local Exchange[L2, complete] completed CON-GET MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload
19:01:02.192 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore - coap Exchange[L3] added with token KeyToken[192.168.1.58:5683-1C8A082B4BF78D89], CON-POST MID=23022, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x0900}, acked
17 f3 eb 42 bf 55 d4 58 85 c5 3e 52 cf 14 28 87 36 29 e3 9c 33 97 !
RESPONSE CODE: CONTENT 2.05
RESPONSE PAYLOAD: 68 6f 6c 61
RESPONSE TEXT: hola
19:01:02.197 [CoapEndpoint-UDP-0.0.0.0:0#1] INFO org.eclipse.californium.oscore.ObjectSecurityContextLayer - Request: CON-GET MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload
19:01:02.197 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.stack.ReliabilityLayer - Exchange[L3] send request
19:01:02.198 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.stack.ReliabilityLayer - Exchange[L3] prepare retransmission for CON-GET MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload
19:01:02.200 [CoapEndpoint-UDP-0.0.0.0:0#1] INFO org.eclipse.californium.oscore.ObjectSecurityContextLayer - Request: CON-GET MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload
19:01:02.200 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.Exchange - Exchange[L3] replace CON-GET MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x, "Uri-Path":"ComsResource"}, no payload by CON-POST MID= -1, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x0901}, 82 40 f6 4f 07 80 f0 50 1b 76 72 df 9b 02 2e b6 19 3f 6e 3b c3 51
19:01:02.201 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore - coap Exchange[L3] added with generated mid KeyMID[192.168.1.58:5683-23023], CON-POST MID=23023, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x0901}, 82 40 f6 4f 07 80 f0 50 1b 76 72 df 9b 02 2e b6 19 3f 6e 3b c3 51
19:01:02.202 [CoapEndpoint-UDP-0.0.0.0:0#1] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore - coap Exchange[L3] added with token KeyToken[192.168.1.58:5683-1C8A082B4BF78D89], CON-POST MID=23023, Token=1C8A082B4BF78D89, OptionSet={"Object-Security":0x0901}, 82 40 f6 4f 07 80 f0 50 1b 76 72 df 9b 02 2e b6 19 3f 6e 3b c3 51

```

Fig. 5.14. Terminales del Cliente y Servidor con *debug*

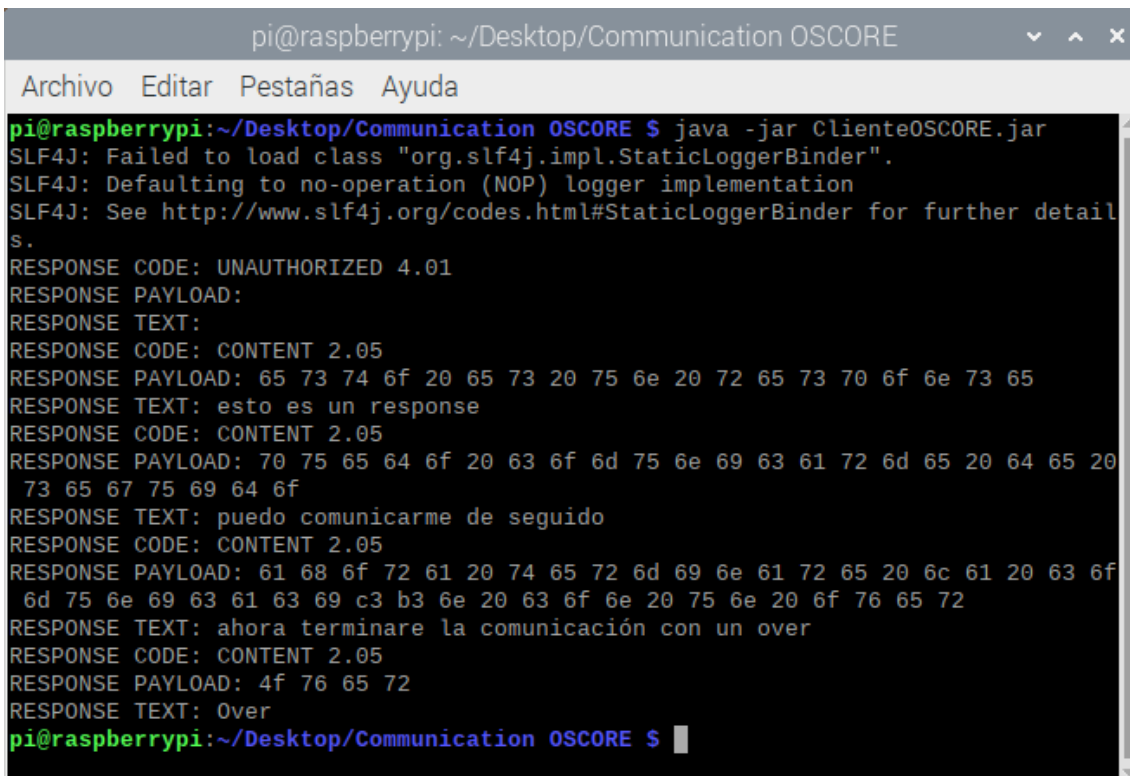
Una vez eliminada dicha funcionalidad, obtenemos lo que sería la implementación final de esta frase, teniendo la parte del servidor en la Figura 5.11. y el cliente en la Figura 5.12.

```

tfg@tfg-VirtualBox: ~/Desktop
tfg@tfg-VirtualBox:~/Desktop$ java -jar ServidorOSCORE.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Accesing ComsResource resource
esto es un response
Accesing ComsResource resource
puedo comunicarme de seguido
Accesing ComsResource resource
ahora terminare la comunicación con un over
Accesing ComsResource resource
Over
tfg@tfg-VirtualBox:~/Desktop$

```

Fig. 5.15. Terminales del Cliente



```
pi@raspberrypi: ~/Desktop/Communication OSCORE
Archivo  Editar  Pestañas  Ayuda
pi@raspberrypi:~/Desktop/Communication OSCORE $ java -jar ClienteOSCORE.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
RESPONSE CODE: UNAUTHORIZED 4.01
RESPONSE PAYLOAD:
RESPONSE TEXT:
RESPONSE CODE: CONTENT 2.05
RESPONSE PAYLOAD: 65 73 74 6f 20 65 73 20 75 6e 20 72 65 73 70 6f 6e 73 65
RESPONSE TEXT: esto es un response
RESPONSE CODE: CONTENT 2.05
RESPONSE PAYLOAD: 70 75 65 64 6f 20 63 6f 6d 75 6e 69 63 61 72 6d 65 20 64 65 20
73 65 67 75 69 64 6f
RESPONSE TEXT: puedo comunicarme de seguido
RESPONSE CODE: CONTENT 2.05
RESPONSE PAYLOAD: 61 68 6f 72 61 20 74 65 72 6d 69 6e 61 72 65 20 6c 61 20 63 6f
6d 75 6e 69 63 61 63 69 c3 b3 6e 20 63 6f 6e 20 75 6e 20 6f 76 65 72
RESPONSE TEXT: ahora terminare la comunicación con un over
RESPONSE CODE: CONTENT 2.05
RESPONSE PAYLOAD: 4f 76 65 72
RESPONSE TEXT: Over
pi@raspberrypi:~/Desktop/Communication OSCORE $
```

Fig. 5.16. Terminales del Servidor

Para terminar, podemos observar en la Figura 5.11. como en cada *Response*, el servidor accede al recurso *ComsResource*, figurando en la siguiente línea la entrada del teclado. Y dicho Response es impreso por pantalla (Figura 5.12) en el cliente, figurando como hemos indicado en la memoria el *Response Payload* y el *Response Text*, y el código indicando que la respuesta se ha recibido correctamente.

5.4. Evaluación de las pruebas

A continuación, presentaremos una tabla de los objetivos propuestos al inicio del proyecto y el resultado de las pruebas, para ello nombraremos los diferentes objetivos propuestos al inicio del proyecto, indicando si se han cumplido, y añadiendo un comentario final de cómo ha discurrido o si ha habido variaciones del planteamiento inicial. De esta manera podremos observar de manera ordenada el resultado del proyecto a presentar. Una vez terminado se procederá a hacer una conclusión de las pruebas, evaluando la complejidad de haber alcanzado los objetivos marcados.

