

Bootstrapping seguro para IoT

Grado en Ingeniería Informática

Convocatoria: Julio 2025

Trabajo Fin de Grado

Autor: Jose Antonio López Sola

Tutor: Antonio Fernando Skarmeta Gómez

Cotutor: Sara Nieves Matheu García

29 de Junio de 2025



Facultad
de Informática
UMU

Bootstrapping seguro para IoT

Autor

Jose Antonio López Sola

Tutor

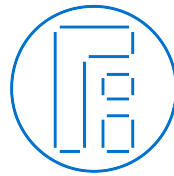
Antonio Fernando Skarmeta Gómez

Ingeniería de la Información y las Comunicaciones

Cotutor

Sara Nieves Matheu García

Ingeniería de la Información y las Comunicaciones



Facultad
de Informática

Grado en Ingeniería Informática



UNIVERSIDAD
DE MURCIA

Departamento
de Ingeniería de la Información
y las Comunicaciones

Murcia, 29 de Junio de 2025

Agradecimientos

Este trabajo supone el punto final a mi etapa como estudiante del grado de ingeniería informática y, con ello, la obtención de un título que me acredita como ingeniero informático. Haber cumplido este objetivo no habría sido posible sin el apoyo de las personas que me rodean. Es por ello por lo que me gustaría dedicarles unas palabras:

A mi familia, gracias por apoyarme y escucharme siempre que lo he necesitado. Vuestros consejos han sido aire fresco en mis peores momentos.

A mi pareja, gracias por entenderme cuando ni yo mismo lo hacía, siempre has estado ahí cuando lo he necesitado. Admiro la paciencia y comprensión que has tenido conmigo.

A mi abuelo, gracias por ser de los primeros en entender el esfuerzo que estaba haciendo. Tus ánimos están siempre presentes en mí.

A mi amigo Fran, nunca podré agradecerte lo suficiente el apoyo que me has dado.

A mi amigo Javier, no puedo estar más feliz de haber compartido contigo toda mi vida académica.

A mis amigos, gracias por ofrecerme una vía de escape y diversión que me ayudase a desconectar de los estudios cuando era necesario.

A los amigos que he hecho en la universidad, esta etapa ha sido más divertida gracias a vuestra compañía. El año que compartimos en Noruega jamás lo olvidaré.

Por último, agradecer a todas aquellas personas que han participado en el desarrollo de este trabajo. Gracias al profesor Rafael Marín por guiarme en la elección del TFG. Gracias a Sara Nieves, Gilson, José Manuel y Pedro, por proporcionarme ayuda y material que me facilitase abordar este proyecto. Gracias al profesor Antonio Fernando Skarmeta por dirigir este trabajo, estoy muy agradecido de que entendiese mi situación personal de este año y me proporcionase un trabajo que se adaptase a estas circunstancias.

Declaración firmada sobre originalidad del trabajo

D./Dña. **Jose Antonio López Sola**, con DNI **49173017Y**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado “**Bootstrapping seguro para IoT**”.

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declaro que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial.

Murcia, a 29 de Junio de 2025

A handwritten signature in black ink, consisting of a stylized 'J' followed by a series of loops and a final 'S' at the end.

Fdo.: Jose Antonio López Sola
Autor del TF

Resumen

En la actualidad, la presencia de dispositivos Internet of Things (IoT) en nuestra sociedad está en aumento, pudiendo encontrarlos prácticamente en cualquier entorno.

Su uso, aunque aporta grandes beneficios, también trae consigo una serie de retos a los que se debe hacer frente. Estos se deben principalmente a dos características de los dispositivos IoT: la posibilidad de manejar información sensible y su capacidad para interactuar con el entorno en el que están desplegados. Por ello, se hace necesario prestar especial interés a la securización de los dispositivos, llevando a la búsqueda de un modelo de gestión que permita asegurar su correcto funcionamiento.

El objetivo del trabajo es estudiar cómo solucionar este problema. Para ello, se ha estudiado el modelo de gestión de la seguridad que está siendo planteado en la Universidad de Murcia, con el proyecto aCtive sEcurity foR connecTed devIces liFecYcles (CERTIFY). Dicho modelo busca proteger a los dispositivos durante todo su ciclo de vida, siendo una de las tareas de mayor importancia el proceso de arranque, al que se conoce como *bootstrapping*. Este proceso es el aspecto central de este trabajo. Su objetivo es permitir a los dispositivos obtener la información necesaria para poder trabajar de forma segura en su entorno. Para conseguir esto, el *bootstrapping* incluye, entre otros aspectos, realizar tareas de autenticación y autorización, que garanticen que los dispositivos desplegados en el entorno son los correctos y que las acciones que se les permite realizar están limitadas bajo ciertos criterios. Además, el *bootstrapping* busca dejar preparados a los dispositivos para que puedan recibir actualizaciones que mantengan el nivel de seguridad con el paso del tiempo. El cómo realizar estas actualizaciones también es estudiado en el modelo mencionado, y en este trabajo se aborda como aspecto secundario.

Una vez estudiada la propuesta de la Universidad de Murcia, el trabajo propone una Proof of Concept (PoC) con el objetivo de poner a prueba ciertos aspectos del modelo. En concreto, se busca realizar el proceso de *bootstrapping* de un dispositivo IoT y tras este, aplicar políticas de comportamiento que limiten/controlen las acciones del dispositivo. Dichas políticas son obtenidas de ficheros que siguen el estándar Manufacturer Usage Description (MUD). Para la realización de la PoC se utiliza como dispositivo IoT una Raspberry Pi 3 Model B.

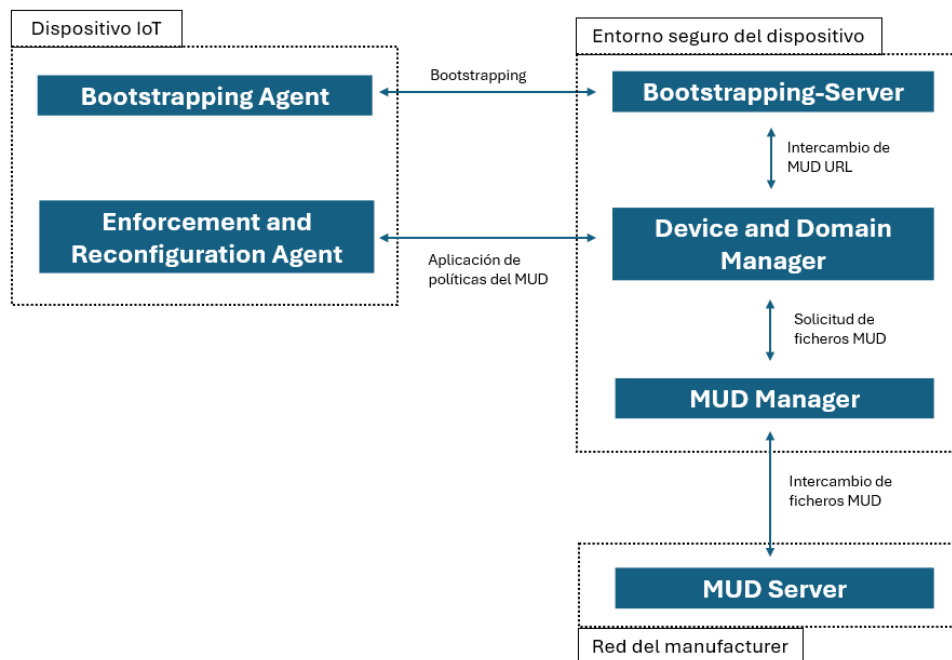


Figura 1: Escenario IoT propuesto

La figura 1 muestra el escenario IoT que se plantea en este trabajo, así como los diferentes componentes necesarios para su despliegue. En particular, las entidades *Bootstrapping Agent* y *Bootstrapping-Server* son las encargadas de permitir el proceso de *bootstrapping* del dispositivo IoT, incluyendo su autenticación. Por su parte, las entidades Enforcement and Reconfiguration Agent (ERA), Device and Domain Manager (DDM), MUD Manager y MUD Server se encargan de aplicar políticas de comportamiento, definidas en un fichero MUD, sobre el dispositivo.

El resultado de la PoC ofrece un escenario que permite gestionar dispositivos IoT, securizando su configuración inicial mediante el proceso de *bootstrapping*, y reforzándola con la aplicación de políticas de seguridad una vez que el dispositivo ha iniciado sus operaciones. La solución planteada también proporciona beneficios al proyecto en el que se basa, CERTIFY; al haber hecho uso de componentes desarrollados en dicho proyecto, se ha podido detectar ciertos errores en las implementaciones y subsanarlos.

Además, el trabajo realizado incluye una evaluación de los tiempos de ejecución de las principales actividades que se realizan en la PoC desarrollada. Se estudia tanto el tiempo empleado para completar el proceso de *bootstrapping* como el tiempo necesario para aplicar políticas de comportamiento. El principal resultado obtenido es que el *bootstrapping* es un proceso mucho más pesado, requiriendo aproximadamente seis veces el tiempo que necesita la aplicación de políticas de seguridad en el dispositivo.

Aunque la PoC planteada no aborda la totalidad del proyecto CERTIFY, teniendo así ciertas limitaciones, supone un buen punto de partida para comprender el modelo de gestión del ciclo de vida de dispositivos IoT que propone dicho proyecto. En particular, permite tener una visión concreta de los conceptos con los que se debe trabajar y proporciona una base sobre la que poder avanzar en futuras mejoras.

Extended Abstract

Nowadays, the number of Internet of Things (IoT) devices is constantly increasing. In fact, we can find these devices almost everywhere. Although these artefacts are very useful, their own nature is related to some challenges that we should confront. For example, some IoT devices can interact with their environment, while others are managing sensitive information or even doing both things at the same time. This naturally highlights the importance of security in IoT systems.

To address this concern, we have studied a model for managing IoT devices that is currently being developed at the University of Murcia within the European project aCtive sEcurity foR connecTeD devIces liFeCYcles (CERTIFY). This model proposes an approach to managing security in IoT devices; its goal is to manage their complete lifecycle, ensuring that they are working in a secure way. As part of this process, one of the main tasks is the secure boot or *bootstrapping* of the device, which is indeed the main focus of this work. Its purpose is to allow devices to obtain the necessary data to operate securely in their environment. To do this, *bootstrapping* includes, among other aspects, performing authentication and authorization tasks to ensure that the deployed devices are allowed to be in such environment, and their actions are restricted to a specific criterion. Moreover, *bootstrapping* prepares the devices to be able to receive updates that maintain their security level during their operational period. The procedure for carrying out these updates is also studied within the mentioned model, and in this work, it is addressed as a secondary aspect.

Once the proposal from the University of Murcia and the associated technologies that it requires have been studied, the work presents a Proof of Concept (PoC) with the purpose of testing certain aspects of the model. In particular, the PoC performs the *bootstrapping* of an IoT device and tries to apply some behavioural policies to restrict/-control the allowed actions of the device. These policies are defined in files that follow the Manufacturer Usage Description (MUD) standard. For the testing, a Raspberry Pi 3 Model B has been used as the IoT device.

To carry out this work, several components previously developed within the framework of the CERTIFY project were reused. This approach allowed us to implement the scenario more efficiently and focus on validating the key functionalities defined by the model. Developing a complete IoT environment from scratch would have been a highly complex and time-consuming task, so the reuse of existing elements was both practical

and aligned with the objectives of the work. Moreover, reusing trusted components contributed to ensuring greater reliability and consistency in the results.

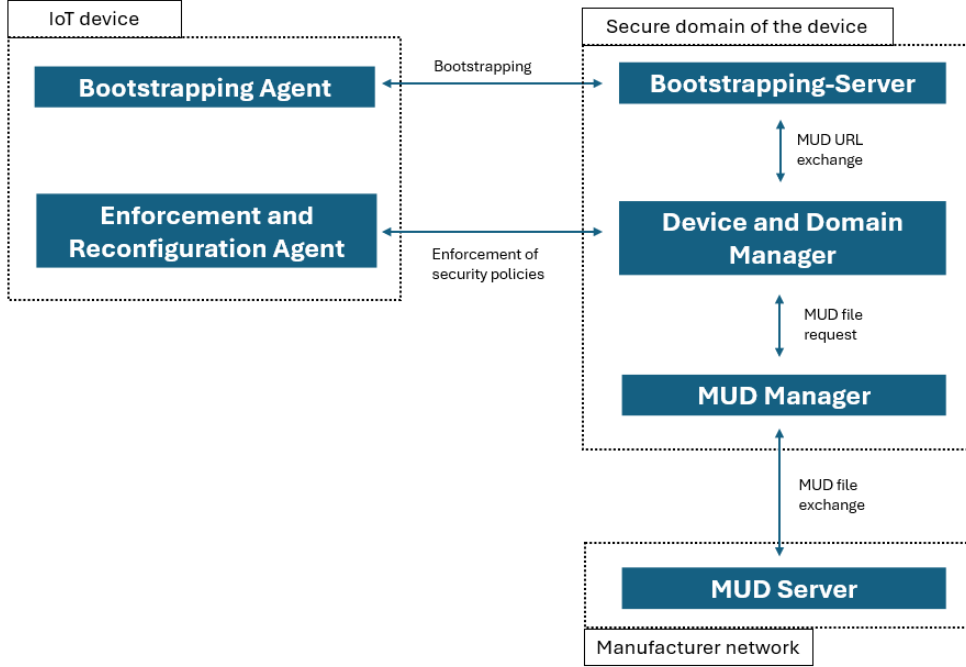


Figure 2: Proposed IoT scenario

Figure 2 shows the IoT scenario proposed in this work. Its deployment and testing have been divided into several phases, which are described below.

Phase 1. Study of the CoAP-EAP *bootstrapping* process.

The first step is to understand the *bootstrapping* model that we will use in our IoT scenario, since this is the key for defining the *Bootstrapping Agent* and *Bootstrapping-Server* presented in figure 2.

The selected *bootstrapping* model was proposed at the University of Murcia and has become a Request for Comments (RFC) of the Internet Engineering Task Force (IETF). It is important to mention that this model focuses on the *bootstrapping* phase of an IoT device, i.e., the first stage of the lifecycle of the device. It determines what should happen the first time the device boots and wants to operate in the environment in which it has been deployed. This model also considers, as a secondary aspect, the operation and maintenance phase of the device, as it provides key material that supports the next stages of the lifecycle.

Although there are many approaches for IoT *bootstrapping*, the authors argue that solutions prior to their model present several limitations, such as the lack of use of lightweight protocols, a critical aspect due to the diversity in IoT devices capabilities; the absence of support for identity federation, essential in scenarios where different organizations must cooperate and devices are registered and deployed in different environments; and the lack of a flexible authentication process. Therefore, the authors propose a new alternative for *bootstrapping*, named CoAP-EAP.

The proposed solution makes use of Extensible Authentication Protocol (EAP), Constrained Application Protocol (CoAP) and Authentication, Authorization and Accounting (AAA) infrastructure. CoAP provides the required lightweight protocol for the transport of authentication information, since it is considered one of the best protocols for communications in IoT devices, and EAP enables flexible authentication as a result of the different methods that can be used. In particular, CoAP-EAP supports various authentication methods, including EAP-PSK (Pre-Shared Key), which is especially well-suited for IoT scenarios. In this method, a PSK is used to authenticate the device during the *bootstrapping* phase, offering a lightweight and secure mechanism that aligns with the resource constraints typical of IoT environments.

The model CoAP-EAP places a centralized entity within the IoT environment. Thus, when an IoT device joins that environment for the first time, the centralized entity, the AAA server, and the device establish a communication to perform authentication and derive cryptographic material, which allows the device to perform its tasks.

Phase 2. Study of an implementation of the COAP-EAP *bootstrapping*.

Once we understand the CoAP-EAP model for *bootstrapping*, we need an implementation of the AAA server. This activity was not done by us, since the server was already developed. Therefore, our objective was to study the available server implementation developed in the C programming language, identify errors and carry out testing.

It is important to note that if we want to test the server, we also need an implementation of a client for the AAA server. This code was also available, since it was developed in parallel with the AAA server. Therefore, it was only necessary to install it on the desired IoT device. In order to test the implementation of CoAP-EAP, we used as IoT device a Raspberry Pi 4 Model B.

We can test the server by running the client over the Raspberry Pi. This establishes a connection with the AAA server, which must be running on a general-purpose computer.