Objetivos	Objetivo superado	Lenguaje de programación	Comentarios
Crear una librería con la que poder leer desde una Raspberry Pi una memoria SRAM a la vez que poder discernir qué bits son estables en dicha memoria.	SÍ	Python y Java	Basado en el protocolo SPI, se diseñaron diversas versiones del sistema hasta alcanzar la óptima, integrando a parte la lectura del número de serie por limpieza. Hubo diversos contratiempos debido a que se quemó la Raspberry y se perdieron los archivos.
Desarrollar los protocolos de inscripción y autenticación del dispositivo en cuestión frente a un servidor para la creación de un sistema de claves.	SÍ	Java	Utilización de sockets TLS. En un principio se diseñó como un servidor de autenticación y otro independiente de inscripción, pero se decidió adoptar esta última versión que los unía. Se independizó de la última prueba para poder evaluar procesos intermedios y seguir paso a paso el proyecto
Extender y desplegar un cliente CoAP en el que se integren las librerías dispuestas anteriormente, con el que abordar la comunicación entre un cliente y un servidor, a través de OSCORE	SÍ	Java	Se inició las pruebas con CoAP para tener una primera prueba de contacto, y se diseñó el recurso de comunicación continuada en CoAP inicialmente, siendo después traducido cuando se extendió OSCORE
Desarrollar un servidor de gestión de claves y autenticación	SÍ	Java	La implantación fue medianamente rápida pero su posterior exportación como JAR supuso la necesidad de

			reinventar el inicio del código
Desplegar y extender un servidor de comunicación OSCORE con el nuevo mecanismo de derivación de la clave	SÍ	Java	En las pruebas finales se ejecutaron ambos servidores de manera independiente, primero el de gestión de claves y después el OSCORE, ambos como archivos comprimidos <i>.jar</i> . Con ello se dio por terminada la implementación y pruebas

Tabla 5.1. Evaluación de las pruebas

Con las pruebas realizadas, podemos concluir que el resultado de todas ha sido satisfactorio, y con ello extender este resultado al conjunto del proyecto. Ha habido momentos donde ha sido bastante complicado avanzar y obtener pruebas positivas, debido a la inexperiencia en este campo, pero con el trabajo al final se ha impulsado dicho proyecto.

Capítulo 6

Entorno socio-económico y planificación

Ahora se expondrá el proceso en el que se ha desarrollado dicho sistema, es decir la planificación llevada a cabo en el mismo, el presupuesto empleado y el impacto socioeconómico del que se puede derivar dichas conclusiones. Por ello a la par de esto último se establecerá el marco regulador bajo el que se regirá el proyecto.

6.1. Entorno socio-económico

Como hemos determinado previamente, el mundo IoT está en constante expansión. Eso conlleva el surgimiento de nuevas aplicaciones y proyectos en los que esta tecnología es clave, con las temáticas de dichos proyectos siendo muy variados, desde mediciones de sensores detectando variaciones de la temperatura hasta aplicaciones biomédicas como puede ser un marcapasos o una bomba de insulina. Por ello, existe la necesidad de proporcionar seguridad a este tipo de proyectos, ya que su correcta funcionalidad puede tener un impacto en sus vidas.

El alcance de las tecnologías IoT en la actualidad es bastante más alto, por ejemplo, como hemos introducido anteriormente, este tipo de tecnologías están muy presente en aplicaciones biomédicas. En España, por cada millón de habitantes, 738 poseen un marcapasos, que permite identificar un funcionamiento irregular por parte del corazón y regularizar los latidos, también casi 6 millones de personas (8 de cada 100) son diabéticos, necesitando bombas de insulina para llevar una vida normal. Son tantas las aplicaciones que también es necesario crear un sistema seguro por detrás, ya que el fallo del mismo podría costar la vida en algunos casos. Otros casos, podrían ser como un detector de sustancias peligrosas en aguas residuales en países con condiciones precarias, lo que puede suponer la aparición de enfermedades, que en países desarrollados serían fácilmente combatidas, pero que suponen una gran mortalidad en este tipo de países, pero que, por intereses políticos, puede que se decida atacar dichos sistemas para ocultar este tipo de realidades

La implementación de la seguridad en dispositivos IoT es algo crucial, al ser una tecnología de futuro, no podemos construir dicho futuro sobre algo débil a los ataques, por lo que el proyecto propuesto puede suponer apoyo necesario a este sector de las tecnologías. Por ello, este tipo de proyectos, y en cuestión, el propuesto en esta memoria, debería estar al alcance de todo el mundo, ya que la seguridad debería ser algo necesario en un sistema, no un extra a proporcionar. Pero nos encontramos el problema de que no es así, gran parte de las PUF que se han creado, no son públicas, proyectos privados desarrolladas por diferentes empresas, sin tener muchas *open-source* PUFs.

Es por ello por lo que la contribución con un ejemplo gratuito y accesible, al alcance de todo el mundo, supone un avance en la integración o refuerzo de dicha seguridad en los dispositivos IoT, y permitirá dar saltos en el avance de estas nuevas tecnologías en sectores de técnicas más tradicionales como son la misma medicina. Este tipo de avances, con la seguridad de crear un sistema robusto, contribuyen a la expansión del mundo digital, y con ello mejorar la calidad de vida.

En general este son el tipo de aplicaciones que motivan a contribuir para mejorar la seguridad de los sistemas IoT, contribuir a convertirla en una tecnología segura y accesible con procedimientos simples y con una implementación fácil.

6.2. Planificación y presupuesto

El sistema en cuestión presenta numerosas partes para tener en cuenta, al ser un proyecto complejo. Es por ello por lo que para su correcta realización debe dividirse en diferentes secciones. Dichas secciones serán analizadas y evaluadas acorde a plazos, planificación previa, problemas encontrados y soluciones propuestas. Una vez analizado eso, podremos evaluar el presupuesto para tener en cuenta en dicho proyecto, como podemos rebajar costes y hacer más accesible a la gente la utilización de dicho proyecto.

6.2.1. Planificación del proyecto

Para empezar a abordar el proyecto fue necesario una documentación previa exhaustiva en la que poder entender y entrelazar los diferentes conocimientos propuestos. Para ello se hizo uso de artículos científicos, así como propuestos por empresas, trabajos de fin de grado y tesis doctorales abordando el asunto entre manos, y las diversas RFC (*Request for comments*) que trataban de forma extensa los temas a desarrollar. Esos estudios están claramente diferenciados por tema a tratar por lo que podemos realizar una clasificación sobre los mismos:

Estudio previo	Inicio	Fin	Días empleados	Dificultad
IoT	16-9-2020	30-9-2020	4	5
Comunicación Sockets	25-9-2020	30-9-2020	5	2
Protocolo CoAP	25-9-2020	4-10-2020	9	5
Protocolo OSCORE	6-11-2020	5-1-2021	60	8
PUF	4-3-2021	30-3-2021	26	6
Lectura de una SRAM	15-4-2021	16-4-2021	1	1
Comunicación SPI (SPIDEV)	15-4-2021	28-3-2021	13	3
Comunicación TLS	20-5-2021	23-5-2021	3	6
			Media de días por tema	Dificultad media
			15.125	4.5

Tabla 6.1. Análisis del estudio previo

Como podemos deducir observando la tabla, sacamos en claro que los estudios previos han sido bastante repartidos, y más que todo aglomerado en el espacio de un tiempo predeterminado, ha sido desarrollado a lo largo de los meses con cada sección que pudimos encontrar.

También podemos observar cómo hay una gran división de fechas, entre la parte de comunicación IoT, y la de la elaboración del sistema de gestión de claves, pudiendo destacar OSCORE como parte más complicada del proyecto, por el cual se dedicó una mayor cantidad de tiempo en proporción al resto. Se puede observar también contrastes de notas y días empleados, por lo que entendemos que la dificultad no estaba siempre en la cantidad a abordar si no los propios conceptos. Sobre los estudios previos del proyecto podemos concluir que la dificultad no reside tanto en la teoría y bases detrás de ello sino más bien en su implementación práctica.

6.2.2. Desarrollo de la comunicación CoAP/OSCORE

La implementación de esta sección es bastante directa. La utilización de la librería Californium resuelve diversos problemas que podríamos encontrar usando otras librerías. Y es que Californium ya posee, como ya habíamos nombrado, ficheros donde se realizaban diferentes pruebas y entre las que estaba la comunicación siguiendo el protocolo CoAP al igual que el OSCORE, es por ello por lo que la implementación de estos no conllevó mucho tiempo. Se utilizaron como base los archivos de prueba desde donde se construyó y testó lo creado.

Tema por desarrollar	Inicio	Fin	Días empleados	Dificultad
Protocolo CoAP	5-10-2020	6-11-2020	32	7
Protocolo OSCORE	10-1-2021	23-2-2021	44	8
			Media de días por tema	Dificultad media
			38	7.5

Tabla 6.2. Análisis de progreso de la comunicación OSCORE/CoAP

El desarrollo de esta sección del proyecto estuvo bastante repartido, esto es debido a que a pesar de la semejanza de ambos (OSCORE se puede entender como una extensión de CoAP), ambos tenían sus particularidades.

Con ello surgió el primer problema, durante la programación de CoAP, en ello hubo un retraso por el mal entendimiento del protocolo, lo que conllevó a intentar ejecutarlo desde el terminal en vez de desde Eclipse sin los ficheros necesarios para ello. Eso provocó un retraso en las fechas de finalización del protocolo, pero fue fácilmente solventado con la utilización de Eclipse en sustitución del terminal.

También durante la realización de la sección de OSCORE surgieron problemas de comprensión, pero en este caso fue por la falta de conocimiento que se pudo encontrar acerca de ello, y conllevó retrasos generalizados en la tarea.

6.2.3. Desarrollo de la generación de la PUF

En este apartado encontramos el gran escollo del proyecto que retrasó bastante las fechas. Como hemos determinado antes, se trató de realizar un primer desarrollo del proyecto mediante la utilización de la memoria DRAM, pero

Tema por desarrollar	Inicio	Fin	Días empleados	Dificultad
Generación de PUF mediante DRAM	4-3-2021	15-4-2021	42	9
Generación de PUF mediante SRAM	15-4-2021	25-5-2021	40	8
			Medía de días por tema	Dificultad media
			41	8.5

Tabla 6.3. Análisis de progreso de la extracción de la PUF

La complicación surgida con el desarrollo inicial en la DRAM supuso más de un mes de retrasos, que, a pesar de avanzar en otros temas, supuso un freno al desarrollo. Una vez abandonada esta vía se tomó la vía de la utilización de una SRAM externa con la que

realizar el proyecto, pero tras unos resultados iniciales bastante positivos, la Raspberry se fundió, haciendo que todos los archivos dentro se perdieran.

Tras ello, y con la compra y renovación de todos los componentes, se trató de implementar el código de cabeza, esto provoco el cambio de un registro, el de modo, lo que provocó que solamente fueran extraídos ceros de la SRAM y, por tanto, hasta localizar el fallo se perdieron también días.

Una vez completado los códigos se quiso simplificar y añadir cambios de calidad de lectura y mejor entendimiento del código, por lo que se volvió a empezar de cero con el código, pero esta vez fue mucho más rápido, al estar reimplementando una idea ya casi perfeccionada (ultimada en detalles). También se pasó a utilizar un solo fichero para realizar la función que antes realizaban dos, con el fin de simplificar y utilizar menos recursos de la Raspberry Pi

6.2.4. Conclusiones del desarrollo

Una vez evaluada las diferentes secciones procederemos a evaluar el trabajo al completo:

Sección del proyecto	Inicio	Fin	Días empleados	Dificultad
Estudios previos	16-9-2020	23-5-2021	249	4.5
1ª Parte: comunicación CoAP/OSCORE	5-10-2020	23-2-2021	141	7.5
2ª Parte: generación de la PUF	4-3-2021	25-5-2021	82	8.5
				Dificultad media Total
				8.5

Tabla 6.4. Análisis del proyecto

Podemos deducir que el proyecto es bastante complejo, pero como hemos mencionado antes, su complejidad no reside en la cantidad por desarrollar, si no en los conceptos a desarrollar, así como su integración en un solo proyecto, como podemos observar en la

Tabla 6.4., la segunda parte del proyecto, aunque más corta, es más compleja, esto es debido a la serie de contratiempos que surgieron a lo largo de esta sección a diferencia de la primera.

Por ello, el desarrollo ha sido bastante arduo, al introducir muchos conceptos desconocidos hasta el momento, pero permiten investigar y desarrollar un proyecto en un campo con un gran camino por evolucionar.

6.3. Presupuesto

Para la realización del proyecto se utilizó una serie de componentes ya referenciados en esta memoria. Una vez sean analizados, haremos un estudio socioeconómico del impacto que puede tener el proyecto.

6.3.1. Costes materiales del proyecto

Para analizar los costes materiales calcularemos la tasa de amortización que es:

$$Tasa = \frac{\text{Número dedicado de meses}}{\text{Periodo de depreciación del componente (60 meses)}} * \text{Coste sin IVA} * \text{Uso del dispositivo (100 \%)}$$

Componente	Precio (€)	Precio sin IVA del 21 % (€)	Tiempo dedicado en meses	Coste Final del Componente (€)
Raspberry Pi 3 Model B	32,95	26,03	6	2,60
Circuito integrado 23LC1024	1,93	1,52	4	0,10
Condensador	0,35	0,28	4	0,02
Cables	7,88	6,23	4	0,42
Protoboard	2,55	2,01	4	0,13
Ordenador	1.100	869	12	173,8
		Coste (€)		Coste final (€)
		905,07		177,07

Tabla 6.5. Presupuesto de los materiales

Los elementos que componen el proyecto son relativamente baratos, a excepción del ordenador. Pero este coste puede ser drásticamente reducido con la adquisición de un portátil de menor precio y características más reducidas, pudiendo reducir el presupuesto en unos 800 €, esto es así al no necesitar demasiados recursos para operar, clave del protocolo CoAP, al ser mensajes livianos.

Con ello podemos deducir que el sistema en cuestión es una solución bastante asequible a la hora de proteger nuestra comunicación, siendo posible replicar dicho proyecto con un bajo presupuesto en mente, así como realizar una comunicación bastante segura, y, además, teniendo en cuenta el componente hardware de dicha seguridad protegiendo dicho componente, puede suponer una mayor seguridad que un sistema de seguridad íntegramente en la nube mucho más caro de implementar.

6.3.2. Costes por persona del proyecto

El proyecto desarrollado ha estado compuesto del estudiante, de un director de proyecto y un tutor de proyecto. Por ello serán asignado diferentes puestos en función del trabajo asignado en el proyecto. Por ello el estudiante será el desarrollador de este, el gestor de proyecto o *Project Manager* será el tutor del proyecto, ya que su funcionalidad ha sido supervisar el proyecto. Y el director será el mismo director del proyecto, encargado de que el mismo siga adelante.

Posición	Salario estimado anual en bruto	Salario por hora a razón de 1.792 horas anuales[42]	Número de horas dedicadas al proyecto	Salario bruto total	Deducciones de Seguridad Social, 6,7 %	Salario neto final
Desarrollador	29.861	16,66	354	5.897,64	395,14	5.502,50
Gestor del proyecto	37.563	20,96	89	1.865,44	124,98	1.740,46
Director del proyecto	56.889	31,75	42	1.333,50	89,34	1.244,16
				Presupuesto total bruto en €		Presupuesto total neto en €
				9.096,58		8.487,12

Tabla 6.6. Presupuesto del personal

6.3.3. Costes finales del proyecto

Una vez calculados todos los costes procederemos a calcular el presupuesto total del proyecto con y sin el IVA.

Presupuesto final = Coste total del personal + Costes materiales con IVA →

*Presupuesto final = 9.096,58 + 177,07 * 1,21 →*

Presupuesto final = 9.310,83

Por tanto, el presupuesto final de este proyecto ha sido de **9.310,83 euros**.

Capítulo 7

Marco regulador

Sobre las comunicaciones IoT existen numerosas regulaciones y estándares, entre los cuales expondremos aquellos que afectan al proyecto:

- **Cybersecurity strategy of the European Union: an open, safe and secure cyberspace [24]:** se presenta la evolución de la ciberseguridad y la creación de un conjunto de normativas que tiene que cumplir cualquier servicio o recurso presente en el ciberespacio.

Para ello define unos principios entre los que se encuentran la protección de los derechos fundamentales, también de la libertad de expresión, datos personales y la privacidad. Define tras ello el derecho a un acceso global y para todos a este ciberespacio, así como una responsabilidad compartida para asegurarse que prevalece la seguridad. Por último, define que este mundo digital no debe ser controlado por una entidad o un conjunto de entidades, si no que tiene que ser desarrollado como una democracia.