To ensure that the AAA server and client were correctly implemented, it was not enough to simply execute both entities. It was necessary to read and analyze the whole code to guarantee that it was following the CoAP-EAP model. By doing this, we ensured that we had correct implementations ready to deploy the *Bootstrapping Agent* and *Bootstrapping-Server* described in figure 2.

Phase 3. Configuration of a secure IoT device.

As we mentioned before, the IoT device has to manage sensitive information as cryptographic material. This makes the device vulnerable, for example, to physical manipulation once the device is deployed. For that reason, we should make an effort to strengthen the security of our device.

To perform this task, we can make use of a Trusted Execution Environment (TEE). This basically creates two different worlds in our device: “the normal world”, the one that we are used to, and “the trusted world”, which is protected and allows us to manage sensitive information, since it is isolated from the “normal world”. By using this technology, we can perform the critical activities of the device with an extra level of security.

In particular, we used the TEE provided by OP-TEE. However, this TEE had some limitations regarding the IoT devices that could be used. Until this moment, we had been working with a Raspberry Pi 4 Model B and the generic operating system recommended by the manufacturer. However, we had to install OP-TEE and, at the time of writing this work, OP-TEE was not supported by a Raspberry Pi 4 Model B, so we had to change to a Raspberry Pi 3 Model B, since this was the highest model of Raspberry that supported this technology.

Regarding the functionality related to the *bootstrapping* of the IoT device, it is important to note that the required code must be installed while deploying OP-TEE. This code was not developed in the context of this work, so in this phase it was only necessary to deploy OP-TEE on a real device, add the mentioned code and test it. Performing this task strengthened the implementation of the *Bootstrapping Agent* described in figure 2.

Phase 4. Integration of the IoT device with the AAA server.

Once we had the AAA server deployed in a generic computer and the IoT device configured to execute the *bootstrapping* in a secure environment, we had to test that the process works properly. However, at this point, we found a problem, OP-TEE had several problems to integrate CoAP in the TEE, so the developer of the code for OP-TEE changed CoAP-EAP to use EAP directly over Transmission Control Protocol (TCP). This was a big problem, since the AAA server was designed to use CoAP, and therefore,

User Datagram Protocol (UDP). Thus, we could not establish communication between these entities. To fix the problem, since working with OP-TEE is considerably more complex than modifying the AAA server, we chose to move away from the strict CoAP-EAP model studied in phase 1, mainly for time-related reasons. This is, obviously, a downgrade, since TCP is not supported in many IoT devices. Hence, this limitation should be addressed in a future version to provide a more generic IoT scenario.

After making some adjustments to the AAA server, both entities were able to communicate and complete the *bootstrapping* process. Therefore, the deployment and testing of the *Bootstrapping Agent* and the *Bootstrapping-Server* described in figure 2 were successfully completed.

Phase 5. Update the configuration of the IoT device to be able to receive commands from external entities.

The next step after completing the *bootstrapping* phase, is to be able to update the device during its whole lifecycle. This includes being able to modify its behaviour at any time, if necessary.

To achieve this, it was necessary to extend the functionality of the IoT device by adding a process capable of receiving commands from a trusted entity. These commands may involve updates to the device's behaviour or information, such as cryptographic material. This process is performed by an entity named Enforcement and Reconfiguration Agent (ERA), as defined within the CERTIFY project and shown in figure 2.

The necessary code to implement the ERA, as happened in phase 3, was already available, so in this stage the task included adding the code to OP-TEE and testing it on a real device.

The main challenge of this phase was that the ERA was implemented in Python. There was no available information on how to add Python to a Raspberry Pi 3 Model B running OP-TEE. Therefore, we had to study and analyze how the OP-TEE installation process builds the entire system of the Raspberry in order to identify a way to include Python.

Phase 6. Establishment of the required infrastructure for retrieving information associated to an IoT device and process it.

Finally, a crucial aspect is determining what information must be received from an IoT device in order to evaluate if its behaviour needs to be modified. Besides, it is important to identify who will provide the changes that should be applied to the device.

The key aspect to answer these questions is what it is called MUD. Each IoT device has associated a MUD file defined by its manufacturer, which describes the expected behaviour in terms of security policies. This file can be obtained using a Uniform Resource Locator (URL), referred to as the MUD URL. The IoT device should send this URL to the AAA server during the *bootstrapping*; then, the MUD URL should be forwarded to what we call, following the CERTIFY project terminology, the Device and Domain Manager (DDM). This new entity has the task of managing a certain number of IoT devices with the purpose of ensuring a secure operation within the deployment domain. Regarding MUDs, the DDM is in charge of requesting MUD files, process their contents and enforcing the security policies that they contain. Moreover, the content of MUD files could change throughout time, for instance, if a new vulnerability is discovered in certain IoT devices, the security policies can be updated to mitigate the issue. For that reason, periodically, the DDM must check if there are changes in the MUD files of the IoT devices that are under its supervision. If the information of the MUD file of one device has changed, it indicates that the manufacturer has established new policies for the IoT device, and therefore, the behaviour of the device must be updated. These periodic updates, although specified in CERTIFY project, have not been addressed in this work.

Hence, in this work, the task of the DDM consists of retrieving the MUD file associated with an IoT device, processing the information it contains and translating it into commands to be sent to the corresponding ERA in order to update the device's behaviour. This process is summarized on the right side of figure 2.

In addition, since the information included in a MUD is accessible via a URL, we also need a server able to provide a MUD file when an entity asks for it; this is what we call MUD Server in figure 2. Thus, in this phase, we had to develop a MUD Server and modify an existing DDM that is part of the CERTIFY project.

Regarding the DDM, we should know that this entity must also take part in the *bootstrapping* phase. However, the deployed AAA server and ERA had some limitations in allowing this. For that reason, we had to restrict the functionality of the DDM to the *post-bootstrapping* phase.

The resulting deployment provides a scenario capable of managing the lifecycle of an IoT device, in which security is one of the main concerns. Moreover, the proposed PoC brings benefits to the project it is based on, CERTIFY, since the reuse of components developed for that project has allowed us to identify and resolve previously unknown issues related to their implementations.

Moreover, this work includes a performance evaluation aimed at measuring the execution time of the two main processes of this PoC: the *bootstrapping* phase and the enforcement of behavioural policies. The results showed that the *bootstrapping* phase is significantly more time-consuming, requiring approximately six times longer than the policy enforcement process.

Although this PoC is not a perfect solution, as it provides limited management capabilities for IoT devices due to certain limitations that should be addressed in future work, it provides a solid starting point for understanding the key concepts involved in managing the lifecycle of IoT devices. In addition, the work provides a clear direction for future improvements. In fact, this PoC connects directly with the CERTIFY project, currently being developed at the University of Murcia, which will continue building on this work.

Índice general

1. Introducción	1
2. Estado del arte	5
2.1. <i>Bootstrapping</i> de dispositivos IoT	5
2.1.1. Modelo de <i>bootstrapping</i> CoAP-EAP	6
2.1.1.1. Constrained Application Protocol (CoAP)	8
2.1.1.2. Extensible Authentication Protocol (EAP)	9
2.2. Trusted Execution Environment (TEE)	10
2.3. Aplicación de políticas de comportamiento en dispositivos IoT	12
2.3.1. Manufacturer Usage Description (MUD)	13
2.3.1.1. Definición de un fichero MUD	13
2.3.1.2. Difusión de un fichero MUD	13
2.3.1.3. Estructura de un fichero MUD	14
2.3.1.4. Limitaciones de los ficheros MUD	15
3. Análisis de objetivos y metodología	17
4. Diseño y resolución del trabajo realizado	19
4.1. Proceso de <i>bootstrapping</i> CoAP-EAP	20
4.1.1. Prueba de una implementación del servidor de <i>bootstrapping</i>	20
4.1.1.1. Configuración de una Raspberry Pi 4 Model B	22
4.1.2. Actualización del servidor de <i>bootstrapping</i> proporcionado	22
4.2. Securización de un dispositivo IoT	24
4.2.1. Prueba del proceso de <i>bootstrapping</i> con un cliente seguro	29
4.3. Infraestructura para aplicar políticas de comportamiento	30
4.3.1. Despliegue de un ERA en un dispositivo IoT	30
4.3.1.1. Prueba del ERA	32
4.3.2. Despliegue de infraestructura para trabajar con ficheros MUD	32
4.3.2.1. Actualización del <i>Bootstrapping-Server</i>	33
4.3.2.2. Despliegue de un servidor que proporcione ficheros MUD	34
4.3.2.3. Despliegue de un servidor que solicite ficheros MUD	35
4.3.2.4. Definición de un fichero MUD	35
4.3.2.5. Coordinación de la aplicación de políticas	37
4.4. Prueba de la PoC desarrollada	40
5. Evaluación de la PoC desarrollada	53

6. Conclusiones y vías futuras	57
Bibliografía	60
A. Anexo I. Fichero MUD generado	64
B. Anexo II. Traducción del fichero MUD a MSPL	67
C. Anexo III. Base de datos del <i>Device and Domain Manager</i>	69
D. Anexo IV. Configurar SSH en una Raspberry Pi 3 Model B	71

Índice de figuras

1.	Escenario IoT propuesto	vii
2.	Proposed IoT scenario	x
2.1.	Interacción para el proceso de <i>bootstrapping</i>	8
2.2.	Estructura de un TEE	11
2.3.	Proceso de obtención de un fichero MUD	14
4.1.	Escenario IoT a desplegar	19
4.2.	PoC: Arranque de las entidades distintas al dispositivo IoT	41
4.3.	PoC: Inicio del <i>bootstrapping</i>	42
4.4.	PoC: Consecuencias del <i>bootstrapping</i>	43
4.5.	PoC: Consecuencias del <i>bootstrapping</i> (continuación)	44
4.6.	PoC: Consecuencias del <i>bootstrapping</i> en el controlador central	46
4.7.	PoC: Consecuencias del <i>bootstrapping</i> en el AAA server	47
4.8.	PoC: Consecuencias del <i>bootstrapping</i> en el <i>Post-Server</i>	48
4.9.	PoC: Consecuencias del <i>bootstrapping</i> en el DDM	49
4.10.	PoC: Consecuencias del <i>bootstrapping</i> en el MUD Server	49
4.11.	PoC: Consecuencias del <i>bootstrapping</i> en el MUD Manager	50
4.12.	PoC: DDM tras aplicar el cambio de algoritmo de firma	51
4.13.	PoC: Raspberry tras aplicar el cambio de algoritmo de firma	52
5.1.	Flujo del escenario IoT desplegado	54
5.2.	Tiempos de ejecución del <i>bootstrapping</i> y del <i>enforcement</i> en la PoC	54
5.3.	Desglose de los tiempos de ejecución del <i>enforcement</i> en la PoC	55
C.1.	PoC: Base de datos del DDM tras completar el <i>bootstrapping</i>	70

Índice de tablas

2.1. Limitaciones de un fichero MUD	16
4.1. Librerías necesarias para ejecutar el servidor de <i>bootstrapping</i>	21

1. Introducción

El concepto de Internet of Things (IoT) tiene cada vez más presencia en nuestra sociedad. Sin embargo, muchas veces este término es utilizado sin tener realmente claro a qué nos referimos por IoT. Por ello, dado que será el elemento central de este trabajo, es conveniente dar una introducción a este concepto y motivar por qué se ha decidido estudiarlo.

Es importante tener claro que no existe una definición universal para IoT [1, 2]. Esto no quiere decir que no haya una forma de describir este concepto, simplemente cuando se trata de explicar qué es el IoT se recurre a las características que se considera que estos tipos de dispositivos poseen. En este trabajo, para dar esta definición usaremos las propiedades que remarca García Carrillo en [1]:

1. **Interacción con el entorno.** El IoT permite mejorar el proceso de obtención de información mediante el uso de máquinas. Estas son capaces de realizar ciertas tareas de un modo más preciso y eficiente que los seres humanos.
2. **Conectividad.** El IoT permite conectar a Internet dispositivos con capacidades muy diversas, desde equipos complejos hasta sensores con recursos muy limitados.

Por tanto, el concepto de IoT se puede resumir en que está compuesto por dispositivos de capacidades muy variadas que son capaces de interactuar con el entorno en el que estén desplegados y de conectarse a Internet.

Esto provoca que el abanico de mejoras tecnológicas a través del uso de IoT se amplíe enormemente, teniendo como consecuencia un incremento muy notorio del número de dispositivos IoT desplegados en todo el mundo. Esta tendencia queda reflejada en [3]: en 2023 se reportó la existencia de 16.6 billones de dispositivos, y en 2024 se esperaba un crecimiento hasta los 18.8 billones. Además, se pronosticó el número de dispositivos IoT desplegados entre 2024 y 2030, estimándose que, para finales del presente año, 2025, se tengan 21.5 billones. Estos datos permiten reflejar la gran relevancia del IoT en la sociedad actual.

Su impacto se extiende a sectores muy diversos, como la agricultura, la industria, la medicina o el comercio, transformando el día a día de las personas mediante soluciones como las “*Smart Homes*”, “*Smart Cities*” o relojes inteligentes que permiten monitorizar la salud de las personas.

Sin embargo, el IoT, además de sus grandes beneficios, trae consigo una serie de retos a los que se debe hacer frente. En particular, destacan los relacionados con la seguridad de los dispositivos, pues su capacidad para acceder a Internet tiene como consecuencia directa que estén expuestos a amenazas que pueden comprometer tanto su funcionamiento¹ como los datos² que gestionan. Esta característica junto al hecho de los dispositivos IoT están masivamente desplegados³ por todo el mundo, provoca que la seguridad de los mismos sea un tema que debe ser tratado con urgencia.

La protección que se debe proporcionar a los dispositivos IoT no debe abarcar únicamente los riesgos que se deban al acceso a Internet. Estos dispositivos son desplegados con la intención de que trabajen de forma autónoma, lo que puede provocar que queden expuestos físicamente y, por tanto, sean susceptibles de sufrir ataques físicos. Entre estos, destaca el conocido como *tampering*, pues suele tener el mismo objetivo que un ataque a través de la red: alterar el funcionamiento del dispositivo o robar información valiosa que maneje.

Por tanto, encontrar una solución que garantice la seguridad en cualquier tipo de dispositivo IoT no es una tarea sencilla, pues la variedad entre los dispositivos que se pueden desplegar hace que las medidas de seguridad se vean limitadas por las capacidades de los dispositivos más restringidos. Además, los dispositivos IoT se caracterizan por tener un bajo coste, por lo que es necesario dotar de seguridad a estos dispositivos repercutiendo lo menos posible en su precio.

También debe considerarse que los dispositivos IoT cuentan con ciclos de vida muy largos. Por ejemplo, un sensor agrícola diseñado para controlar la humedad del suelo puede mantenerse en funcionamiento durante años. Esto provoca que la probabilidad de que con el paso del tiempo puedan ir apareciendo nuevas amenazas que dejen expuestos a los dispositivos, se incremente. Es necesario diseñar estos dispositivos para que puedan ser actualizados de un modo seguro y eficiente, evitando que queden desprotegidos. De este modo, los dispositivos IoT deben poseer un sistema de seguridad que no solo los proteja en el inicio de su ciclo de vida, sino que pueda ser actualizado mientras el dispositivo se mantenga activo, asegurando así su protección durante toda su vida útil.

¹El hecho de que un atacante pueda alterar el comportamiento de un dispositivo IoT puede tener consecuencias en todo el sistema en el que este opera.

²Proteger la información manejada por los dispositivos IoT es importante tanto por intereses personales como por la regulación actual existente, la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPD) [4].

³Un fallo de seguridad detectado en un dispositivo puede generar problemas en un gran número.

La securización de los dispositivos IoT expuesta evidencia la necesidad de establecer mecanismos de protección robustos y adaptativos desde el momento del despliegue y durante toda la vida útil de estos dispositivos. Actualmente, la Universidad de Murcia, bajo el proyecto aCtive sEcurity foR connecTed devIces liFecYcles (CERTIFY) [5], está desarrollando una propuesta que permita realizar una gestión segura de los dispositivos IoT, asegurando su correcto funcionamiento.