- **Directiva 2013/40/UE del Parlamento Europeo y del Consejo, de 12 de agosto de 2013, relativa a los ataques contra los sistemas de información y por la que se sustituye la Decisión marco 2005/222/JAI del Consejo [25]:** A través de esta directiva reconoce la importancia social, económica y política que supone el

mundo digital, y con ello da visibilidad a la existencia de amenazas contra dicho terreno y sus consecuencias, así como su continuo aumento. Por ello establece la necesidad de crear un marco común nuevo frente a dichas amenazas, así como sanciones más severas derivadas de la ciberdelincuencia.

Esta directiva afecta de lleno al IoT, al ser un asunto crítico por la falta de desarrollo del tema, lo que provoca que sean uno de los principales objetivos de la ciberdelincuencia.

- **Reglamento (UE) 2019/881 del Parlamento Europeo y del consejo [26]:** En este documento procedente del Parlamento europeo, se reconoce la necesidad de protección frente a las amenazas en el mundo digital. También aborda el crecimiento exponencial al que está abogando el Internet de las Cosas, siendo esperado un gran número de dispositivos IoT, y como no se enfoca como debería la seguridad de estos.

Introduce la necesidad de tener cierta seguridad y garantía como cliente, con una descripción precisa y detallada sobre qué nivel de seguridad se ha implementado en dicho dispositivo, así como, incentivar a que los sectores privados adopten diversas medidas con las que ofrezcan una mayor confianza al cliente final, con el fin de promover y conseguir un mayor desarrollo e interés por el mundo IoT.

Por último, propone la existencia de una certificación común a los estados miembros de la UE de seguridad empleada en mundo IoT, con el fin de poder ofrecer dicha seguridad y facilidad a la hora de certificar dichos productos o servicios en cualquier estado miembro de la UE.

- **Directiva (UE) 2016/1148 del Parlamento Europeo y del Consejo también denominada Directiva de la Unión Europea NIS [27]:** Establece el marco legal de Protección de **servicios esenciales que dependen de servicios digitales** (según LSSI) que sean mercados en línea, motores de búsqueda o servicios *Cloud* y es la legislación europea que después se desarrolla en cada país de la Unión con legislaciones específicas. Todas ellas razonablemente similares. Incide en la notificación de incidentes (especialmente con impacto transfronterizo), establece

infracciones y sanciones. Indica que los operadores de servicios esenciales (OSE) se agrupan en los mismos sectores estratégicos, y los designa a cada uno, una autoridad competente. Los proveedores de servicios digitales (PSD) han de comunicar su actividad proactivamente. Establece el rol del Responsable de Seguridad de la Información en las organizaciones sujetas a la directiva. En 2016 Entra en vigor de la **Directiva NIS** con medidas destinadas a garantizar un elevado nivel común de seguridad de las redes y sistemas de información de la Unión (**servicios esenciales y servicios digitales**). En 2018 se transpone la **Directiva NIS** en España y se identifican parte de los servicios esenciales y operadores de sectores estratégicos. Esta identificación se completa en 2019.

- **Leyes de Protección de las Infraestructuras Críticas 8/2011, Reglamento de protección de las infraestructuras críticas (Real Decreto RD704/2011, de 20 de mayo) [28] y Real Decreto 43/2021 de 26 de enero [29]:** Se trata de la Ley desarrollada en España para la protección de las Infraestructuras Críticas nacionales ante amenazas deliberadas (físicas y digitales). Este marco legal es similar al que se ha desarrollado en otros países de nuestro entorno y de Sudamérica con el mismo objetivo. Impulsa la colaboración entre los operadores de infraestructuras críticas, las Administraciones Públicas y las fuerzas y cuerpos de seguridad del estado. Los operadores se agrupan en sectores estratégicos (telecomunicación, energía, banca, salud, ...) y sus instalaciones críticas son designadas oficialmente, si bien esta designación no es pública. Establece el rol del responsable de Seguridad y Enlace en las empresas afectadas (habilitado como Director de Seguridad por el Ministerio del Interior). En 2005 La Secretaría de Estado de Seguridad, del Ministerio del Interior, aprobó el Plan Nacional para la Protección de **Infraestructuras Críticas**. Posteriormente, en 2007, El Consejo de Ministros aprobaría la creación del Centro Nacional de Protección de Infraestructuras Críticas (CNPIC). En 2008, se redactó la Directiva 2008/114/CE del Consejo, de 8 de diciembre de 2008, sobre la identificación y designación de infraestructuras críticas. En 2011, Ley 8/2011 que establece medidas para la protección de las infraestructuras críticas. Finalmente, en 2015, Se aprueban los nuevos contenidos mínimos de los Planes de Seguridad del Operador y de los Planes de Protección Específicos sobre infraestructuras críticas. La ley establece determinados actores involucrados en las actividades de protección: El Centro

Nacional de Protección de Infraestructuras Críticas (CNPIC): impulso, supervisión y coordinación. La Oficina de Coordinación de Ciberseguridad (OCC): asume la función de supervisión y coordinación en materia de ciberseguridad. El CERTSI – INCIBE: apoyo operativo ante ciberamenazas y ciberincidentes.

El Centro de Inteligencia contra el Terrorismo y el Crimen Organizado: establece el nivel de alerta, que determina medidas de protección y reporte. Por último, están las Fuerzas y Cuerpos de Seguridad del Estado (FFCCSE): protección física de infraestructuras críticas. La ley establece determinados requerimientos a las empresas que gestionan infraestructuras críticas: Designar formalmente un Responsable de Seguridad y Enlace (habilitado por el Ministerio del Interior como Director de Seguridad), designar un Delegado de Seguridad por cada instalación y un Responsable de Seguridad TI. Entregar un Plan de Seguridad del Operador (PSO), conforme a unos contenidos mínimos establecidos : Política de Seguridad y Marco de Gobierno, Relación de Servicios Esenciales prestados el operador, Metodología de Análisis de Riesgos (amenazas físicas y de ciberseguridad), Criterios de aplicación de Medidas de Seguridad Integral. Entregar un Plan de Protección Específico (PPE) **por cada instalación crítica**, conforme a unos contenidos mínimos establecidos: Los aspectos organizativos desde un punto de vista de Seguridad Integral la descripción de la infraestructura crítica los resultados de los diferentes análisis de riesgos realizados. el plan de acción propuesto por activo para la instalación industrial. Estos planes deben actualizarse cuando se produzcan cambios significativos o como mucho, 2 años después de su aprobación. En 2021 se desarrolla el Real Decreto 43/2021 que establece la aplicación de la directiva NIS al mercado español y concreta sus requerimientos y obligaciones. Sus requerimientos están muy alineados con legislaciones previas y que se han comentado en la primera parte de este punto. La directiva NIS establece los siguientes actores: Oficina de Coordinación de Ciberseguridad (OCC): Autoridad competente para operadores críticos públicos y privados. Centro Criptológico Nacional (CCN): Autoridad competente para OSE y PSD que no sean críticos y sean del sector público. Autoridades competentes de cada sector estratégico para operadores no críticos ni públicos (Ministerios, Banco de España, Consejo de Seguridad Nuclear, ...). CERTSI-INCIBE: CSIRT de los OSE y PSD

privados. Departamento de Seguridad Nacional: Punto de Contacto Único para comunicaciones de los CSIRT a otros estados de la Unión Europea. La directiva requiere designar formalmente un **Responsable de Seguridad de la Información** que ejercerá las funciones de punto de contacto y coordinación técnica con la autoridad competente y CSIRT de referencia. (Esto no aplica para los PSD). Se debe entregar una **Declaración de Aplicabilidad** para cada instalación que sustenta el servicio esencial. Indica que las medidas de seguridad que deberán estar en funcionamiento tomarán como referencia normativas y buenas prácticas ya existentes. Como referencia, se propone el Esquema Nacional de Seguridad (ENS), definido para el ámbito de la administración pública. Se establece la obligatoriedad de la **notificación de incidentes** de determinado nivel, de acuerdo con la Guía Nacional de Gestión y Notificación de Incidentes. Establece **un régimen sancionador**.