En este trabajo se presenta una Proof of Concept (PoC) que permita poner a prueba ciertos aspectos con los que se trabaja en CERTIFY. En concreto, la PoC se centra en el proceso de *bootstrapping* de un dispositivo IoT. Este proceso supone la gestión de la primera fase del ciclo de vida de un dispositivo, dictaminando qué debe ocurrir cuando un dispositivo es desplegado en un determinado entorno y es arrancado por primera vez. El objetivo del *bootstrapping* es realizar tareas de autenticación y autorización que aseguren que el dispositivo desplegado puede operar en dicho entorno y tiene limitadas las acciones que puede realizar sobre este. Además, el *bootstrapping* busca preparar al dispositivo para que pueda continuar con las siguientes etapas de su ciclo de vida: operación y mantenimiento. Para ello, esta fase debe concluir con la obtención, por parte del dispositivo, de material criptográfico que le permita trabajar de forma segura.

Dado que los ciclos de vida de los dispositivos IoT son muy largos, es necesario poder realizar actualizaciones sobre estos, ya sean sobre sus comportamientos o sobre la información que poseen. La PoC realizada también incluye una exploración inicial del uso de políticas de comportamiento mediante el estándar Manufacturer Usage Description (MUD) [6], como mecanismo para actualizar o adaptar la configuración de los dispositivos una vez desplegados. En concreto, estudia cómo definir y aplicar políticas de seguridad en los dispositivos IoT que han completado el proceso de *bootstrapping*.

Para el desarrollo de la PoC se ha hecho uso de distintas tecnologías. El proceso de *bootstrapping* sigue el modelo CoAP-EAP propuesto en [1], consiguiendo un proceso de autenticación ligero para el dispositivo IoT. Las claves criptográficas y las operaciones críticas del mismo son protegidas mediante el uso de un Trusted Execution Environment (TEE). Además, se implementa un mecanismo para desplegar políticas en el dispositivo IoT haciendo uso del estándar MUD.

En resumen, este trabajo propone un escenario de gestión básica del ciclo de vida de dispositivos IoT, basado en el proyecto CERTIFY y con foco en el proceso de *bootstrapping*. A pesar de sus limitaciones, representa un primer paso hacia la integración de los componentes clave del proyecto y proporciona una base sobre la cual desarrollar futuras mejoras.

Finalmente, presentamos la organización que sigue este documento para exponer el trabajo realizado. La sección **2** ofrece una visión teórica de las diferentes tecnologías, elementos y protocolos que han sido utilizados en este trabajo. Tras esto, en la sección **3** se presenta de forma estructurada el objetivo de la PoC y cómo se ha trabajado para desplegarla. En la sección **4** se expone con detalle cada uno de los pasos seguidos para crear el escenario IoT que se propone en este trabajo, así como los problemas encontrados durante su desarrollo y una prueba de la PoC desplegada. La sección **5** presenta una evaluación de los tiempos de ejecución de las principales actividades que se realizan en el escenario desplegado. Por último, en la sección **6** se discuten los resultados obtenidos con la PoC, exponiendo el trabajo futuro que se puede desarrollar a partir de este escenario IoT.

2. Estado del arte

A lo largo de esta sección se expondrán los conceptos teóricos de las herramientas y tecnologías utilizadas en este trabajo. El objetivo de hacer esto es tratar de proporcionar un entendimiento general de los diferentes elementos utilizados y tratar de motivar su uso.

En concreto, comenzaremos dando una introducción al proceso de *bootstrapping*. Una vez conocido qué es este proceso, presentaremos el modelo elegido, CoAP-EAP, definiendo así su funcionamiento y los protocolos sobre los que se basa. Tras esto, trataremos el concepto de los TEE, motivando así por qué estos son de gran utilidad para proteger dispositivos IoT. Finalmente, hablaremos del estándar MUD, concepto clave para poder controlar el comportamiento de los dispositivos IoT.

2.1. *Bootstrapping* de dispositivos IoT

Tal y como menciona García Carrillo en [1], el ciclo de vida de un dispositivo IoT se divide en 3 fases: *bootstrapping*, operación y mantenimiento. La primera de estas etapas supone el arranque y configuración inicial del dispositivo para permitir que se una de forma segura a la red IoT donde el dispositivo pretende trabajar. Tras esta, comienza la fase de operación, donde el dispositivo realiza las tareas para las que ha sido diseñado. Cada cierto tiempo, el dispositivo requerirá entrar en la fase de mantenimiento para recibir actualizaciones que mantengan su correcto funcionamiento.

En esta sección nos centramos en la primera de estas fases, el *bootstrapping*. Esta etapa tiene lugar cuando, tras su fabricación, el dispositivo es desplegado en el entorno en el que trabajará. Como ya se ha comentado, un dispositivo no debe poder empezar a trabajar conforme es desplegado, pues esto supondría una serie de riesgos tanto para el dispositivo como para el propio entorno donde está operando. Por este motivo, aparece el concepto de *bootstrapping*. Este proceso busca llevar a cabo dos actividades para considerar al dispositivo como seguro y permitir que comience con sus operaciones [1, 7]. Estas son:

- **Autenticación y autorización del dispositivo.** Se debe comprobar quién es el dispositivo y qué permisos tiene para realizar acciones dentro del entorno.

- **Obtención, por parte del dispositivo, de la información necesaria para poder llevar a cabo sus funciones.** Esto incluye obtener una dirección IP para tener acceso a Internet y obtener parámetros de seguridad, como material criptográfico, que le permitan realizar sus actividades de forma segura.

De este modo, cuando el *bootstrapping* se completa, puede avanzar a la siguiente fase de su ciclo de vida: operación. Además, periódicamente requerirá entrar en la fase de mantenimiento para actualizar su comportamiento y/o información almacenada.

En [1] se expone un gran número de propuestas para llevar a cabo el proceso de *bootstrapping* en dispositivos IoT. Sin embargo, se concluye que la mayoría de las soluciones existentes hasta ese momento carecen de características deseables en el proceso de *bootstrapping*. Por ejemplo, hay modelos que no hacen uso de protocolos ligeros que se adapten a las capacidades de cualquier tipo de dispositivo IoT, limitando así su ámbito de aplicación. Otras propuestas no proporcionan un sistema de autenticación flexible, aspecto que puede ser considerado esencial, dado que los dispositivos IoT pueden ser desplegados en entornos muy variados, donde las diferentes organizaciones impongan requisitos distintos para completar el proceso de autenticación. Además, hay soluciones que no ofrecen soporte para el uso de una federación de identidad, lo que dificulta la colaboración entre organizaciones. El trabajo mencionado, resume, junto a los ejemplos expuestos, otras propiedades que son deseables en un proceso de *bootstrapping* y comprueba que no existe una alternativa que reúna al mismo tiempo todas las características. De este modo, el trabajo de este autor se centra en desarrollar un nuevo proceso de *bootstrapping* que cubra esas carencias. Este modelo fue desarrollado dentro de la Universidad de Murcia, llamado CoAP-EAP, y ha sido el elegido para estudiar en este trabajo.

Es interesante destacar, que el modelo CoAP-EAP es actualmente un Request for Comments (RFC) del Internet Engineering Task Force (IETF), pudiendo encontrar su descripción en [8]. Este hecho refuerza la decisión de elegir CoAP-EAP como modelo para realizar el proceso de *bootstrapping*.

2.1.1. Modelo de *bootstrapping* CoAP-EAP

García Carrillo en [1] propone el modelo de *bootstrapping* CoAP-EAP. Este modelo se basa en tres conceptos: infraestructura Authentication, Authorization and Accounting (AAA) [9, 10], protocolo Extensible Authentication Protocol (EAP) [11] y protocolo Constrained Application Protocol (CoAP) [12].

Por un lado, la infraestructura AAA es utilizada para centralizar el proceso de autenticación y autorización de dispositivos IoT en una entidad dedicada a dichas tareas, conocida como Identity Provider (IdP). De este modo, la infraestructura AAA facilita

la colaboración entre organizaciones a través del uso de una Federación de Identidad. Esto es de gran utilidad cuando se despliegan dispositivos de una determinada organización en entornos que pertenecen a otra. Mediante el uso de la Federación de Identidad y la infraestructura AAA se facilita la autenticación de un determinado dispositivo. Cabe mencionar que la parte de *Accounting* de la infraestructura AAA corresponde a un seguimiento de las acciones realizadas por un dispositivo. Esto, aunque no es una tarea estrictamente necesaria, siempre es de utilidad para monitorizar el estado de una red.

Por otro lado, EAP es el protocolo elegido para realizar la autenticación de los dispositivos. El motivo de usar EAP es que proporciona una autenticación flexible gracias a la variedad de métodos que soporta, permitiendo así adaptar la autenticación a las necesidades que se tenga. Además, EAP permite finalizar el proceso de autenticación con la derivación de claves que conformen el material criptográfico del dispositivo y le permitan comenzar a operar en el entorno en que ha sido desplegado.

Finalmente, la motivación del uso de CoAP se debe a que los dispositivos IoT no son capaces de soportar protocolos con altas exigencias de cómputo o almacenamiento, por lo que se requiere hacer uso de protocolos ligeros que tengan en cuenta estas limitaciones. CoAP se ha convertido en el protocolo estándar para el intercambio de información entre dispositivos de capacidades limitadas, incluyendo así los dispositivos IoT. De este modo, CoAP es el elegido para transportar los mensajes EAP necesarios para el proceso de *bootstrapping*.

Una vez conocidas las bases sobre las que se apoya el proceso de *bootstrapping* CoAP-EAP, este se puede resumir del siguiente modo:

1. Un dispositivo IoT es desplegado en un determinado entorno. Por tanto, debe comenzar el proceso de *bootstrapping* para ser considerado un elemento seguro dentro de esa red IoT.
 2. En dicho entorno existe una entidad encargada de controlar el acceso a la red IoT. Por tanto, es la encargada de interactuar con el dispositivo IoT para llevar a cabo el *bootstrapping*. Entre sus tareas se encuentra autenticar y autorizar al dispositivo, por lo que hace uso de la infraestructura AAA (ya sea por tenerla implementada o porque se comunica con otra entidad que posee dicha infraestructura).
 3. El intercambio de información entre el dispositivo y la entidad dedicada a la autenticación se hace por medio de mensajes EAP transportados sobre CoAP.
 4. Como consecuencia del proceso de *bootstrapping* el dispositivo IoT adquiere material criptográfico para poder realizar sus funciones dentro de la red IoT.
-

Los mensajes específicos que se intercambian dentro de este protocolo están definidos en [8]. Por este motivo, para no hacer demasiado extenso el documento, consideramos conveniente no entrar en los detalles más específicos del protocolo. Sin embargo, sí que consideramos conveniente tener conocimiento sobre cómo se interconectan los elementos que se definen en el mismo. Esto queda reflejado con la figura 2.1¹:

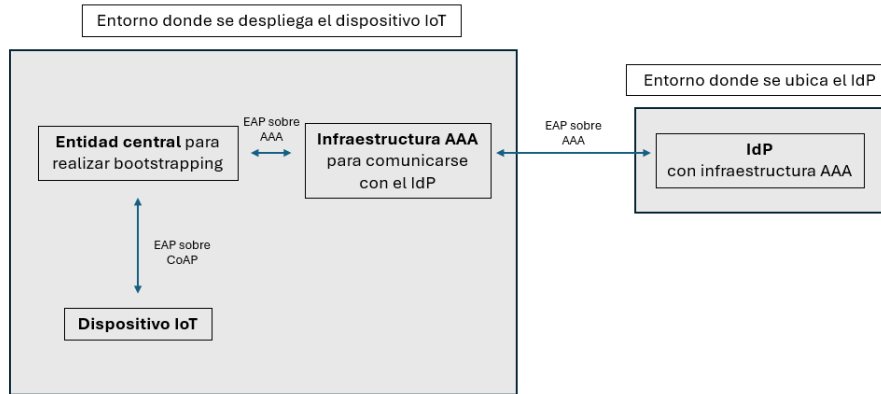


Figura 2.1: Interacción para el proceso de *bootstrapping*

2.1.1.1. Constrained Application Protocol (CoAP)

Tal y como se menciona en [13], CoAP es un protocolo que funciona en la capa de aplicación y está diseñado para dispositivos de capacidades restringidas. El objetivo de CoAP es proporcionar un protocolo que permita a este tipo de dispositivos comunicarse a través de Internet.

La principal característica en el diseño de CoAP es su semejanza con el protocolo Hypertext Transfer Protocol (HTTP) [14]. Este protocolo es demasiado costoso para dispositivos limitados, sin embargo, su modelo de solicitudes y respuestas es de gran utilidad. Por ello, CoAP busca adaptar este protocolo para conseguir un funcionamiento análogo en dispositivos con capacidades limitadas.

Por tanto, podríamos resumir CoAP [12] en una adaptación de HTTP para dispositivos restringidos en cuanto a capacidades, incluyendo así a los dispositivos IoT. Sin embargo, aunque CoAP está basado en HTTP tiene ciertas diferencias que es conveniente conocer. En particular, destacamos las que consideramos más relevantes:

1. Mientras que HTTP trabaja sobre Transmission Control Protocol (TCP) [15], CoAP lo hace sobre User Datagram Protocol (UDP) [16].
2. CoAP solo soporta ciertos métodos de HTTP. En concreto: GET, PUT, POST y DELETE.

¹El diagrama ha sido adaptado de [1]

3. CoAP no soporta todos los códigos de respuesta que se ofrecen en HTTP. Además, añade algunos propios.
4. CoAP añade una funcionalidad llamada observación. Esta permite que se notifique automáticamente de cambios en recursos previamente solicitados.
5. Aunque CoAP trabaja sobre UDP, permite trabajar con dos tipos de mensajes: confirmables y no confirmables. De este modo, si se elige trabajar con mensajes confirmables, el emisor del mensaje esperará una confirmación por parte del receptor del mensaje. Así, aunque se trabaje con UDP, podemos esperar un funcionamiento similar a los Acknowledgements (ACKs) de TCP.

El uso de CoAP puede ser muy interesante en situaciones donde tengamos que trabajar con dispositivos de capacidades limitadas, ya sea en cuanto a memoria, cómputo o energía, y busquemos una funcionalidad similar a la que nos daría el uso de HTTP. CoAP está pensado para reducir de forma considerable el consumo de recursos por parte del dispositivo. Además, la funcionalidad de observación de recursos mencionada puede ser muy interesante, en particular en entornos IoT. Por ejemplo, si trabajásemos con las temperaturas de un invernadero, podríamos hacer que se nos notificase automáticamente cuando los sensores encargados de realizar mediciones obtuviesen nuevos valores. Esta funcionalidad no es única de CoAP, pero si estamos en un entorno con dispositivos limitados, CoAP es una muy buena opción.

Por tanto, tal y como hemos expuesto, CoAP es un protocolo idóneo para comunicaciones de dispositivos IoT. Es por ello por lo que el modelo CoAP-EAP para el *bootstrapping* de estos dispositivos elige a CoAP para el transporte de los mensajes EAP.

2.1.1.2. Extensible Authentication Protocol (EAP)

El funcionamiento general de EAP está definido en [11]. La característica principal de este protocolo es su uso para la autenticación de entidades. Para ello, ofrece una gran variedad de posibilidades a través de lo que se denomina métodos EAP. Al finalizar la autenticación, EAP, proporciona claves compartidas para proteger las comunicaciones.

Para nuestro caso de estudio, CoAP-EAP, el método EAP que se utiliza para realizar la autenticación es EAP-PSK, definido en [17]. Este método se basa en el uso de una Pre-Shared Key (PSK) entre la entidad que se quiere autenticar y la entidad que lo autentica. Solo estas deben conocer el secreto compartido, de lo contrario la seguridad del método se ve comprometida. EAP-PSK tiene un diseño enfocado a la simplicidad para ser útil en dispositivos de todo tipo, especialmente en aquellos con baja capacidad de procesamiento y memoria.

El proceso de autenticación ofrecido con EAP-PSK se divide en las siguientes fases²:

1. La entidad que debe ser autenticada, la llamaremos cliente, recibe un mensaje EAP Request ID con el que se solicita su identidad. Debe responder a esta solicitud con un mensaje EAP Response ID en el que proporcione la información de identificación adecuada.
2. **EAP PSK 1.** La entidad encargada de la autenticación, la llamaremos servidor, recibe una identidad tras el intercambio de mensajes EAP Request ID y EAP Response ID. Para comprobar que verdaderamente está comunicándose con la entidad que recibe en el EAP Response ID, posee una PSK asociada a dicha entidad. De este modo, comienza un proceso de intercambio de 4 mensajes que tiene como objetivo autenticar al cliente. El mensaje EAP PSK 1 es el primero de dicha secuencia. En este, el servidor incluye su identidad y un valor aleatorio generado por él (RAND_S).
3. **EAP PSK 2.** El cliente responde con un mensaje que contenga su identidad, un valor aleatorio generado por él (RAND_P) y un valor calculado a partir de la PSK y los valores aleatorios e identidades intercambiadas hasta ese momento (MAC_P). De este modo, MAC_P sirve como forma de autenticación del cliente, pues el servidor, tras recibir el mensaje EAP PSK 2, puede recalcular este valor con la información que posee y, si coincide, significa que se está comunicando con el cliente correcto.
4. **EAP PSK 3:** El servidor envía un mensaje similar al EAP PSK 2 para autenticarse ante el cliente. En este incluye el valor MAC_S, análogo al MAC_P.
5. **EAP PSK 4:** El cliente confirma la autenticación del servidor. Además, se generan claves criptográficas que permiten securizar las comunicaciones.
6. El servidor envía un mensaje EAP Success para concluir que el proceso de autenticación ha terminado con éxito.