- **La Estrategia de Ciberseguridad Nacional. La Orden PCI/487/2019, de 26 de abril [30], es la norma a través de la cual se publica la Estrategia Nacional de Ciberseguridad (en adelante, ENCS).** La ENCS se trata de la base del modelo a aplicar en el intercambio de recursos entre el Estado con las diferentes empresas tanto privadas como públicas, así como con los ciudadanos. Para la consecución positiva de dicho intercambio se determinó una estructura en la que construir, ligada al Sistema de Seguridad Nacional, y que permite operar de forma coordinada las operaciones del Estado, con las diferentes operaciones llevadas a cabo con el resto de los integrantes.

Esta Estrategia se basa en los diferentes principios de la ciberseguridad, entre los que encontramos la resiliencia, que permita operar sin perturbaciones a pesar de que se produzcan diferentes ataques, la anticipación, que nos permite prevenir diferentes ataques o tomar las medidas necesarias de antemano con la que reducir el impacto de dichos ataques, la unidad de acción, con la que afrontar de manera conjunta diferentes situaciones en las que más de una parte sea afectada, y la eficiencia, que permite actuar de forma rápida frente a ataques. Con ellos se establece un objetivo principal que se trata de: “Seguridad y resiliencia de las redes y los sistemas de información y comunicaciones del sector público y de los servicios esenciales”, en el cual se establece la importancia de que las diferentes

partes que conforman la ENCS se comprometan a modernizar y mejorar los sistemas de ciberseguridad en los que se apoyan, así como emplear un comportamiento modélico de implementación de la seguridad en el sistema. También se define el modelo de responsabilidad compartida, entre las partes de dicha Estrategia, en el que realizar un intercambio de información, así como de avisos de posibles ataques, o soluciones a ataques ya sufridos con el que reducir el impacto general que pueda tener.

- **Otros estándares y marcos de referencia en entorno de ciberseguridad industrial:** ISO 27001 [31], ISO 27002 [32], ISA99/IEC 62443 [33], NIST 800-82 [34]. ISA99/IEC62443 es el marco de referencia internacional para sistemas OT que se focaliza en aspectos de integridad y disponibilidad de los sistemas industriales, estableciendo requerimientos a lo largo de su ciclo de vida, Las normas ISO 27001 y 27002 establecen estándares a seguir en materia de seguridad de la información. La referencia NIST 800-82 establece estándares de seguridad industrial en sistemas de control. Aporta arquitecturas de referencia, amenazas y vulnerabilidades específicas de estos sistemas e identifica posibles medidas mitigantes.
- **Recientemente se ha publicado una Guía de Controles de Seguridad en Sistemas OT por parte de la Secretaría de Estado de la Seguridad [35].** No se trata de una ley si no de una guía para ayudar a determinar el impacto de las múltiples leyes existentes en los sistemas OT. Desarrolla las medidas básicas de Seguridad en entornos industriales que son: la asignación de determinados roles y responsabilidades en cada instalación en esta materia, disponer de una estrategia de seguridad, realizar un inventario de los activos (SCI sistemas de control industrial) de la instalación y mantenerlo actualizado, indica los requerimientos de control del acceso lógico a los sistemas de control, establece los requerimientos de configuración de los equipos SCI, determina las arquitecturas marco de las redes OT, indica que estas redes deben estar segregadas de las redes IT, establece guías de trazabilidad de la actividad de los usuarios en los SCI y determina el modelo de gestión de incidentes y vulnerabilidades. También indica guías de medidas antimalware, medidas preventivas y reactivas ante ataques de denegación de servicio, medidas de seguridad ante el uso de redes Wifi corporativas,

procedimientos de parcheo para aplicaciones, firmware y sistemas operativos, requerimientos de seguridad de los equipos portátiles y dispositivos móviles que acceden a las redes OT, guías de protocolos seguros, controles para dispositivos extraíbles, controles y requerimientos de supervisión del acceso remoto. También describe el plan de respuesta a incidentes, necesidad de capacidades de copia y restauración, control estricto sobre roles de administración y gestión del cambio sobre los SCI, gestión de archivos de registro y pistas de auditoría y seguridad en la cadena de suministro. Por último, menciona la necesidad y prioridad que tiene la formación y concienciación del personal y las medidas de seguridad física en las instalaciones donde se ubiquen los SCI.

Capítulo 8

Conclusión y líneas futuras

Como se puede discernir del proyecto elaborado, este ha tratado de elaborar una posible solución al problema de seguridad actual en los dispositivos IoT, para ello se ha implementado un mecanismo de gestión de claves para OSCORE, en el que el secreto maestro del contexto de dicho protocolo ha sido modificado para varias de manera dinámica, con ello creando un sistema mucho más complejo y robusto frente a amenazas y ataques que puedan darse, dando una mayor confianza.

8.1. Revisión de los objetivos propuestos

En el inicio del proyecto se establecieron diversos objetivos a conseguir a lo largo del proyecto. Estos se dividían en las diferentes fases a abordar en función de los aspectos que abarcaban en el proyecto.

Inicialmente se estableció la necesidad de crear un algoritmo con el que determinar los bits estables de una SRAM para proporcionar a la Raspberry Pi una señal de identidad frente al servidor, este objetivo fue perfeccionado en gran medida, habiendo múltiples versiones de los elementos que lo componen, hasta alcanzar como estaba establecido un sistema para determinar que bits eran estables. Tras ello, se determina la creación de un sistema de inscripción y autenticación de la Raspberry frente al servidor, que fue la

sección con mayor evolución a lo largo del proyecto, con cambios en el lenguaje de programación en el que se basaba incluso, se terminó con la implementación de dos librerías bastante completas con las que se podría hacer todas las operaciones pertinentes por parte de tanto el servidor como el cliente con el fin de realizar una inscripción y autenticación correcta. Por último, fue el turno de la comunicación entre los dispositivos, llevada a cabo mediante los protocolos CoAP y OSCORE, los cuales habían sido modificados para la implementación que se consideró en el proyecto, que permitieron completar el sistema complejo que se había propuesto en un inicio.

Este proyecto ha abarcado múltiples secciones y funcionalidades, lo que ha supuesto un gran aprendizaje del mismo y del entorno en el que se desarrolla, profundizando en un mundo para muchos aún desconocido como es el mundo de los dispositivos interconectados, el Internet de las Cosas, siendo este proyecto solo la punta del iceberg de un gran y complejo desarrollo que ya hay hecho sobre él, así como del que aún queda por desarrollarse, debido a que este mundo es el mundo del futuro, pero también del presente.

8.2. Líneas futuras

Este proyecto se puede considerar el inicio de implementación mucho más compleja, teniendo aun un gran margen de desarrollo e investigación de este. Algunos de los posibles pasos a seguir son:

- **Implementación de una base de datos en el servidor**, en sustitución del repositorio en el sistema de fichero de la información de los dispositivos. Aunque su ejecución pudiera ser más lenta al precisar de más pasos, esto podría aportar una mayor escalabilidad del sistema.
- **Testing de la robustez del algoritmo mediante Machine Learning**, ya que como hemos nombrado anteriormente, uno de los ataques posibles se da por la predicción de los bits estables. Un mayor conocimiento de la robustez de la PUF nos podrá dar una mejor idea de cómo extender nuestro proyecto.
- **Integración de un sistema con un sensor para la comunicación**, esto nos permitiría simular una situación cotidiana del proyecto, pudiendo determinar sus

tiempos de respuesta, con el fin de optimizar aquello que sea necesario y perfeccionar el algoritmo.

- **Introducción de más de una SRAM para determinar la PUF**, lo cual aumentaría de manera exponencial la seguridad del sistema ya que el atacante tendría que saber determinar, como por ejemplo con *Machine Learning* como enunciamos antes, no solo el patrón de una si no varias SRAM, algo altamente imposible.
- **Perfeccionar la escalabilidad de las comunicaciones**, actualmente el sistema sirve para comunicarse con un dispositivo a la vez, esto es debido a la posesión de una sola Raspberry Pi a la hora de realizar el proyecto, lo que impedía determinar cuánto de fiable sería el sistema con varios dispositivos a la vez. Por ello, con la incorporación de más dispositivos al sistema nos permitiría hacer crecer el proyecto, creando una red mucho mayor.

Por ello a través de estas posibles ideas más muchas otras tantas, se puede hacer crecer en gran medida este proyecto, y con ello contribuir a un mundo digital más seguro y colaborar con el crecimiento y expansión del Internet de las Cosas, lo que conllevaría un incremento de la calidad de vida de las personas, así como ayudar en aquellos aspectos donde sea requerido para llevar una vida de calidad excelente.