2.2. Trusted Execution Environment (TEE)

Los dispositivos IoT deben ser securizados para evitar problemas asociados con ataques de personas (ya sean a través de la red o mediante accesos físicos a los dispositivos), que busquen extraer información sensible de estos.

Para reforzar su seguridad, una estrategia efectiva consiste en incorporar un entorno de ejecución seguro dentro del sistema del dispositivo. Este tipo de entorno, conocido como TEE, permite aislar las funciones más sensibles, de modo que puedan ejecutarse

²Los pasos 1 y 6 son comunes en todos los métodos EAP.

de forma segura.

Para comprender mejor el concepto de TEE nos remitimos a una analogía presentada en [18]:

“Imaginemos que disponemos de una caja que contiene algo con un gran valor para nosotros. Cerramos esa caja con llave y la guardamos en la caja fuerte de un banco de modo que solo se puede abrir con la llave que nosotros tenemos. En el momento en que necesitamos acceder al contenido de la caja, esta se mueve a una habitación aún más segura donde podemos abrirla tranquilamente con nuestra llave. Una vez que terminamos de manejar su contenido, la cerramos y la devolvemos a la caja fuerte del banco. De este modo, el contenido de nuestra caja nunca se ve expuesto al exterior del entorno seguro que crean la llave, la caja fuerte del banco y la habitación privada en la que se abre la caja. Un TEE trabaja de forma similar, pero en el mundo del software”.

Así, un TEE se puede entender como una zona del sistema de un dispositivo que está aislada del resto, permitiendo gestionar la información sensible que maneja de una forma segura (incluso si el resto del sistema se ve comprometido). En concreto, un TEE proporciona un entorno seguro dentro de un dispositivo para ejecutar código que necesite de un alto nivel de protección. Destacar que las aplicaciones que se ejecutan dentro del TEE son llamadas Trusted Applications (TAs) [19].

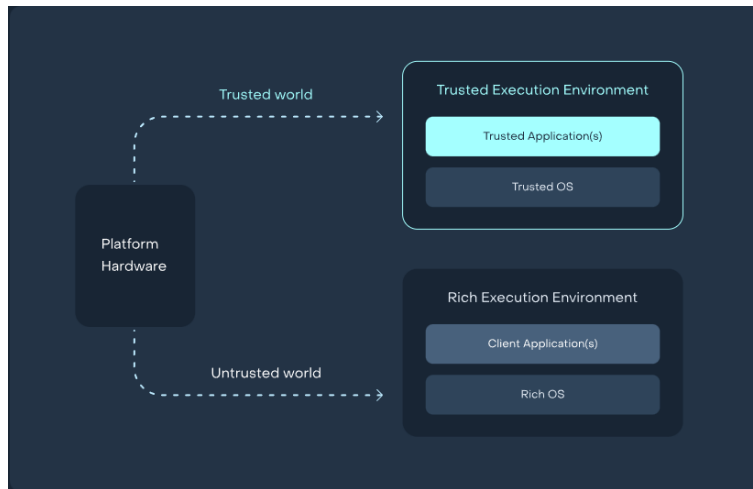


Figura 2.2: Estructura de un TEE

La figura 2.2³ muestra la estructura de un dispositivo que incorpora un TEE. El dispositivo (*platform hardware*) divide sus operaciones entre dos mundos: el *trusted world*, donde se encuentra desplegado el TEE y se ejecutan las TAs; y el *untrusted world*,

³Imagen extraída de: [20]

donde se ejecutan las aplicaciones normales y tenemos el funcionamiento habitual del dispositivo. Cabe destacar que, para permitir la interacción entre ambos mundos, es necesario hacer uso de una API.

Es importante tener claro qué propiedades se persiguen alcanzar al utilizar TEEs [20, 21]. Estas son:

- **Aislamiento.** Se debe crear un entorno dentro del dispositivo que esté aislado de los procesos y aplicaciones no críticas.
- **Confidencialidad.** Los datos de una TA no deben poder ser obtenidos por ninguna otra aplicación.
- **Integridad.** Las TAs no deben poder ser modificadas.
- **Ejecución segura.** Hay que evitar que nadie pueda descifrar qué ocurre mientras se ejecuta una TA.
- **Acceso controlado.** Cada TA debe poder acceder únicamente a sus propios recursos.

Concluimos esta sección con un ejemplo de uso de los TEEs: proteger las claves que se utilizan para cifrar las comunicaciones. En muchas ocasiones, estas claves están de forma clara en el código de nuestras aplicaciones. Esto provoca que, si alguien consigue acceder al código fuente, pueda obtenerlas. A través del uso de un TEE podemos proteger esas aplicaciones dentro del entorno seguro. Así, uno de los usos más habituales de los TEEs es cuando se necesita realizar operaciones criptográficas.

2.3. Aplicación de políticas de comportamiento en dispositivos IoT

Los dispositivos IoT son capaces de conectarse a Internet, lo que les lleva de manera inevitable a estar expuestos a amenazas. Por tanto, un objetivo que debe ser fundamental cuando se trabaja con IoT es minimizar los riesgos a los que están expuestos.

En este sentido, el MUD [6], un estándar definido por el IETF, permite definir en un fichero una serie de políticas que describan el comportamiento esperado en un dispositivo. Estas políticas pueden ser utilizadas para monitorizar comportamientos sospechosos (como se propone en [22]) o para reducir la superficie de ataque del dispositivo mediante su aplicación (como se plantea en este trabajo).

Por tanto, el MUD permite estandarizar la definición de políticas de comportamientos de dispositivos IoT. Los ficheros que contienen dichas políticas son llamados

ficheros MUD y el propio estándar MUD establece una arquitectura para desplegar y obtener estos ficheros.

Para comprender mejor el concepto de un fichero MUD recurrimos al ejemplo presentado en [6]: “Una bombilla tiene como objetivo iluminar una habitación, pero, además, podría ser diseñada para permitir que sea controlada de forma remota a través de la red. Cualquier acceso a la red por parte de la bombilla que no tenga este propósito supone una comunicación no deseada. Así, la bombilla no debe comunicarse con ningún servicio ni dispositivo que no esté relacionado con esta funcionalidad. Teniendo esto claro, podemos establecer que la bombilla solo debe poder acceder a Internet para conectarse al servicio que la controla de forma remota”. Este tipo de comportamientos son los que se busca controlar al definir un fichero MUD: se estudia el tipo de dispositivo y se restringen sus comunicaciones a través de la red para dotarlo de mayor seguridad frente a ataques.

2.3.1. Manufacturer Usage Description (MUD)

Los ficheros MUD definen las conductas permitidas en un dispositivo IoT. Por ello, para poder trabajar con ellos es conveniente conocer quién se encarga de definirlos, cómo se difunden, cuál es la estructura que deben tener y qué limitaciones presentan. Las respuestas a estas cuestiones se abordan en las siguientes secciones.

2.3.1.1. Definición de un fichero MUD

La entidad encargada de definir las políticas de comportamiento esperadas en un determinado dispositivo IoT se llama, según [6], *manufacturer*. Este término se usa con cierta ambigüedad, ya que normalmente hace referencia al fabricante del dispositivo. Sin embargo, en el contexto de los ficheros MUD, su significado puede variar en función del dispositivo IoT concreto. En el estándar citado, se expone que, aunque el término *manufacturer* puede resultar vago, hace referencia a una entidad dentro de la cadena de suministro del dispositivo que se encarga de definir un fichero MUD para describir su propósito de uso.

2.3.1.2. Difusión de un fichero MUD

Una vez que el *manufacturer* genera un fichero MUD, este debe poder ser consultado. El proceso para poder obtener el contenido de un fichero MUD es el siguiente:

1. El dispositivo IoT es capaz de proporcionar la ubicación del fichero MUD a través de una Uniform Resource Locator (URL) a la que se llama MUD URL. Esta información se comparte durante el proceso de *bootstrapping*.

2. Debe existir una entidad, denominada MUD Manager, que sea capaz de recibir un MUD URL y a partir de este generar una petición para obtener el fichero MUD.
3. Los ficheros MUD son almacenados en entidades llamadas MUD Server. Cuando un MUD Manager hace una petición de un fichero MUD a través de su MUD URL, el MUD Server es el encargado de responder con el fichero MUD pertinente.

De este modo, una vez se tiene el fichero MUD, se puede interpretar su contenido para aplicar las políticas de comportamiento definidas en este.

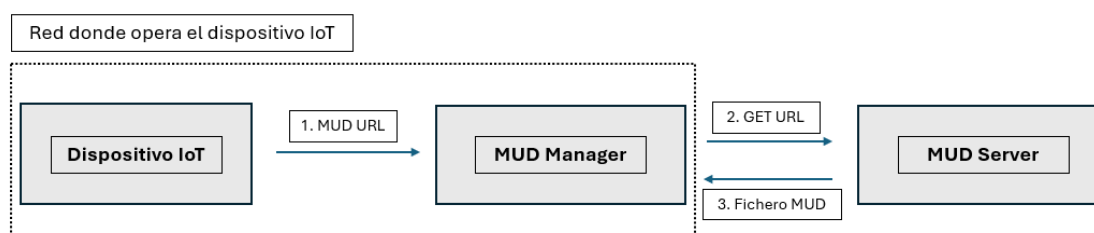


Figura 2.3: Proceso de obtención de un fichero MUD

La figura 2.3 muestra un esquema resumen de cómo se obtiene un fichero MUD⁴.

Es importante señalar que un fichero MUD puede ser modificado con el paso del tiempo, por lo que el MUD Manager debe estar preparado para tener esto en cuenta y actuar en consecuencia.

2.3.1.3. Estructura de un fichero MUD

La estructura de un fichero MUD está ampliamente detallada en [6]. Por ello, en esta sección se destacarán solo los aspectos que se consideran más relevantes.

La estructura de un fichero MUD está definida a través de un modelo Yet Another Next Generation (YANG) [23]. Además, para su transmisión por la red se usa serialización Javascript Object Notation (JSON) [24]. Su contenido se conforma a través de los siguientes elementos:

- **ietf-mud** [6]: Contiene información relevante para comprobar la validez del propio fichero MUD, además de una referencia a las Access Control Lists (ACLs)⁵ [25] que se definen en el fichero, indicando su nombre y el sentido de la comunicación en el que se aplican (hacia/desde el dispositivo).

⁴Imagen basada en [22].

⁵Una ACL constituye una política de red que puede aplicarse sobre un dispositivo.

- **ietf-access-control-list** [25]: Permite definir las reglas que conforman las diferentes ACLs que se quieren aplicar.
- **ietf-acldns** [6]: Permite utilizar nombres de dominio en reglas que se definan sobre una ACL.

Un fichero MUD válido contiene como contenedores raíz a *ietf-mud* e *ietf-access-control-list*. Es interesante destacar que es posible hacer uso de extensiones para poder añadir contenedores raíz, ampliando así la información transmitida en un fichero MUD. En el caso de querer utilizar extensiones estas deben ser declaradas dentro del contenedor *ietf-mud* usando el valor *extensions*.

2.3.1.4. Limitaciones de los ficheros MUD

Respecto a los ficheros MUD no solo es importante tener un modo estandarizado para la definición de políticas de comportamiento de un dispositivo y la obtención de ficheros de este tipo; también es necesario ser capaces de aplicar dichas políticas. Además, estas políticas pueden cambiar durante el ciclo de vida del dispositivo, requiriendo por tanto actualizarlas. Estos aspectos no son incluidos en la definición del estándar MUD [26].

Existen otros problemas asociados al uso del MUD tal y como se expone en [22]:

Limitación	Descripción de la limitación
Necesidad de un proceso de traducción	Las políticas que se definen a través de un fichero MUD no pueden ser directamente aplicadas. Es necesario realizar un proceso de traducción del contenido del fichero para poder procesar y aplicar estas políticas correctamente.
Falta de expresividad	Los ficheros MUD tienen limitaciones para poder expresar políticas que vayan más allá de la capa de red.
Existencia de conflictos	Es habitual que en los entornos donde se despliegan dispositivos IoT que hacen uso de ficheros MUD, existan ya definidas ciertas políticas de seguridad. En ocasiones, las políticas definidas en el entorno del dispositivo IoT pueden entrar en conflicto con las que aparecen indicadas en el fichero MUD.
Tiempo de validez de un fichero MUD	Entre el contenido de un fichero MUD, aparece el tiempo de validez de este. Así, el MUD Manager no debería comprobar actualizaciones de este fichero antes de expirar dicho periodo. Sin embargo, el trabajo en Internet implica cambios casi constantes, por lo que un fichero MUD podría ser modificado antes de que se cumpla el tiempo de validez. De este modo, seguir fielmente el estándar implicaría que pudiera darse el caso de que el fichero MUD usado esté obsoleto, por lo que estaríamos aplicando políticas incorrectas en el dispositivo.

Limitación	Descripción de la limitación
Actualización de los ficheros MUD	El estándar MUD define que el fichero MUD es obtenido cuando el dispositivo IoT envía el MUD URL al MUD Manager. Sin embargo, no especifica ningún modo de obtener el fichero MUD cuando haya actualizaciones de este. Tampoco define ningún mecanismo para que el <i>manufacturer</i> del dispositivo pueda avisar de cambios de este fichero.
MUD Server como único punto de fallo	Si el servidor que proporciona un fichero MUD se ve comprometido, el sistema se vería en problemas, pues podríamos utilizar ficheros MUD que no sean correctos.

Tabla 2.1: Limitaciones de un fichero MUD

Cabe mencionar que para algunos de los problemas mencionados ya se están ofreciendo soluciones. Por ejemplo, en el caso particular de la falta de expresividad del MUD, actualmente existe una propuesta para su extensión a través del uso del lenguaje Medium-level Security Policy Language (MSPL) [27, 28]. Dicha propuesta está descrita en [29] y propone ampliar el contenido del MUD mediante una extensión llamada `umu-mspl-list:mspls`. A través de esta se pueden incluir políticas de seguridad más complejas y que no estén enfocadas únicamente en la capa de red. Es importante tener en cuenta que la propuesta mencionada requiere introducir un paso intermedio para poder aplicar las políticas que se incluyan en el MUD, estas deben ser previamente traducidas a lenguaje MSPL para poder interpretarlas y así aplicarlas. Esta extensión es utilizada en este trabajo, en concreto, en el fichero MUD que se define en la sección 4.3.2.4 se explica cómo ha sido utilizada.

3. Análisis de objetivos y metodología

El objetivo de este trabajo es desarrollar una PoC del proyecto CERTIFY que está siendo desarrollado en la Universidad de Murcia. En concreto, se busca obtener un escenario simplificado que permita realizar una gestión segura del ciclo de vida de dispositivos IoT. Para ello, se presta especial interés a la fase de *bootstrapping* y se trabaja de forma básica con la aplicación de políticas de comportamiento en dispositivos IoT. Por tanto, la gestión de las fases de operación y mantenimiento del dispositivo se trata como aspecto secundario.

De este modo, el trabajo se divide en tres bloques:

1. **Proceso de *bootstrapping*.** Se ha buscado lograr una implementación funcional del modelo de *bootstrapping* propuesto en la Universidad de Murcia por García Carrillo en [1]. En concreto, la implementación utilizada es la que está actualmente aceptada por el IETF [8].
2. **Securización del dispositivo IoT.** Para dotar de la mayor seguridad posible a los dispositivos, se ha estudiado el despliegue de un TEE que permita proteger las operaciones críticas del dispositivo. Por ejemplo, las operaciones criptográficas durante el proceso de *bootstrapping*.
3. **Etapas de operación y mantenimiento.** Aunque la gestión de estas etapas no es cubierta en su totalidad, se han desarrollado mecanismos que permitan actuar sobre dispositivos que hayan completado la etapa de *bootstrapping*. En concreto, se ha realizado un despliegue que permita aplicar políticas de comportamiento que estén definidas en ficheros MUD.