Referencias

- [1] "Ciberseguridad en Infraestructuras Críticas e Industria 4.0 - Centro de Ciberseguridad Industrial", *Centro de Ciberseguridad Industrial*, 2021. [Online]. Available: <https://www.cci-es.org/activities/ciberseguridad-en-infraestructuras-criticas-e-industria-4-0/>. [Accessed: 21- Feb- 2021]
- [2] C. Duggan, "Lessons Learned from the Colonial Pipeline Attack - Embedded Computing Design", *Embedded Computing Design*, 2021. [Online]. Available: <https://www.embeddedcomputing.com/technology/iot/wireless-sensor-networks/lessons-learned-from-the-colonial-pipeline-attack>. [Accessed: 01- Jun- 2021]
- [3] M. Wainwright, "The Florida water plant attack signals a new era of digital warfare — it's time to fight back", *Darktrace.com*, 2021. [Online]. Available: <https://www.darktrace.com/en/blog/the-florida-water-plant-attack-signals-a-new-era-of-digital-warfare-its-time-to-fight-back>. [Accessed: 01- Jun- 2021]
- [4] C. McLellan, "How hackers attacked Ukraine's power grid: Implications for Industrial IoT security | ZDNet", *ZDNet*, 2021. [Online]. Available: <https://www.zdnet.com/article/how-hackers-attacked-ukraines-power-grid-implications-for-industrial-iot-security/>. [Accessed: 01- Jun- 2021]
- [5] "El virus que tomó control de mil máquinas y les ordenó autodestruirse - BBC News

Mundo", *BBC News Mundo*, 2021. [Online]. Available: https://www.bbc.com/mundo/noticias/2015/10/151007_iwonder_finde_tecnologia_virus_stuxnet. [Accessed: 01- Jun- 2021]

[6] M. Valle, "Sacan a la luz el mayor ciberataque de la historia - Globb Security", *Globb Security*, 2015. [Online]. Available: <https://globbsecurity.com/aramco-mayor-ciberataque-historia-35593/>. [Accessed: 02- May- 2021]

[7] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

[8] M. Iglesias-Urkia, A. Orive and A. Urbietta, "Analysis of CoAP Implementations for Industrial Internet of Things: A Survey", *ScienceDirect*, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050917309870?via%3Dihub>. [Accessed: 01- Jun- 2021]

[9] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

[10] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

[11] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

[12] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

- [13] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.
- [14] Sönke, F. Hinz, "eknoes/puf-lab", *GitHub*, 2019. [Online]. Available: <https://github.com/eknoes/puf-lab>. [Accessed: 02- Feb- 2021]
- [15] "SRAM PUF - Intrinsic ID | Home of PUF Technology", *Intrinsic ID / Home of PUF Technology*, 2021. [Online]. Available: <https://www.intrinsic-id.com/sram-puf/>. [Accessed: 01- May- 2021]
- [16] "Serial Peripheral Interface | Wikiwand", *Wikiwand*, 2021. [Online]. Available: https://www.wikiwand.com/es/Serial_Peripheral_Interface. [Accessed: 06- Apr- 2021]
- [17] "23A1024/23LC1024", *Microchip*, 2021. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/20005142C.pdf>. [Accessed: 06- Apr- 2021]
- [18] MIT - Massachusetts Institute of Technology "spidev 3.5", *PyPI*, 2012. [Online]. Available: <https://pypi.org/project/spidev/>. [Accessed: 09- Apr- 2021]
- [19] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [20] sysadmin1138, "What is a Pem file and how does it differ from other OpenSSL Generated Key File Formats?", *Server Fault*, 2018. [Online]. Available: <https://serverfault.com/questions/9708/what-is-a-pem-file-and-how-does-it-differ-from-other-openssl-generated-key-file>. [Accessed: 09- Jun- 2021]
- [21] Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC 1422, DOI 10.17487/RFC1422, February 1993, <<https://www.rfc-editor.org/info/rfc1422>>.

[22] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, DOI 10.17487/RFC7292, July 2014, <<https://www.rfc-editor.org/info/rfc7292>>.

[23] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <<https://www.rfc-editor.org/info/rfc5914>>.

[24] Cybersecurity Strategy of the European Union: An Open, Safe and Secure Cyberspace
<https://eeas.europa.eu/archives/docs/policies/eu-cyber-security/cybsec_comm_en.pdf>.

[25] Directiva 2013/40/UE del Parlamento Europeo y del Consejo, de 12 de agosto de 2013, relativa a los ataques contra los sistemas de información y por la que se sustituye la Decisión marco 2005/222/JAI del Consejo.
<<https://www.boe.es/doue/2013/218/L00008-00014.pdf>>.

[26] REGLAMENTO (UE) 2019/881 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 17 de abril de 2019 relativo a ENISA (Agencia de la Unión Europea para la Ciberseguridad) y a la certificación de la ciberseguridad de las tecnologías de la información y la comunicación y por el que se deroga el Reglamento (UE) n. o 526/2013 («Reglamento sobre la Ciberseguridad»)
<<https://www.boe.es/buscar/doc.php?id=DOUE-L-2019-80998>>.

[27] Directiva (UE) 2016/1148 del Parlamento Europeo y del Consejo, de 6 de julio de 2016, relativa a las medidas destinadas a garantizar un elevado nivel común de seguridad de las redes y sistemas de información en la Unión
<<https://www.boe.es/doue/2016/194/L00001-00030.pdf>>.

[28] Real Decreto 704/2011, de 20 de mayo, por el que se aprueba el Reglamento de protección de las infraestructuras críticas.
<<https://www.boe.es/eli/es/rd/2011/05/20/704/con>>.

[29] Real Decreto 43/2021, de 26 de enero, por el que se desarrolla el Real Decreto-ley 12/2018, de 7 de septiembre, de seguridad de las redes y sistemas de información. <<https://www.boe.es/eli/es/rd/2021/01/26/43>>.

[30] Orden PCI/487/2019, de 26 de abril, por la que se publica la Estrategia Nacional de Ciberseguridad 2019, aprobada por el Consejo de Seguridad Nacional. <<https://www.boe.es/eli/es/o/2019/04/26/pci487>>.

[31] "ISO 27001 - Software ISO 27001 de Sistemas de Gestión", *Software ISO*, 2021. [Online]. Available: <https://www.isotools.org/normas/riesgos-y-seguridad/iso-27001/>. [Accessed: 04- Jun- 2021]

[32] "ISO 27002: The Code of Practice for Information Security Controls", *ISMS.online*, 2021. [Online]. Available: <https://www.isms.online/iso-27002/>. [Accessed: 04- Jun- 2021]

[33] "El estándar ISA99/IEC62443", *Secure&IT*, 2021. [Online]. Available: <https://www.secureit.es/ciberseguridad-industrial/el-estandar-isa99-iec62443/>. [Accessed: 04- Jun- 2021]

[34] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams and A. Hahn, *Guide to Industrial Control Systems (ICS) Security*, 2nd ed. 2021 [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf>. [Accessed: 04- Jun- 2021]

[35] Ministerio del Interior, "Guías sobre controles de seguridad de sistemas OT", *Ismsforum.es*, 2021. [Online]. Available: <https://www.ismsforum.es/ficheros/descargas/Gu%c3%ada%20OT.pdf>. [Accessed: 04- Jun- 2021]

[36] A. El Maataoui Nahal, "Análisis de Funciones Físicas Inclonables (PUFS) en FPGA", *Universidad Carlos III de Madrid*, 2017 [Online]. Available: https://e-archivo.uc3m.es/bitstream/handle/10016/27628/TFG_Ayoub_El_Maataoui_Nahal_2017.pdf?sequence=1&isAllowed=y

- [37] W. Che, F. Saqib and J. Plusquellic, "PUF-based authentication," 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2015, pp. [Online]. Available: <https://ieeexplore.ieee.org/document/7372589>.