Respecto a la metodología de trabajo seguida, esta no consiste en una investigación, sino en una PoC. De este modo, para la prueba del escenario desplegado se ha hecho uso de una Raspberry Pi 3 Model B y una Raspberry Pi 4 Model B como dispositivos IoT. Para la puesta en marcha de las entidades que no corresponden al dispositivo IoT, se ha utilizado un ordenador portátil de propósito general.

El trabajo desarrollado para crear el escenario IoT que se presenta en este trabajo se puede resumir en las siguientes etapas:

1. Despliegue y prueba de un servidor que permita realizar el *bootstrapping* siguiendo el modelo CoAP-EAP.

2. Despliegue de un TEE en una Raspberry Pi 3 Model B.
 3. Prueba del proceso de *bootstrapping* entre el servidor desplegado en la etapa 1 y el dispositivo IoT configurado en la etapa 2.
 4. Despliegue y prueba de un proceso que permita aplicar políticas de comportamiento en una Raspberry Pi 3 Model B que haya completado el proceso de *bootstrapping*. Siguiendo la terminología del proyecto CERTIFY a dicho proceso se le llama Enforcement and Reconfiguration Agent (ERA).
 5. Despliegue y prueba de un servidor capaz de proporcionar ficheros MUD a partir de un MUD URL
 6. Despliegue y prueba de un servidor capaz de procesar MUD URLs, solicitar los ficheros MUD asociados, y traducir las políticas que contengan para aplicarlas en un dispositivo IoT.
 7. Integración de las etapas 4, 5 y 6 para aplicar políticas de comportamiento definidas en ficheros MUD sobre dispositivos IoT.
 8. Prueba del escenario completo: unión de la etapa 3 y 7.
 9. Documentación del trabajo realizado.
-

4. Diseño y resolución del trabajo realizado

En esta sección se presenta el trabajo realizado para construir la PoC basada en el proyecto CERTIFY [5]. Para ello, utilizando como base código que pertenece a dicho proyecto, se discute cómo ha sido utilizado, modificado cuando ha sido necesario, y testeado para poder integrarlo en un escenario que cumpla con los objetivos definidos en la sección 3.

En concreto, se describe cómo se ha realizado el despliegue de un proceso de *bootstrapping* que siga el modelo CoAP-EAP. Tras esto, se define cómo realizar el despliegue de un TEE que permita proteger las operaciones críticas de una Raspberry Pi 3 Model B. Finalmente, se presenta el trabajo desarrollado para definir y distribuir un fichero MUD que pueda ser utilizado para aplicar políticas de comportamiento en un dispositivo IoT que haya completado el proceso de *bootstrapping*. Además, para que estas políticas puedan ser aplicadas en el dispositivo, se explica cómo desplegar un ERA en una Raspberry Pi 3 Model B. Una vez presentadas todas las entidades que constituyen el escenario IoT del trabajo, se expone una prueba del correcto funcionamiento de la PoC.

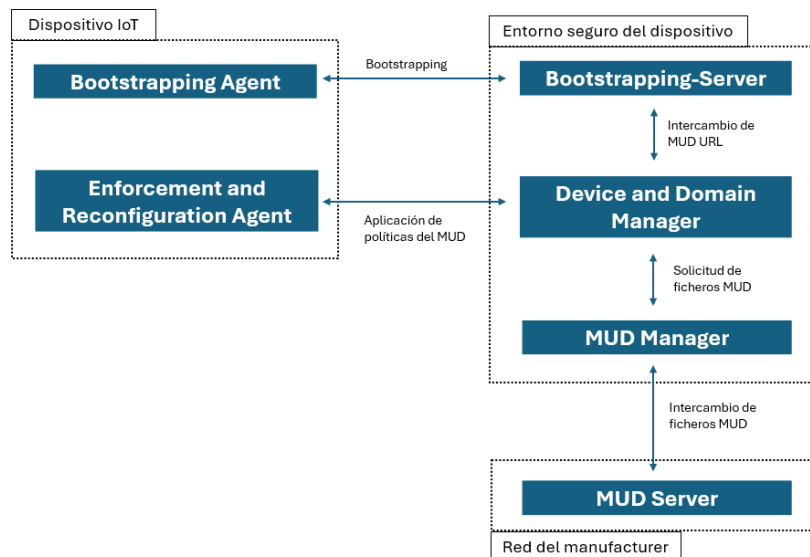


Figura 4.1: Escenario IoT a desplegar

La figura 4.1 muestra un resumen del escenario que se desarrolla en este trabajo. A lo largo de esta sección se irá describiendo el papel de cada una de las entidades que aparecen en este diagrama, así como el proceso que se siguió para incorporarlas en el escenario.

4.1. Proceso de *bootstrapping* CoAP-EAP

En esta etapa el objetivo era desplegar un servidor capaz de realizar el proceso de *bootstrapping* CoAP-EAP definido en [8]. Para ello, se hizo uso de dos materiales:

1. Aunque el modelo CoAP-EAP está definido en el estándar del IETF, se proporcionó un diagrama del flujo esperado durante el proceso de *bootstrapping*. Este diagrama pertenece al proyecto CERTIFY, cuyo objetivo está expuesto en [26]. Dado que este proyecto aún está en desarrollo y su material no me pertenece, este diagrama no se incluye por cuestiones de privacidad.
2. Una versión de un servidor que sigue el modelo CoAP-EAP, basada en la publicada en [30].

De este modo, dado que ya se disponía de un código completamente desarrollado para el servidor de *bootstrapping*, el objetivo era comprobar que este era funcional, pues, aunque había sido probado en simulaciones, no se había probado con máquinas reales.

Para comprobar el correcto funcionamiento de dicho servidor era necesario entender el modelo CoAP-EAP y el diagrama de flujo proporcionado. Una vez hecho esto, se debía comprobar que el código fuente del servidor seguía ambos modelos. Finalmente, se debía poner en marcha el servidor y ver que el intercambio de mensajes para completar el *bootstrapping* era correcto.

En este punto, es importante tener en cuenta que el código proporcionado pertenece a un proyecto que aún está vivo, lo que provocó tener que trabajar con varias versiones diferentes del servidor.

4.1.1. Prueba de una implementación del servidor de *bootstrapping*

Tras recibir la primera versión del código, lo primero que se hizo, fue tratar de ponerla en ejecución, para que, independientemente de que su implementación fuese correcta o no, se comprobase que no había ningún error de programación oculto por el uso de un entorno de simulación para su desarrollo.

Dado que el código estaba desarrollado en C, era necesario compilarlo. Para ello, previamente había que instalar librerías que se habían usado para desarrollar el código, pues estas son muy específicas y por lo general no suelen estar instaladas. En concreto, la prueba estaba siendo realizada sobre un Ubuntu 24.04, y con la instalación genérica, las librerías que se tuvieron que instalar fueron:

Librería	Descripción
libcoap3-dev	Permite trabajar con CoAP.
libmicrohttpd-dev	Permite implementar servidores HTTP de forma sencilla.
libssl-dev	Permite trabajar con OpenSSL [31], siendo este útil para trabajar con criptografía.
uuid-dev	Permite crear Universally Unique Identifiers (UUIDs) [32] de forma sencilla.
libcurl4-openssl-dev	Permite hacer peticiones a servidores web de forma segura para utilizarlo de manera similar a como se usa la herramienta de línea de comandos curl.
libcjson-dev	Permite trabajar con datos en formato JSON.

Tabla 4.1: Librerías necesarias para ejecutar el servidor de *bootstrapping*

Una vez instaladas todas las librerías necesarias y compilando con la herramienta gcc, se pudo poner en ejecución el servidor. Esta prueba detectó un problema: había un error de programación que impedía la ejecución. El motivo de este error era que se hacía una comprobación sobre el estado de un socket de la comunicación antes de inicializarlo, lo que provocaba la finalización prematura del código. Por tanto, se solucionó ese fallo y se reportó al desarrollador del código.

Una vez que el servidor estaba en funcionamiento, sin comprobar su implementación, se decidió testear si este era capaz de comunicarse con otra entidad. Para hacer esto, se utilizó un código de cliente (código que se debería instalar en un dispositivo IoT) de prueba que también se había proporcionado junto al del servidor de *bootstrapping*. Al lanzar ambas entidades, se localizó un nuevo problema, pero ahora en el cliente. Este hacía uso de la interfaz de red del dispositivo que ejecutase ese código; sin embargo, este valor era necesario introducirlo de forma estática en el programa, lo que provocaba que fuese dependiente de la máquina en la que se ejecutase.

Tras solucionar este problema y reportarlo, tanto cliente como servidor eran capaces de comunicarse y, por los mensajes de *debug* que se mostraban, parecían realizar un proceso de *bootstrapping*. Sin embargo, ahora era el turno de comprobar que el código desarrollado era correcto y seguía el modelo CoAP-EAP.

Antes de comenzar con esta tarea se recibió una nueva versión del código, que había corregido los errores reportados y algunos que aún no se habían descubierto. Por este motivo, se abandonó esta versión y se pasó a trabajar con la nueva.

4.1.1.1. Configuración de una Raspberry Pi 4 Model B

Antes de continuar con la explicación sobre el despliegue del servidor para el *bootstrapping*, es conveniente destacar dónde se probó el código del cliente proporcionado. Este código fue ejecutado desde una Raspberry Pi 4 Model B, para hacer el papel de dispositivo IoT que quería realizar el *bootstrapping*.

Para poder trabajar con una Raspberry Pi 4 Model B es necesario configurar un sistema operativo. En este caso, se optó por seguir la guía ofrecida en [33]. Esta ofrece un manual sencillo y directo para poner en marcha un dispositivo de este tipo. Así, obtenemos un sistema operativo desde el que poder ejecutar el código del cliente.

4.1.2. Actualización del servidor de *bootstrapping* proporcionado

Como hemos comentado, la nueva versión del servidor corregía errores de la primera. Además, pasaba a tener un código mucho más modularizado. En la primera versión, la funcionalidad se agrupaba en unos pocos ficheros, lo que hacía difícil seguir el código. Sin embargo, en esta nueva versión, el código había sido segmentado por funcionalidad, haciéndose mucho más fácil su comprensión.

En este punto, dado que el cliente de prueba y el servidor se ejecutaban y se comunicaban, tocaba comprobar que el proceso de *bootstrapping* seguía el modelo CoAP-EAP. Para hacer esto, se utilizó como base el diagrama del proyecto CERTIFY. Este diagrama define un proceso de *bootstrapping* que amplía el definido en [1]. Sin embargo, el servidor cubría únicamente el modelo CoAP-EAP y dejaba sin implementar el resto de los aspectos definidos en el diagrama. Esto no suponía ningún problema, pues podíamos utilizar la primera parte de ese diagrama para comprobar que se cumpliese con CoAP-EAP e ignorar el resto. En concreto, el diagrama define el intercambio de 55 mensajes, pero en este punto podíamos fijarnos únicamente en los primeros 24.

Para comprobar que se cumpliese con el modelo CoAP-EAP se hizo una lectura completa del código, identificando el momento en el que se implementaba el envío y recepción de cada uno de los mensajes que se debían intercambiar. Era importante detectar que estos mensajes contenían la información esperada y se procesaba correctamente. Además, dado que el código estaba vagamente comentado, durante la lectura completa del mismo, se fue comentando la funcionalidad: qué mensaje se estaba creando, cuándo se enviaba, a qué paso del diagrama correspondía, cómo se estaba generando la información, etc. De este modo, se conseguía un entendimiento completo del código y se detectaba si había algún error, ya fuese que se omitiese el envío de algún mensaje, faltase información en el mensaje que se enviaba, algún dato estuviese mal generado según los estándares o el procesamiento de la información recibida fuese erróneo. Destacar que, durante esta comprobación, se prestó especial interés en que las claves

que se debían generar tras el proceso EAP-PSK fuesen correctas en ambos lados de la comunicación (cliente y servidor).

Como resultado de este proceso, se comprobó que el proceso de *bootstrapping* que estaban siguiendo cliente y servidor cumplía con el modelo CoAP-EAP. Además, se detectaron algunos puntos de mejora, los cuales fueron reportados. Ejemplos que se incluyeron en este reporte fueron:

- Existencia de errores en el diagrama del proyecto CERTIFY. Estos iban desde errores ortográficos hasta errores en la información sobre el contenido del mensaje intercambiado (en ocasiones se indicaba el envío de cierta información que no se ajustaba con lo definido en el estándar).
- El código trataba de seguir la notación usada en el diagrama del proyecto CERTIFY, pero en ocasiones, esta notación se cambiaba. En pro de mantener uniformidad en el código, era interesante resolver estas diferencias.
- En el código del cliente se había olvidado el cálculo de la Master Session Key (MSK), clave criptográfica que se obtiene como consecuencia del proceso EAP-PSK. Era necesario añadir ese código.
- Existencia de errores en los comentarios originales del código. Algunos comentarios indicaban que se realizaban ciertas acciones sin que estas ocurriesen en realidad.
- En algunos mensajes, al crearlos, se completaba el *payload* duplicando cierta información.
- En ocasiones, se intentaba procesar información que no se incluía en el *payload* del mensaje recibido.
- No se actualizaba un campo de los utilizados para el protocolo EAP, en concreto, el encargado de relacionar solicitudes con respuestas.
- Se hicieron sugerencias sobre mejoras en la estructura del código:
 - Se definían funciones que no se llegaban a utilizar.
 - Se utilizaba números mágicos que dificultaban seguir el código (a futuro dificultarían el mantenimiento de este).
 - Se definían funciones que compartían gran cantidad de código. Estos trozos, en lugar de repetirlos, se podían llevar a funciones auxiliares para ahorrar líneas de código y por ende memoria.

Tras esto, se obtuvo el despliegue de la primera entidad del escenario IoT definido en la figura 4.1: Bootstrapping-Server.

4.2. Securitización de un dispositivo IoT

La sección 2.2 muestra la importancia de securizar los dispositivos IoT para proteger sus operaciones más críticas. Dado que el proceso de *bootstrapping* implica trabajar con criptografía, es muy interesante proteger estas operaciones, por tanto, el siguiente paso que se abordó fue securizar el dispositivo IoT que se iba a utilizar en nuestra PoC.

Para hacer esto se desplegó el TEE llamado OP-TEE y presentado en [34, 35]. Es importante tener en cuenta que este TEE tiene una limitación muy importante: no es posible desplegarlo en una Raspberry Pi 4 Model B (dispositivo IoT que habíamos elegido en un inicio para hacer nuestras pruebas). Por este motivo, se tuvo que utilizar una Raspberry Pi 3 Model B.

Para desplegar este TEE nos hemos basado altamente en la guía que ofrece la documentación oficial de OP-TEE [36]. Además, también nos hemos apoyado en la guía que se proporciona en [37]. De este último, hay que tener en cuenta que su guía está hecha para desplegar OP-TEE en un entorno simulado, por lo que hay que hacer ciertas modificaciones para adaptar la guía a un despliegue en un dispositivo real.

Dado que el despliegue de OP-TEE que se ha hecho es una combinación del explicado en [36] y [37], consideramos interesante ofrecer una guía que resuma los pasos seguidos. Esta puede ser de utilidad si se quiere replicar el despliegue de este TEE en una Raspberry Pi 3 Model B.