- [38] A. Braeken, "PUF Based Authentication Protocol for IoT", *MDPI*, 2018. [Online]. Available: <https://www.mdpi.com/journal/symmetry>. [Accessed: 20- Jun- 2021]
- [39] S. Perez, J. Hernandez-Ramos, S. Raza and A. Skarmeta, "Application Layer Key Establishment for End-to-End Security in IoT", *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2117-2128, 2020 [Online]. Available: https://www.researchgate.net/publication/337937077_Application_Layer_Key_Establishment_for_End-to-End_Security_in_IoT_IEEE_Internet_of_Things_journal. [Accessed: 20- Jun- 2021]
- [40] M. Tiloca, F. Palombini and J. Park, "Key Management for OSCORE Groups in ACE", *Semanticscholar.org*, 2021. [Online]. Available: <https://www.semanticscholar.org/paper/Key-Management-for-OSCORE-Groups-in-ACE-Tiloca-Palombini/bb59c90e006870cecba8b98f8a8e91bb17ffdfbe>. [Accessed: 20- Jun- 2021]
- [41] "An Application-Layer Approach to End-to-End Security for the Internet of Things", *OMA SpecWorks*, 2019. [Online]. Available: <https://omaspecworks.org/end-to-end-security-for-the-internet-of-things/>. [Accessed: 20- Jun- 2021]
- [42] "BOE-A-2021-1908 Resolución de 29 de enero de 2021, de la Dirección General de Trabajo, por la que se registra y publica el Convenio colectivo de Balkanica Distral, SL.", *Boe.es*, 2021. [Online]. Available: [https://www.boe.es/eli/es/res/2021/01/29/\(3\)](https://www.boe.es/eli/es/res/2021/01/29/(3)). [Accessed: 20- Jun- 2021]

Anexos

1. Summary

For the following pages, a summary of the project will be presented. As a requirement for this project, this summary will be written in English in its entirety.

1.1. Introduction

This project is designed to introduce greater security in communication with Internet of Things devices. These devices lack a strong security system, this is due to some of their characteristics, among which are the low power at which they can operate, the need to use light resources.

That is why, with the exponential growth of this technology sector, further research and development of projects is necessary according to the theme. The world of the Internet of Things is spreading very quickly, and with it the attacks on this type of technology and its consequent communications. Techniques like *eavesdropping*, trying to listen to a private communication, DDOS (*Denial of Service*), network saturation with multiple *Requests*, *Spoofing*, identity theft or *Sniffing*, passing through the device in front of the server, reveal the multiple threats that IoT devices face.

And due to their low robustness, they are one of the most important objectives to be taken into account by any attacker, which has resulted in attacks that have caused great losses, as well as damage, as we can see in the example of the cyberattack suffered by The Colonial Pipeline Company [2], which due to a *malware* introduced in the IT network, had to preventively disconnect certain systems in the IT and OT network, which caused the paralysis of the pipeline operations and led to different errors in the distribution of gas and oil throughout the East coast of the United States, or even to provoke even political consequences such as the attack that promoted an increase in tensions between Ukraine and Russia [4], in which through *phishing* entered the network, and once inside, they attacked using a *exploit*, which led to the failure of electricity to reach many homes in Ukraine, something critical, causing various blackouts. This attack was perpetrated by cyber attackers related to or backed by the Russian government, which, as we mentioned before, led to an increase in tensions that already existed between the countries.

For all these examples and for more that there has been and for having it is necessary to make a greater development in the security of this type of systems, for what to design a security system for the Internet of Things.

1.2. Objectives

The objective of this work is to implement a hardware-based key management system, with which to establish secure communication between a client and server. For this, the OSCORE protocol (Object Security for Constrained RESTful Environments) will be integrated with which to encode the communication that together with CoAP (Constrained Application Protocol) allow to carry out a secure communication between a Raspberry Pi and a server on a computer, with the particularity that OSCORE protects the data to be transferred and the Request / Response method, leaving the rest unprotected, which allows us to send lighter messages.

To give a greater layer of security, instead of having the master key with which the communication will be carried out in the hardcoded code or in a file, a system will be created with which the key will be derived. This will be done with a PUF (Physical

Unclonable Function). This procedure takes advantage of the imperfections produced during the manufacture of the SRAM memory, which allows, depending on the stability of its bits, to create a unique memory map, impossible to repeat with another SRAM, although it has the same characteristics.

1.3. System Implementation

The project in question will consist of three differentiated parts, the first will be the extraction of the stable bits from the SRAM by a Raspberry Pi 3 Model B. For this, a PUF will be programmed, which, by means of repetitive reboots of the device, It will allow us to achieve the stability of the sequence until we achieve it.

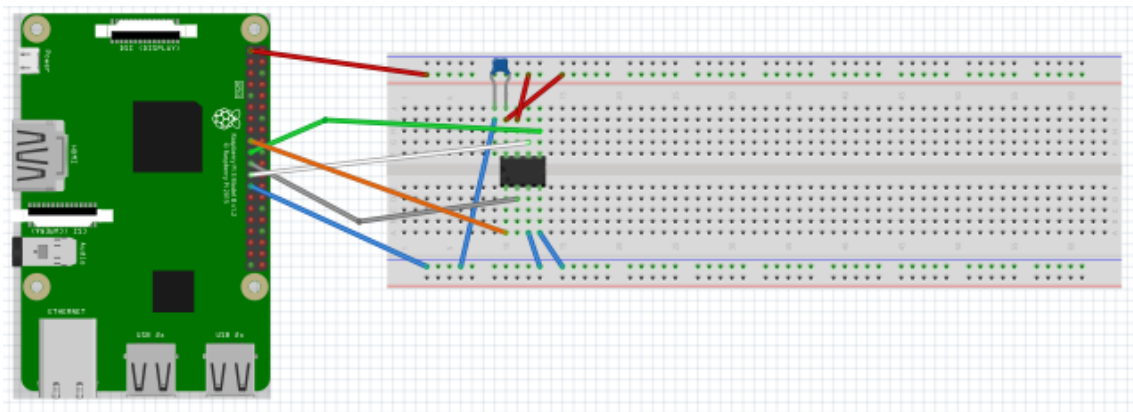


Fig. A.1. Raspberry-SRAM connection with 0.1 μ F capacitor

The second part will verify the registration of said device to a server, by sending the stable bits obtained in the previous phase, and after that an authentication, where it will be tested, if the device is who it claims to be in front of the server.

And if it is successful, the last phase will take place, communication. Based on CoAP, an effective communication protocol in highly restricted devices, and extended through OSCORE, providing a necessary security context to carry out a more secure communication between the different devices.

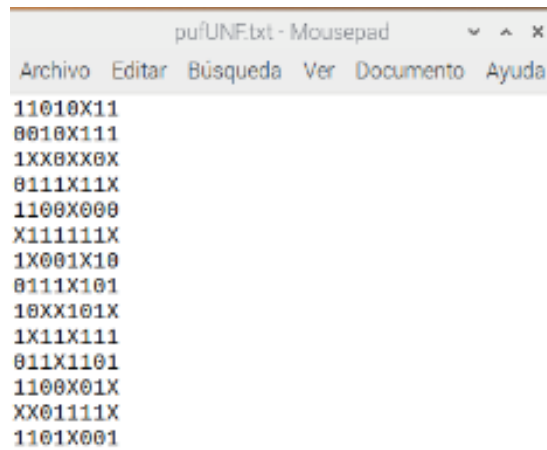
After that the whole system will be implemented as one, with the three different parts being interconnected but working independently from one another.

1.3.1. PUF generation

The second part will verify the registration of said device to a server, by sending the stable bits obtained in the previous phase, and after that an authentication, where it will be tested, if the device is who it claims to be in front of the server .

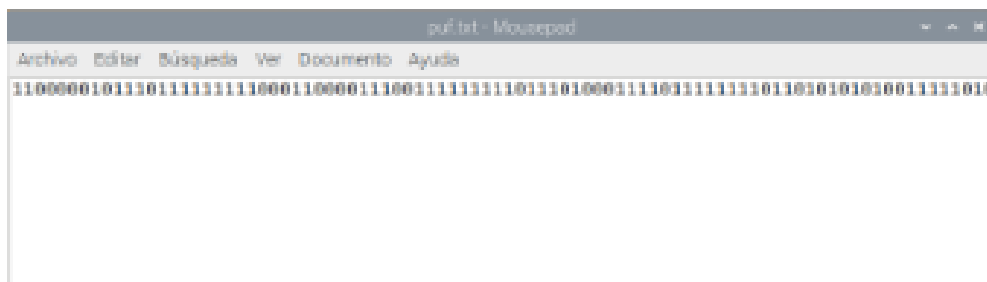
And if it is successful, the last phase will take place, communication. Based on CoAP, an effective communication protocol in highly restricted devices, and extended through OSCORE, providing a necessary security context to carry out a more secure communication between the different devices. *chip select*, which defines which device has the turn to transmit or receive the information.