El primer paso es crear un contenedor Docker en el que trabajar para poder crear la imagen SD que se necesita instalar en la Raspberry. El motivo de hacer esto es que crear la imagen requiere el uso de herramientas que en una instalación predeterminada de Ubuntu no están presentes y además, no es común que sean utilizadas. Por ello, para evitar instalar en nuestra máquina herramientas innecesarias y no recurrir al uso de máquinas virtuales, podemos utilizar Docker. Así, el primer paso es crear un fichero DockerFile con el siguiente contenido:

```
FROM ubuntu:22.04
ARG DEBIAN_FRONTEND=noninteractive
ENV FORCE_UNSAFE_CONFIGURE=1
RUN apt update && apt upgrade -y
RUN apt install -y \
adb \
acpica-tools \
autoconf \
automake \
bc \
bison \
build-essential \
ccache \
cpio \
cscope \
curl \
```

```

device-tree-compiler \
e2tools \
expect \
fastboot \
flex \
ftp-upload \
gdisk \
git \
libattr1-dev \
libcap-ng-dev \
libfdt-dev \
libftdi-dev \
libglib2.0-dev \
libgmp3-dev \
libhidapi-dev \
libmpc-dev \
libncurses5-dev \
libpixman-1-dev \
libslirp-dev \
libssl-dev \
libtool \
libusb-1.0-0-dev \
make \
mtools \
netcat \
ninja-build \
python3-cryptography \
python3-pip \
python3-pyelftools \
python3-serial \
python3-tomli \
python-is-python3 \
rsync \
swig \
unzip \
uuid-dev \
wget \
xdg-utils \
xsltproc \
xterm \
xz-utils \
zlib1g-dev
RUN curl https://storage.googleapis.com/git-repo-downloads/repo > /bin/repo && chmod a+x /bin/↵
↵ repo

```

Una vez se tiene el Dockerfile preparado, se debe crear la imagen Docker que se necesita para iniciar el contenedor. Para ello, situándonos en la carpeta donde se tenga el Dockerfile, se debe ejecutar lo siguiente:

```
docker build -t optee .
```

El valor optee es el nombre que se ha elegido para el contenedor, puede ser cambiado por el nombre que se prefiera. Además, podemos comprobar que la imagen se ha creado correctamente utilizando:

```
docker images
```

Una vez creada la imagen, ya podemos acceder al contenedor. Para ello se utiliza el siguiente comando:

```
docker run -it optee /bin/bash
```

Es muy importante no incluir en este comando la opción `-rm`. Si hacemos esto, cuando nos salgamos del contenedor, automáticamente este se eliminará, perdiendo así todo el trabajo que se haya hecho dentro del mismo. En algunos despliegues esta característica es muy interesante, pero en el caso de OP-TEE no lo es, pues crear la imagen SD que necesita la Raspberry, sin tener en cuenta la configuración que se debe hacer dentro del Docker, dura aproximadamente dos horas utilizando un ordenador promedio. Si cada vez que queramos generar esa imagen debido a que queremos modificar el contenido que hay dentro del TEE tenemos que repetir todo el proceso, el tiempo de espera se puede convertir en una experiencia desagradable. Por este motivo evitamos el uso de `-rm`. Antes de continuar con la guía, es importante tener claro cómo podemos volver a acceder a este contenedor tras habernos salido, pues volver a ejecutar el comando anterior crearía un nuevo contenedor en el que no tendríamos ninguna información. Esto se hace del siguiente modo:

```
docker ps -a          # Buscar la entrada con nombre el dado al crear
                        # la imagen del contenedor
docker start -ai <ID_CONTENEDOR>
```

Tras este inciso volvemos a la guía para el despliegue de OP-TEE. Una vez accedemos al contenedor creado, se debe crear una carpeta de trabajo donde tendremos toda la información relacionada con OP-TEE:

```
mkdir /optee
cd /optee
```

En este directorio se debe clonar un repositorio de GitHub que contiene los datos necesarios para desplegar OP-TEE en una Raspberry Pi 3 Model B. Por tanto, es necesario configurar una cuenta de GitHub que nos permita acceder a este repositorio para descargarlo. Se debe hacer lo siguiente:

```
git config --global user.name "your_user_name"
git config --global user.email "yor_email"
```

Así, se puede descargar el contenido del repositorio a través del siguiente comando:

```
repo init -u https://github.com/OP-TEE/manifest.git -m rpi3.xml && repo sync -j10
```

Al ejecutar este comando es posible que ocurra un error. Este se debe al uso de la opción `-j10`, que busca acelerar el proceso lanzando 10 hilos en paralelo. Si se produce este error podemos solucionarlo quitando el paralelismo. Aunque incrementará el tiempo de espera, evita el problema. Tras completar este paso, se debe ejecutar lo siguiente:

```
cd build
make -j3 toolchains
```

Este paso puede dar bastantes problemas, pues no siempre se ejecuta correctamente a la primera. Para asegurarnos de que ha tenido éxito se debe comprobar el contenido de los subdirectorios de `/optee/toolchains`. Si alguno de estos subdirectorios está vacío o tiene algún fichero `.tar` ha habido un error durante la ejecución del comando. Para solucionarlo, se debe borrar el contenido de `/optee/toolchains` y repetir el proceso. Se recomienda, igual que ocurría con el paso anterior, reducir el paralelismo.

La siguiente tarea es obtener el código necesario para que se pueda ejecutar el *bootstrapping*, es decir, necesitamos un código de cliente que funcione con el servidor desplegado en la sección 4.1. Esta parte del trabajo reutiliza, igual que ocurría con el servidor de *bootstrapping*, código que pertenece al proyecto CERTIFY. Por ello, se debe utilizar el código disponible en [37]¹. Para hacer esto, se debe hacer lo siguiente:

```
cd /optee/  
git clone https://ants-gitlab.inf.um.es/jmanuel/certify-op-tee
```

Una vez tenemos el código, debemos modificar un fichero para establecer la dirección IP en la que está trabajando el servidor de *bootstrapping*. Para ello, se debe modificar el siguiente fichero:

```
/optee/certify-op-tee/security_api/host/rfc4764.c
```

Tras esto, se debe hacer lo siguiente:

```
cp -r certify-op-tee/security_api/ optee_examples/  
cd optee_examples/security_api/  
./config.sh 64
```

Con todo esto, ya se tiene todo preparado para generar la imagen SD de la Raspberry. Esto se hace del siguiente modo:

```
cd optee/build  
apt install parted fdisk udev dosfstools  
make
```

La primera vez que se ejecuta esto, como ya se ha indicado, tarda alrededor de dos horas. Sin embargo, si se trabaja siguiendo esta guía, las siguientes modificaciones de la imagen SD serán muchísimo más rápidas.

Nota. Si al ejecutar el `make` recibimos algún error, podemos ejecutar `make -n` para ver qué comandos se deben ejecutar cuando hacemos el `make`. Así, si el error ocurrió en los pasos finales, tras solucionarlo, podemos ejecutar manualmente los comandos que hayan faltado. Esto puede ser de gran utilidad, pues, por ejemplo, se nos podría haber olvidado instalar las herramientas indicadas en el paso previo al `make`. Esto provocaría

¹Es importante tener en cuenta que al pertenecer este código al proyecto CERTIFY, el cual está aún en desarrollo, el código que aparece en el GitHub puede haber sido actualizado respecto a la versión utilizada en este trabajo.

que el make fallase justo en el último comando, por lo que podemos ejecutarlo a mano tras instalarlas.

Tras realizar este proceso, en `/optee/out/` tenemos la imagen SD que se necesita para la Raspberry Pi 3 Model B. En concreto, en este directorio se debe buscar el archivo `rpi3-sdcard.img` y copiarlo en la SD que se vaya a usar para la Raspberry. El siguiente paso es explicar cómo hacer esto.

Lo primero que se debe hacer es conectar la SD a nuestro PC y vaciar su contenido (si es que tiene alguno):

```
lsblk                                # Vemos dónde se ha montado la SD en
                                    # nuestro sistema

sudo umount <RUTA1>                 # Desmontamos todos los puntos dónde se
...                                 # haya montado la SD
sudo umount <RUTAN>

sudo fdisk <NOMBRE_SD>              # Accedemos a la SD con fdisk usando el
                                    # nombre obtenido con lsblk. El objetivo
                                    # será borrar todas las particiones que
                                    # tenga y crear una nueva desde cero. Es
                                    # importante guardar los cambios antes
                                    # de salir

sudo mkfs.vfat -F 32 <PARTICIÓN_SD_CREADA> # Formateamos la partición creada en la SD
```

Tras esto, se debe copiar la imagen creada en la SD:

```
# Copiamos la imagen creada del contenedor a nuestro PC
docker cp <CONTAINER_ID>:/optee/out/rpi3-sdcard.img .

# Copiamos la imagen a la SD
sudo dd if=rpi3-sdcard.img of=<NOMBRE_SD> bs=1024k conv=fsync status=progress
```

Por último, antes de pasar a comprobar que el funcionamiento con el TEE creado para la Raspberry es correcto, se puede hacer una comprobación rápida para ver que el contenido de la SD es el adecuado, pues en ocasiones la operación de copia de la imagen puede fallar, provocando que, si se va directamente a la Raspberry, no arranque. Esta comprobación se puede hacer del siguiente modo:

```
# Vemos las particiones que tiene la imagen creada (deberían ser 2)
sudo fdisk -l rpi3-sdcard.img

# Comprobamos que nuestra SD tiene las mismas particiones
sudo fdisk -l <NOMBRE_SD>

# Para cada partición que tenga la SD hacemos el siguiente proceso (el objetivo es
# comprobar que tiene la estructura de ficheros esperada):
sudo mkdir -p /mnt/<RUTA-DESEADA-PARA-LA-PRUEBA>
sudo mount <PARTICIÓN-X-SD> /mnt/<RUTA-DESEADA-PARA-LA-PRUEBA>
ls <PARTICIÓN-X-SD> /mnt/<RUTA-DESEADA-PARA-LA-PRUEBA>
sudo umount /mnt/<RUTA-DESEADA-PARA-LA-PRUEBA>
```


Tras completar este último paso, se tiene una SD lista para usar en la Raspberry con el código necesario para desplegar un cliente seguro durante el proceso de *bootstrapping*. Por tanto, se tendría desplegada la segunda entidad del escenario IoT definido en la figura 4.1: *Bootstrapping Agent*. Sin embargo, antes de dar por hecho que esto es así, se debe comprobar que el *bootstrapping* se completa de forma exitosa. Esta es la tarea que se debe hacer antes de dar por concluida esta sección.

4.2.1. Prueba del proceso de *bootstrapping* con un cliente seguro

La prueba de que el TEE y el *bootstrapping* funcionan correctamente es muy sencilla gracias a que tanto los desarrolladores de OP-TEE como el desarrollador del código del cliente para el *bootstrapping* han desarrollado tests. Así, ejecutando estos tests y viendo que son pasados con éxito, se puede dar por concluida la prueba.

Una vez se arranca la Raspberry, disponemos de una terminal en la que se pueden ejecutar comandos. Para probar los tests de los desarrolladores de OP-TEE basta con ejecutar el comando `xtest`, mientras que para ejecutar los test relacionados con el *bootstrapping*, se debe ejecutar el comando `security_api_test`². Durante la prueba, se obtuvieron los siguientes resultados:

- Los tests creados por los desarrolladores de OP-TEE fueron pasados con éxito.
- Los tests relacionados con el *bootstrapping* dieron error en su ejecución. Esto fue reportado al desarrollador del código y se consiguieron detectar dos problemas:
 1. **Había un error en el nombre de una de las funciones del código.** Dos ficheros tenían una función llamada `main`. Esto hacía que el punto de comienzo del programa entrase en conflicto entre estos ficheros y se rompiese el orden de ejecución, provocando la finalización prematura del programa.
 2. **El código del cliente era incompatible con el del servidor.** En la sección 4.1 habíamos hablado de que junto al servidor se había utilizado un cliente de prueba. Para el desarrollo del cliente securizado se pensaba que el código que se estaba instalando en la Raspberry era el mismo que el probado en aquel momento, pero modificado para hacer uso del TEE. Sin embargo, la realidad estaba muy alejada de esto. Para encontrar el origen del problema hubo que ponerse en contacto con los desarrolladores de ambos códigos. Tras hablar con ellos, se identificó el problema. El servidor había sido desarrollado siguiendo el modelo CoAP-EAP; sin embargo, no había sido posible lograr el uso de CoAP con OP-TEE, por lo que el desarrollador del cliente securizado optó por transportar los mensajes EAP directamente

²Para que este test funcione es necesario que el servidor de la sección 4.1 esté en funcionamiento y que la Raspberry esté conectada a Internet. La forma más cómoda de hacer esto último es utilizar Ethernet.

sobre TCP. Esto provocaba que el cliente securizado fuese totalmente distinto del probado al desplegar el servidor de *bootstrapping*. Además, al usar cliente y servidor protocolos de transporte distintos (servidor UDP, cliente TCP), la comunicación entre ellos no era posible. Se plantearon entonces dos posibles soluciones:

- a) Desarrollar un proxy que hiciese de traductor entre cliente y servidor.
- b) Actualizar el servidor para trabajar directamente con EAP sobre TCP.

Cualquiera de las soluciones elegidas no era idónea, pues se descartaba el uso de CoAP y UDP, el cual es un aspecto clave para seguir el modelo CoAP-EAP. Sin embargo, dadas las limitaciones de tiempo para completar el trabajo y la dificultad que suponía trabajar con OP-TEE, hubo que asumir esta limitación en el escenario. Finalmente, la solución elegida fue la segunda.

De este modo, tras encontrar los problemas mencionados y ponerles solución, se obtuvo una tercera versión del servidor de *bootstrapping*. Además, hubo que regenerar la imagen de la SD para actualizar el código del cliente (esto solo requirió actualizar el código de GitHub y volver a hacer el make). Tras esto, se repitió la prueba con las nuevas versiones de cliente y servidor y todos los tests fueron pasados exitosamente, completando así con el trabajo en el *Bootstrapping Agent* de la figura 4.1.

4.3. Infraestructura para aplicar políticas de comportamiento

Una vez desplegado el modelo CoAP-EAP para realizar el proceso de *bootstrapping* en un dispositivo IoT, el paso final para completar la PoC propuesta en este trabajo, es realizar una aproximación a la gestión del dispositivo tras avanzar a la siguiente fase de su ciclo de vida: operación. En esta sección, se explica el proceso seguido para desplegar la infraestructura necesaria que permita aplicar políticas de seguridad sobre un determinado dispositivo IoT.

4.3.1. Despliegue de un ERA en un dispositivo IoT

Tras desplegar un dispositivo IoT capaz de realizar el *bootstrapping*, la siguiente tarea es configurarlo para que sea capaz de recibir órdenes de entidades externas que permitan tanto ajustar sus políticas de comportamiento como ejecutar acciones sobre el mismo.

La entidad desplegada para poder hacer esto, siguiendo la terminología del proyecto CERTIFY, es llamada Enforcement and Reconfiguration Agent (ERA). Además, se ha

reutilizado la implementación desarrollada en dicho proyecto para este trabajo.

Para desplegar el ERA en la Raspberry hay que tener en cuenta que se debe modificar el código del que dispone el sistema OP-TEE. Por tanto, es necesario regenerar la imagen SD que se utiliza. En concreto, se usó una nueva versión de [37]. Además, el ERA diseñado para el proyecto CERTIFY estaba desarrollado en Python, por lo que se debía instalar Python en la Raspberry. Este paso fue bastante complejo de realizar, pues no se encontró ninguna fuente que diese directrices sobre cómo añadir Python en una Raspberry Pi 3 Model B que haga uso de OP-TEE. Por este motivo, se considera conveniente proporcionar una guía sobre cómo realizar la instalación de Python.

Cuando se desplegó OP-TEE en la sección 4.2 se creó en el contenedor Docker el directorio /optee. En este se dispone de una estructura de directorios que permite hacer la construcción del sistema a través de la herramienta make. Para hacer esto, dentro de los diferentes directorios se encuentran ficheros Makefile que se van llamando entre sí para acabar formando la imagen de la SD. Por tanto, para añadir Python al sistema es necesario encontrar el fichero exacto en el que se debe indicar que se haga esto. Tras estudiar el contenido de los diferentes ficheros Makefile, se llegó a la conclusión de que la clave estaba en los directorios /optee/buildroot y /optee/out-br.

En /optee/out-br hay un fichero oculto, .config, en el que se debe indicar que se quiere añadir Python al sistema. Para modificar este archivo, no es conveniente hacerlo a mano, pues esto puede ser complejo y acabar con errores. En su lugar, se puede hacer uso de una interfaz gráfica accesible a través de /optee/buildroot. Así, el proceso que se debe seguir para instalar Python (asumiendo que se trabaja dentro del Docker) es el siguiente:

```
# Accedemos a la interfaz gráfica a través de /optee/buildroot e indicamos con la opción "0"
# que la configuración seleccionada y los ficheros generados se guarden en
# /optee/out-br. De este modo actualizamos el fichero .config de /optee/out-br
cd /optee/buildroot
make 0=/optee/out-br menuconfig

# Dentro de la interfaz gráfica seleccionamos lo siguiente:
"Target packages" >> "Interpreter languages and scripting" >> "[*] python3"

# Guardamos los cambios y nos salimos de dicha interfaz
# Compilamos la configuración elegida para actualizar .config de /optee/out-br
make 0=/optee/out-br
```

Una vez configurado Python, para terminar con el despliegue del ERA, se debe actualizar el código con el que cuenta la Raspberry. En concreto, hay que actualizar el código procedente de [37]³ y añadir los ficheros Python proporcionados para desplegar el ERA en /optee/out-br/target/root. Tras esto, se puede regenerar la imagen para la SD.

³Este código está en continuo cambio por ser parte de CERTIFY.

4.3.1.1. Prueba del ERA

Para probar el funcionamiento del ERA, igual que ocurrió con el *Bootstrapping Agent* de la sección 4.2, el autor del ERA diseñó un test que comprueba la correcta funcionalidad de este. El test incluye realizar el *bootstrapping* y, una vez se completa, el dispositivo pasa a ejecutar el ERA. Para poder comunicarse con este, se proporcionó otro *script* en Python en el que se definen dos comandos de prueba⁴:

1. Reiniciar el dispositivo para poder realizar un nuevo *bootstrapping*.
2. Cambiar el algoritmo de firma que utiliza el dispositivo.

Mediante la ejecución del test junto a los comandos anteriores, se puede comprobar la correcta funcionalidad del ERA. Sin embargo, para completar el despliegue de la entidad *Enforcement and Reconfiguration Agent* de la figura 4.1 es necesario que los comandos lleguen de la entidad *Device and Domain Manager*, definida también en dicha figura. Por tanto, el despliegue del ERA se podrá dar por concluido una vez los comandos lleguen de la entidad adecuada y se apliquen correctamente en el dispositivo. Esto es lo que estudiamos en la siguiente sección del documento.

4.3.2. Despliegue de infraestructura para trabajar con ficheros MUD

Una vez tenemos la Raspberry Pi preparada para completar un proceso de *bootstrapping* y recibir comandos que regulen su comportamiento, es momento de desarrollar las entidades necesarias para generar dichos comandos de forma adecuada. Para hacer esto, nos basamos en el uso de ficheros MUD, desplegando así la infraestructura necesaria para: proporcionar ficheros MUD; recuperar ficheros MUD a partir de un MUD URL; y convertir las políticas definidas en un fichero MUD a comandos comprensibles para el ERA desplegado. Destacar que, para las pruebas de dicha infraestructura, se utilizará un fichero MUD que es definido desde cero en este trabajo.

A la entidad central que gestionará el trabajo con los MUD se le llamará *Device and Domain Manager* (DDM), tal y como se propone en [38]. Además, para desplegar toda la funcionalidad mencionada se toma como punto de partida código que ha sido desarrollado para el proyecto CERTIFY. Es importante tener en cuenta que este proyecto es ambicioso respecto al uso de los MUDs, cubriendo así aspectos que exceden el objetivo de este trabajo. Por este motivo, el código reutilizado cuenta con partes que no son útiles en la PoC que se propone en este documento, teniendo, por tanto, que detectarlas y eliminarlas.

⁴El ERA desplegado solo es capaz de ejecutar los dos comandos mencionados.

Otro aspecto a destacar sobre el código del proyecto CERTIFY es que está desarrollado para interactuar con el dispositivo IoT no solo una vez que finaliza el *bootstrapping*, sino que también pretende intervenir a mitad de este. El motivo de hacer esto es que las políticas de comportamiento se deben aplicar en la configuración inicial del dispositivo y no esperar a que esté ya funcionando. Sin embargo, ni el servidor de *bootstrapping* ni el ERA desplegados están preparados para interactuar con la parte de gestión de políticas durante el *bootstrapping*. Estas entidades están aún en desarrollo, por lo que las diferentes partes del proyecto CERTIFY no están perfectamente integradas entre sí. El ERA desplegado se activa una vez completado el *bootstrapping*, mientras que el servidor de *bootstrapping* simula la comunicación con la parte de gestión de políticas⁵. Así, la falta de integración entre el ERA, el servidor de *bootstrapping* y el código que gestiona las políticas, provoca tener que hacer modificaciones importantes para poder conectar todo el escenario.

4.3.2.1. Actualización del *Bootstrapping-Server*

Como ya hemos comentado, con la versión actual del servidor de *bootstrapping* y del ERA es imposible que la gestión de políticas se aplique también durante el proceso de *bootstrapping*. Por ello, por cuestiones de tiempo, se tomó la decisión de que las políticas de comportamiento se aplicasen una vez completado el *bootstrapping*.

Esta decisión buscaba no tener que modificar el servidor de *bootstrapping*. Sin embargo, hay un aspecto que debemos tener en cuenta: el envío del MUD URL del dispositivo, valor necesario para poder obtener el fichero MUD que genera el *manufacturer*. Este valor es enviado por el dispositivo al *Bootstrapping-Server* y es necesario que se reenvíe al DDM en el mismo momento en que se recibe. Esto permite que, una vez completado el *bootstrapping*, se sepa cómo obtener el fichero MUD del dispositivo. Este hecho genera ciertas dificultades, pues como ya hemos expuesto, el servidor de *bootstrapping* no está desarrollado para comunicarse con el DDM, sino que simula la comunicación enviando la información del dispositivo a un servidor “vacío”. Si nos limitamos a cambiar dicho servidor por el DDM sin preocuparnos de nada, se bloquearía el proceso de *bootstrapping*, pues el *Bootstrapping-Server* quedaría a la espera de una respuesta la cual no es capaz de procesar.

Para solventar este problema, se buscó una solución intermedia que afectase lo menos posible a la estructura tanto del DDM como del servidor de *bootstrapping*. La solución planteada consistió en modificar el *Bootstrapping-Server* para que la información que mandaba al servidor “vacío” fuese reenviada automáticamente al DDM. Para evitar

⁵La simulación mencionada consiste en publicar en un servidor de prueba, datos del dispositivo IoT que está tratando de realizar el *bootstrapping*. Este servidor de prueba automáticamente responde con una respuesta sin contenido relevante, la cual es ignorada en el servidor de *bootstrapping* y se continúa con el proceso.

que este último pudiese enviar un mensaje de respuesta, se estableció un *timeout* muy bajo que cerrase la conexión antes de que el DDM pudiese generar una respuesta. De este modo, el servidor de *bootstrapping* continuaría su ejecución sin verse alterado por la funcionalidad del DDM. Este cierre de conexión puede provocar un error en el DDM si este trata de aplicar las políticas en el dispositivo, pues el ERA no está aún en funcionamiento. Por este motivo, se decidió implementar una interrupción en la ejecución del DDM para que el administrador del sistema decidiese retomarla una vez completado el *bootstrapping* y evitar así el error.

Como es evidente esta solución supone fuertes limitaciones en el funcionamiento del escenario, pues solo podremos gestionar un dispositivo IoT en cada momento. Además, la labor del DDM debería estar automatizada para no depender de la supervisión de una persona. Sin embargo, dadas las limitaciones presentadas en el servidor de *bootstrapping* y en el ERA que se tenían desplegados, esta era la opción más viable para completar el escenario.

4.3.2.2. Despliegue de un servidor que proporcione ficheros MUD

Suponiendo que tenemos ficheros MUD válidos, haciendo uso de Python se ha desplegado un servidor HTTP que sirva peticiones GET para proporcionar ficheros MUD. Es importante tener en cuenta que para que el servidor fuese completamente funcional, debería tener *endpoints* configurados que correspondiesen a MUD URLs. Sin embargo, debido a la naturaleza de este proyecto, nuestro dispositivo no tiene un MUD URL asociado. En su lugar, tiene una cadena de prueba que simboliza este valor. Es por ello, que el servidor está configurado para responder peticiones a *endpoints* del tipo `/mud/<device_id>`, donde `device_id` es un identificador del dispositivo IoT del que se quiere obtener el fichero MUD asociado⁶. Si quisiésemos hacer más realista este escenario, podríamos definir para el dispositivo IoT de prueba un MUD URL que tuviese un mayor sentido, y configurar entonces este servidor para responder únicamente a dicho *endpoint*. Destacar que, para esta prueba, este servidor proporcionará únicamente un fichero MUD, el definido en la sección 4.3.2.4.

Así, con el despliegue de esta entidad tenemos listo el MUD Server definido en la figura 4.1.

⁶Este valor es intercambiado entre el dispositivo IoT, el servidor de *bootstrapping* y el DDM, por lo que podemos hacerlo llegar hasta este servidor.

4.3.2.3. Despliegue de un servidor que solicite ficheros MUD

Una vez que tenemos un servidor que es capaz de proporcionar ficheros MUD ante peticiones GET, necesitamos una entidad que sea la encargada de hacerle peticiones. Esta es la que llamamos MUD Manager en la figura 4.1.

La implementación de esta entidad está altamente basada en la desarrollada para el proyecto CERTIFY, pero ha sido simplificada para tener lo estrictamente necesario para el despliegue de este escenario. Así, mediante Python se define un servidor HTTP que, mediante peticiones GET, es capaz de obtener un fichero MUD a partir de un MUD URL. Este proceso es iniciado cuando el MUD Manager recibe un mensaje POST al *endpoint* /mud. Dicho mensaje debe ser enviado por el DDM y tendrá en su contenido un MUD URL y el identificador del dispositivo IoT al que corresponde dicho MUD URL. Este identificador es lo que llamamos <device_id> en la sección 4.3.2.2.

Un aspecto a destacar sobre esta implementación es que, aunque el código se ha dejado listo para poder utilizar MUD URLs, cuando se va a hacer la petición al MUD Server, independientemente del MUD URL que se haya recibido, siempre se utiliza el mismo *endpoint*: /mud/prueba. Esto se hace así en pro de la simplicidad, pues como ya hemos comentado, los MUD URLs que se utilizan son simples cadenas de texto de prueba. Además, el MUD Server tiene un único MUD disponible. Por tanto, para la prueba del correcto funcionamiento, no es importante el *endpoint* que usemos para el GET⁷.

4.3.2.4. Definición de un fichero MUD

El objetivo de toda la sección 4.3.2 es poder aplicar políticas de configuración en nuestra Raspberry Pi una vez ha completado el proceso de *bootstrapping*. Para poder hacer esto necesitamos generar un fichero MUD que sea compatible con el ERA desplegado, es decir, el MUD debe contener políticas soportadas por el ERA.

En nuestro caso, se decidió optar por crear un MUD que tuviese la política de cambio de algoritmo de firma⁸. Además, se completó el contenido de este fichero con políticas de red que definiesen un MUD más completo, aunque estas últimas no fuesen usadas.

En esta sección se expone cómo se ha realizado la definición del fichero MUD que es utilizado para la Raspberry. Basándonos en [6] se crea la estructura básica de un fichero MUD que cumpla con el estándar. Para ello, se hace uso de los contenedores ietf-mud:mud e ietf-access-control-lists:acls. En estos se define el siguiente contenido:

⁷No obstante, el código se ha dejado listo para poder pasar a usar MUD URLs cuando se desee.

⁸Esta política es una de las soportadas por el ERA desplegado, tal y como se describe en la sección 4.3.1.1

- **ietf-mud:mud.** Los principales aspectos que se incluyen en este bloque son la definición del nombre de las ACLs que aparecen en el contenedor `ietf-access-control-lists:acls` y la declaración de las extensiones que se utilizan para ampliar el estándar MUD.
- **ietf-access-control-lists:acls.** En este bloque definimos una política de comunicación para el dispositivo. Asumiendo que todo lo que no se especifique mediante ACLs será considerado como tráfico no permitido, se establecen reglas para restringir la comunicación del dispositivo IoT únicamente a conexiones con dispositivos del entorno donde ha sido desplegado (se supondrá que dicho entorno es la red 192.168.100.0/24). En particular, el dispositivo IoT solo podrá generar tráfico hacia dispositivos de esa red si utiliza Hypertext Transfer Protocol Secure (HTTPS) [39], mientras que podrá recibir tráfico de cualquier tipo siempre que provenga de esta red.

Para evitar el problema de falta de expresividad del MUD mencionado en la sección 2.3.1.4, se hace uso de la extensión propuesta en [29]: `umu-mspl-list:mspls`. De este modo, podemos expresar políticas más complejas que las que permiten definir las ACLs. En nuestro caso, se hace uso de esta extensión para:

- **Definir la política de cambio de algoritmo de firma en el dispositivo:** se debe pasar a usar Hash-based Message Authentication Code (HMAC) [40].
- **Ampliar la política de red definida mediante ACLs:** se añade que todo el tráfico que llegue al dispositivo desde un servidor de *bootstrapping* de la Universidad de Murcia sea permitido.

El contenido del fichero MUD resultante se puede ver en el anexo A. Destacar que la definición del MUD fue una tarea bastante compleja, pues este debía ser aceptado por un traductor diseñado para trabajar junto al DDM. La problemática estaba en que el funcionamiento exacto de este traductor era desconocido, pues la persona que lo desarrolló no estaba disponible. Por tanto, hubo que descifrar cómo funcionaba el traductor para crear un MUD con sentido y que a la vez fuese aceptado por este.

El fichero MUD definido es el que proporciona el MUD Server explicado en la sección 4.3.2.2. De este modo, cuando se procese la política de cambio de algoritmo de firma, esta será enviada al ERA, mientras que el resto de las políticas podrían ser enviadas a entidades que gestionen la red donde trabaja el dispositivo IoT, por ejemplo, a una Software-Defined Networking (SDN) [41], tal y como se sugiere en [22].

Para concluir con esta sección, cabe mencionar que el significado de todos los campos utilizados en los contenedores `ietf-mud:mud`, `ietf-access-control-list:acls` y `umu-mspl-list:mspls` puede ser consultado en [6], [25] y [29]. Por ello, para no hacer demasiado extenso el documento se evita poner las definiciones aquí.

4.3.2.5. Coordinación de la aplicación de políticas

Una vez tenemos toda la infraestructura relacionada con la aplicación de políticas basada en ficheros MUD es necesario conectar todo el escenario. Esto se hace a través del *Device and Domain Manager* presentado en la figura 4.1. El DDM desplegado para este trabajo tiene las siguientes características:

1. Se comunica con el *Bootstrapping-Server* para obtener información sobre los dispositivos que tratan de realizar el proceso de *bootstrapping*. Toda la información sobre estos dispositivos es almacenada en una base de datos gestionada por el DDM y cierta información que se utiliza en esa base de datos se obtiene a través de esta comunicación, como la dirección IP o el MUD URL.
2. Solicita al MUD Manager los ficheros MUD de los dispositivos IoT que está gestionando. Para ello hace uso de los MUD URLs que va obteniendo.
3. Interpreta el contenido de los ficheros MUD que obtiene del MUD Manager y los traduce a políticas de comportamiento que puedan ser utilizadas. Esta traducción se hace a lenguaje MSPL siguiendo el modelo sugerido en [29, 38].
4. Interpreta las políticas obtenidas con la traducción para enviar comandos al ERA cuando sea pertinente.

Para desplegar dicho DDM los pasos que se han realizado pueden ser resumidos del siguiente modo:

- **Paso 1. Creación de un entorno virtual para Python.**

El código que ha sido reutilizado del proyecto CERTIFY para desplegar el DDM hace uso de librerías de Python muy específicas. Por ello, para no instalar en nuestro sistema todas estas librerías, podemos crear un entorno virtual que tenga estrictamente lo necesario para trabajar con el DDM. Esto se puede hacer del siguiente modo:

```
# Asumimos que estamos en el directorio donde tenemos el código fuente del DDM

# Creamos un entorno virtual al que llamamos venv. Si este comando falla, previamente
# habrá que ejecutar: sudo apt install python3.12-venv
python3 -m venv venv

# Activamos el entorno virtual creado
source venv/bin/activate

# Una vez que tenemos el entorno activado, comenzamos a instalar las librerías necesarias
# dentro de este entorno
pip3 install pyangbind
pip3 install PyYAML
pip3 install SQLAlchemy
pip3 install Flask
pip3 install requests
pip3 install bitarray
```

```
pip3 install six
pip3 install dicttoxml
pip3 install xmltodict
pip3 install crcmod
pip3 install hkdf
pip3 install pycryptodome

# Para poder replicar el escenario, podemos crear un fichero requirements que contenga todas
# las dependencias. Así, si se quisiese replicar este entorno, a través de
# pip install -r requirements.txt generaríamos el escenario necesario para trabajar
# con el DDM
pip freeze > requirements.txt
```

- **Paso 2. Obtención de un traductor de MUD a MSPL.**

El código del DDM hace uso de un traductor de MUD a MSPL. Sin embargo, este traductor no se incluyó cuando se proporcionó el DDM, por lo que tratar de ejecutarlo generaba un error. Dado que el desarrollo de un traductor desde cero era una tarea que consumía una gran cantidad de tiempo, hubo que solicitar este código. Destacar que tener este código también era importante para poder entender cómo se había definido el trabajo con la extensión `umu-mspl-list:mspls`.