Through the code developed puf.py, a sequence of stable bits will be obtained in a file, as well as a diagram of which bits are stable for each 8-bit address, and which are not, the latter by designating it as "X", as we can see in the figure below.



11010X11
0010X111
1XX0XX0X
0111X11X
1100X000
X111111X
1X001X10
0111X101
10XX101X
1X11X111
011X1101
1100X01X
XX01111X
1101X001

Fig. A.2. Table representation classified as stable or unstable bits



```
110000010111011111111000110000111001111111101110100011100111111110110101010011111011
```

Fig. A.3 Bit stream

This will be possible by reading the bits from the SRAM output, but to do this initially, you must determine which section of memory needs to be read, as well as the number of bytes to read and the reading mode. . For this, the methods used are *writebytes* and *readbytes*, defined in the Spidev library, as well as other configuration of the operation. For the *chip select*, the operation was carried out manually using the GPIO library (*General Purpose Input/Output*), This is due to different errors caused in reading with the SPI port dedicated to the *chip select* on the Raspberry Pi itself.

1.3.2. Enrollment and Authentication

In the second phase, the one corresponding to the registration and authentication of the system, Java was used, due to its ease of implementing TLS Sockets, which are necessary to protect the transfer of bits from the client to the server, through TCP. , ensuring that the information and resources found in said communication flow are not modified, as well as preventing other devices from getting in the way of communication, that is, that they comply with the principles of authentication, confidentiality and integrity. since, in the event of an attacker, this communication will be compromised.

To carry out this communication, two certificates are required, one from the server that is the *server.pem* A certificate that includes the keys, as well as other certificates, such as root certificates, that identify the authority that issues the certificates, from which the one to be used in the implementation is derived, the *server.pkcs12* which is the same as the *server.pem*, with the variation of being encrypted, and only being able to be decrypted with the key, thus becoming a .pem file. And one by the client, the *client.trust*, where the trust anchor is stored, which certifies whether the device you connect to is trusted, as well as the connection between the devices. This trust anchor is in charge of verifying digital signatures and certification routes. From both, we can verify that the connection is secure and start the TLS connection.

After that we find two functionalities, that of registration and authentication, both implemented in a single code, corresponding to the file *LibraryServer.java*, to all those functions in which the necessary methods are implemented to solve all the server

operations, and the *LibraryClient.java*, which in this case corresponds to the client. A file has been created on both *ClientEA.java* and *ServerEA.java*, where the different methods already defined above are used, applied in the code.

1.3.2.1. Enrollment

In both codes, both the client and the server determine the phase of the client device trying to connect, so if this results in enrollment, the client reads the file *puf.txt* in which the output of the PUF is generated and sent to the server. After that, its function in the registration has been completed so it simply indicates the end of the process.

On the other hand, the server determines if it finds a record of the device, if not, how the enrollment would be treated, creates a directory whose name will be the serial number of the client device, with this you can quickly access the information of the specific device later. This associates the device ID, in our case as we have mentioned the serial number, with said sequence of bits, and after that creates a JSON file with both fields and saves it in the directory corresponding to the serial number.

1.3.2.2. Autenticación

On the other hand, if the phase is authentication, the process is more complex. In this phase the client reads the input sequence provided by the server. This sequence has a length equal to the sequence of stable bits, varying as a function of the different stable bits of the different memories. After that, it first finds the corresponding PUF file based on the serial number, which, as we previously determined, coincides with the name of the directory where to find said file. Once both sequences have been extracted, an XOR of them is carried out, with which a single sequence is extracted, and, to finish the process, a *Hash* of the XOR. To do this, first the *MessageDigest*, that it is a cryptographic function of an output length determined by the algorithm to use, in our case “*SHA-256*”, which is made up of 256 bits, that is, 32 bytes. For a quicker comparison, said *hash* is then converted from binary to hex and then sent to the server.

The server for its part extracts the value of the PUF from the JSON file, for this it makes use of the serial number with which to find said PUF, which, as we observe below, allows

us to define the *path* in which the file is defined and after that extract with the method *get*, the value of the PUF field. Once this sequence is extracted, its length is analyzed, and by means of the function *generateBinRandom*, we obtain a random sequence with the same length as the PUF, which is sent at the same time to the client.

After that, we proceed as before to make an XOR of the random sequence and a hash of the XOR obtained, in the same way as the client method. After that, the server will check both *hashes*, and if the comparison is successful, it will indicate it to the client, being able then to derive the master secret from the common context, creating a JSON file with said sequence, and perform the OSCORE communication.

Finally, to carry a *log* of devices that have been authenticated, an entry will be added to the file *log* of the authentications, which will be stored in the server. This entry will include the time and date at which the device was authenticated, as well as its serial number, something that will be necessary for you to initiate an OSCORE communication with said device.

1.3.3. OSCORE communication between devices

For the implementation of the last section of the project we are working with the Californium libraries. For this, the libraries that were used are **californium-core** and **cf-oscore**. Initially, the server and client implemented in CoAP were tested (*HelloWorldClient.java* and *HelloWorldServer.java*). These codes were the basis of both *ClientOSCORE.java* and *ServerOSCORE.java*, both were also extended with the OSCORE protocol. Said protocol implements a security context based on a master secret combined with a master salt, which, from said context, allows the client and server contexts to be derived.

The functionality of this context has been modified to not implement the master secret in hardcoded in the code, rather that, it changes dynamically, being the output of the authentication hash, which due to its randomness (since it is implemented randomly due to the combination of stable bits with a random sequence bits), allows the system to have a unique security context for each session.

Once both codes have been started and the security context verified, the client and server will be connected. When this is done, an ongoing communication resource will be created called *ComsResource*, that will establish the *endpoints* from where both client and server will communicate. This resource will allow the server to respond to the different *Request* sent by the client, at which the server collects the value of the scanner which is configured to read the keyboard input and establishes that value as the payload of the message to be sent to the client, and after that performs a *Response* to the GET request made by the client.

On the client side, a *Request* will be performed. After that, having received the determined *Response* from the server to the specific *Request* sent before by the client itself, the client will print the result on the screen, being the result both the *Response Code* and the *Response Text*. Said communication between both will be prolonged until the server unless the server indicates the opposite, for this, the server will send the message “*Over*” closing its connection and also dictating to the client that must do the same.

1.3.4. Integration of all subsystems

After that, the configuration has finished, proceeding to an integration of all the mentioned parts in a whole system. For this, initially the idea of integrating all the functionalities in a single point was considered. But, although it could reduce some intermediate procedures, it added unnecessary complexity, since the result of such integration was that of a chaotic system. Therefore, it was decided to create three interconnected but independent parts, which brought more clarity to the project and allows each event to be analyzed independently.

To do this, **the PUF is initially generated**, the result of which will be stored in a .txt, and, as we have said before, in the system **enrollment** state, the file content will be sent to the server, which will create a new directory in its library of devices, and then will store the JSON file for that device in the directory. In the next session, which corresponds to the **authentication** session, said file will be located and used to verify the identity of the client device, to which, in the case of a positive comparison, the master secret that has

been derived from the output will be recorded, being the output of the authentication as the hash. of the PUF with a help information, which is a random binary sequence available to the client. And, with the server relying on the .log file to locate the device that has just been authenticated, the server will locate said secret which will be used to configure the communication master context of the **OSCORE server and client**, completing the connection, and with it, is completed the **IMPLEMENTATION. OF A KEY MANAGEMENT MECHANISM FOR OSCORE.**

1.4. System tests

All the different tests have been positive, and the project objective has been fulfilled successfully, although many issues have been encountered in the way, which is normal due to the complexity of the project proposed, as every system needed to work in order to the rest work correctly or for it to be able to be used in the project, as even if one worked without the rest, by itself it does not mean anything.

1.5. Conclusion of the project

This project combines multiple parts, that joint together result in a challenging but engaging project. A lot of research and investigation was made to make it come true as it is a project with many new concepts.

To end, this project has required a lot of effort and knowledge of the field, but is only the tip of the iceberg, as a lot of growth can be expected from it.