- **Paso 3. Visualización de la base de datos creada por el DDM.**

El DDM genera una base de datos con la información de los dispositivos IoT que gestiona. Para visualizar este contenido, se puede instalar la siguiente herramienta:

```
sudo apt install sqlitebrowser
```

En relación con esta funcionalidad, es importante tener en cuenta que cada vez que se ejecuta la infraestructura para trabajar con ficheros MUD, se generan los ficheros `DeviceDomainManager.db` y `MUDManager.db` para almacenar la información relacionada con los dispositivos IoT que se registran en el DDM. Por ello, cada vez que se quiera probar el escenario, estos ficheros deben ser borrados. De lo contrario, al detectar que el dispositivo IoT ya está registrado en el DDM, no se hará nada.

- **Paso 4. Solucionar errores del DDM.**

El código del DDM proporcionado tenía varios problemas asociados con el proceso de traducción de un MUD. Estos impedían tanto que el contenido del MUD llegase correctamente hasta el traductor como que el resultado de este se obtuviese correctamente. Para solucionar estos problemas fue necesario hacer tareas de *debug* sobre el código. Con el objetivo de acelerar este proceso, se eliminó la interacción con la Raspberry Pi. En su lugar, se simuló el mensaje que debía enviar el *Bootstrapping-Server* al DDM con la información del dispositivo que estaba realizando el *bootstrapping*. Para ello, a través de la línea de comandos se generó un mensaje POST que tuviese la información necesaria para activar el DDM. En concreto, el mensaje enviado desde una terminal fue el siguiente:

```
curl -X POST http://localhost:4321/bootstrapping -H "Content-Type: application/json" -d '{"uuid": "b8113602", "device": "Linux", "ip_address": "192.168.18.2", "mud-url": "http://example.com/mud-url"}'
```

Este mensaje simulaba la comunicación entre el *Bootstrapping-Server* y el DDM, permitiendo comenzar el proceso de depuración de una forma más rápida, pues podíamos evitar realizar el proceso de *bootstrapping*.

Una vez se consiguió solucionar los problemas del DDM se obtuvo la traducción del MUD a MSPL. El resultado de esta traducción se puede consultar en el anexo **B**.

- **Paso 5. Procesamiento de MSPLs.**

Una vez se tenía la traducción del MUD, el paso final era implementar en el DDM una función capaz de procesar el contenido de los MSPLs para buscar políticas compatibles con el ERA desplegado. Gracias a que estos están en formato Extensible Markup Language (XML) [42], esta tarea se puede hacer de forma sencilla haciendo uso de la librería de Python `xml.etree.ElementTree` [43].

En nuestro caso, para detectar la política de cambio de algoritmo de firma debemos comprobar si alguno de los MSPLs contiene el valor USE-HMAC en el campo `<software-protection-type>`.

Mencionar que, para comprobar el correcto funcionamiento de esta funcionalidad, se utilizaron los MSPLs del anexo **B** para crear un test que comprobase si el código diseñado era capaz de detectar que había que aplicar la política de cambio de algoritmo de firma. Además, se utilizaron otros MSPLs que se tenían de ejemplo para comprobar que en esos casos la necesidad de aplicar la política de cambio de algoritmo de firma no se activaba.

4.4. Prueba de la PoC desarrollada

Tras desplegar todas las entidades definidas en la figura 4.1, el paso final es poner a prueba el escenario desplegado. Para facilitar esta tarea, se creó un *script* que permitiese automatizar el despliegue de todas las entidades que no correspondían a la Raspberry Pi 3 Model B. En concreto, se definió el siguiente *script*⁹:

```
#!/bin/bash

# Activa el entorno virtual que necesita Python para el uso del DDM
source ~/Desktop/TFG-INFO/7-Integrar-MUD/venv/bin/activate

# Lanza el Bootstrapping Server
gnome-terminal -- bash -c "echo '[BOOTSTRAPPING]'; python3 ~/Desktop/TFG-INFO/7-Integrar-MUD/↵
↵ Servidor/bootstrapping_request_manager.py; exec bash"
gnome-terminal -- bash -c "echo '[AAA SERVER]'; ~/Desktop/TFG-INFO/7-Integrar-MUD/Servidor/↵
↵ aaa_server_test; exec bash"

# Lanza la entidad intermediaria que se usa para comunicar el Bootstrapping Server y el DDM
gnome-terminal -- bash -c "echo '[POST SERVER]'; cd ~/Desktop/TFG-INFO/7-Integrar-MUD/Post-Server↵
↵ -ParaConectarTodo/ && gcc post_server.c -o post_server -lmicrohttpd -lcjson -lcurl && ./↵
↵ post_server; exec bash"

# Lanza el MUD Serer
gnome-terminal -- bash -c "echo '[MUD SERVER]'; python3 ~/Desktop/TFG-INFO/7-Integrar-MUD/↵
↵ Proveedor-De-MUDs/mud_server.py; exec bash"

# Lanza el MUD Manager
gnome-terminal -- bash -c "echo '[MUD MANAGER]'; python3 /home/theonlyone_65/Desktop/TFG-INFO/7-↵
↵ Integrar-MUD/DDM/Extended-MUD-Manager/MUDManager.py; exec bash"

# Lanza el DDM
gnome-terminal -- bash -c "echo '[DDM]'; python3 ~/Desktop/TFG-INFO/7-Integrar-MUD/DDM/Device-and↵
↵ -domain-manager/DeviceDomainManager.py; exec bash"
```

Al ejecutar este *script*, se pone en marcha el *Bootstrapping-Server*, *Device and Domain Manager*, MUD Server y MUD Manager de la figura 4.1. Por tanto, iniciando la Raspberry Pi 3 Model B con una SD en la que no se haya completado aún el proceso de *bootstrapping* se puede poner a prueba el funcionamiento del escenario.

En esta sección exponemos los pasos¹⁰ que se deben seguir para realizar una prueba satisfactoria de la PoC desarrollada.

⁹Las rutas definidas en el *script* son dependientes del ordenador en el que se realizó la prueba.

¹⁰Dado que se hace uso de imágenes, en pro de la legibilidad algunas han sido recortadas cuando no aportaban información relevante. Se puede encontrar un vídeo de la prueba en: *ver aquí*.

El primer paso consiste en ejecutar el *script* desarrollado para poner en marcha todas las entidades que no corresponden al dispositivo IoT.

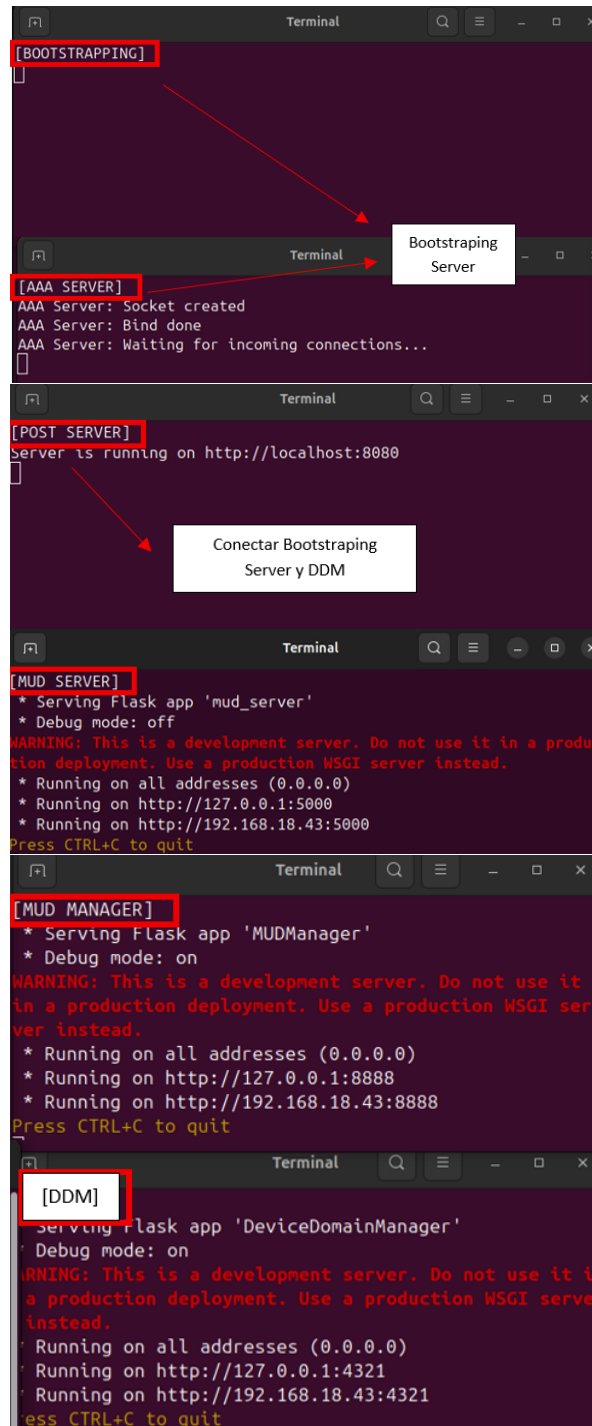


Figura 4.2: PoC: Arranque de las entidades distintas al dispositivo IoT

La figura 4.2 muestra el estado inicial de las entidades *Bootstrapping-Server*¹¹, MUD Server, MUD Manager y *Device and Domain Manager*. Además, aparece una entidad desconocida hasta el momento: *Post-Server*. Esta entidad es la que permite establecer la comunicación entre el *Bootstrapping-Server* y el *Device and Domain Manager*. Corresponde a lo que se definió como servidor “vacío” en la sección 4.3.2.1.

Tras esto, el siguiente paso es iniciar la Raspberry Pi 3 Model B y comenzar el proceso de *bootstrapping*.

```

$ python3 era_testing.py
Certificate to store: empty
Alleged size: 5
Certificate contents: 0x41 75 6d 6d 79
Invoking TA to install PSK
in installation PSK
Invoking TA to store
TA stored successfully
Invoking TA to store
TA stored successfully
Invoking TA to retrieve from secure storage
retrieved cert at retrieve_from_ta: 0x68 74 74 70 73 3a 2f 77 77 77
TA retrieved successfully
Message sent successfully
Create socket
Configure server address structure
Bind socket to port
Listen for incoming connections
Accept a connection
Receive data
Finished receiving
Client: Identity: usera1
Client: EAP-response length: 11
Message sent successfully
Size of received first EAP message: 32
Contents of first 16 bytes of received buffer (first EAP message):
0x01 01 00 20 2f 00 ae 5c 91 36 ce c4 11 de 24 07
EAP-PSK first message: received header:
Code: 1
ID: 1
Length: 2000
Type: 2f
Flags: 0
0xae 5c 91 36 ce c4 11 de 24 07 f2 83 f4 26 4c fa
0xae 5c 91 36 ce c4 11 de 24 07 f2 83 f4 26 4c fa
Invoking TA to install generate random buffer
TA generated random buffer
Invoking TA to sign
TA signed
CHAC: 0xf3 43 13 ea 0f ff 12 f0 90 b0 1c d9 da e0 7a d1
Message sent successfully
Invoking TA to sign
TA signed
Server CHAC: 0xda fe d8 99 de 45 de 42 14 37 64 d5 fc a8 67 e9
Invoking TA to sign
TA signed
Server authenticated successfully
Message sent successfully
Successfully sent fourth EAP message
Invoking TA to derive MSK
TA generated key
Successfully derived MSK
Message sent successfully
Bootstrapping successful

```

Figura 4.3: PoC: Inicio del *bootstrapping*

La figura 4.3¹² muestra que el proceso de *bootstrapping* se ha completado exitosamente. Además, se puede observar que el programa queda en ejecución. Esto se debe a lo que expusimos en la sección 4.3.2, el ERA se pone en funcionamiento una vez se completa el *bootstrapping*.

¹¹Como se puede observar, el *Bootstrapping-Server* está formado por varias entidades.

¹²Las imágenes de la Raspberry están hechas con un teléfono móvil. El motivo de hacer esto es enfatizar que estamos trabajando con un dispositivo IoT. Se podrían haber obtenido haciendo uso de Secure Shell (SSH) [44] tal y como se explica en el anexo D.

Es importante tener en cuenta que la ejecución del proceso de *bootstrapping* provoca que todas las entidades de la figura 4.2 actualicen la información que muestran.

```
[BOOTSTRAPPING]
Connection from ('192.168.18.67', 57010) established.
Received: b'\x01\x02\x00(https://www.example.com/yourmudfile.js
Received hex: 0102002868747470733a2f2f7777772e6578616d706c652e63
f6d2f796f75726d756466696c652e6a736f6e
Message length: 44
Mud Url length: 40
Mud Url: https://www.example.com/yourmudfile.json
Type: 1
Code: 2
Calling high end bootstrapping process...
C program output captured:
AA Manager: Connection established with the device
Server: Construted EAP request identity...
[EAP] request identity:
- Code: 01
- ID: 01
- Length: 7
- Type: 01
AA Manager: Sent EAP request

AA Manager: Received EAP message (length: 11)

theonlyone_65@UbuntuPepe: ~/Desktop/TFG-INFO/7-L...
AAA Server: Encrypted PChannel Data: 22b9b8a3dd98d2fcbef8efae964b
r5ezba3ab1292390ee88c7a80dc70af130b8996e280213d386bf869ac196a34f4
a7449e84cf6589b455d070f0785c115000000000000000000000000000000
- Derived MSK: ac4ee3a550eb3000761dcf5ac10df26e55acc3068f
c6c3b95f1ba36085e350337c288e4c0b2cc93da4c428cd628f63ecdce52037568
d446c9311bb0eddc
- Derived EMSK: 2e065523dfa794574d9de9c03c4d79dbc6c420919
5107dc551f73acb9722c8ac6b58cb2af575ab24886926a4b256eef85949402feef
ecf7baeaa0e38b24a6
AAA Server: Constructing EAP Message:
[EAP] Access-Success:
- Code: 03
- ID: 00
- Length: 5
- Type: 2f
AAA Server: [EAP] Access Accept sent successfully (length: 5)
AAA Server: [EAP] Authentication finished!

Terminal
[POST SERVER]
Server is running on http://localhost:8080
Initialized POST data handler
Received POST data:
{
  "uuid": "a96a18fc-a9b3-404d-88e0-2448a9466d3e",
  "device": "high_end_device-192.168.18.67",
  "ip_address": "192.168.18.67",
  "mud-url": "https://www.example.com/yourmudfile.js
on"
}
Device stored: a96a18fc-a9b3-404d-88e0-2448a9466d3e -> https://
www.example.com/yourmudfile.json
```

Figura 4.4: PoC: Consecuencias del *bootstrapping*


```

[MUD SERVER]
* Serving Flask app 'mud_server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.18.43:5000
Press CTRL+C to quit
127.0.0.1 - - [09/Jun/2025 18:17:46] "GET /mud/prueba HTTP/1.1"
200 -

ftware-protection", "rule-set-configuration": {"confi
guration-rule": {"configuration-action": {"software-p
rotection-action": {"software-protection-type": "USE-
HMAC"}}, "name": "change_signature_rule"}, {"name": "c
hange_signature_rule_set"}}, {"name": "mspl_change_sig
nature", "type": "software-mspl-type"}, {""configurati
on": {"capability": "Filtering_L4", "rule-set-configu
ration": {"configuration-rule": [{"configuration-acti
on": {"filteringAction": {"filtering-action-type": "A
LLOW"}}, "configuration-condition": {"filtering-confi
guration-condition": {"packet-filter-condition": {"de
stination_dnsname": "raspberrypi-3-modelB-TFG.umu",
"destination_port": "any", "direction": "INBOUND", "p
rotocol_type": "6", "source_dnsname": "bootstrapping-
server.umu", "source_port": "any"}}}], "name": "filter
ing_rule"}], "name": "filtering_rule_set"}}, {"name":
"mspl_filtering", "type": "ipv4-mspl-type"}]}], "devi
ce_idenfifier": "a96a18fc-a9b3-404d-88e0-2448a9466d3e
"}
127.0.0.1 - - [09/Jun/2025 18:17:46] "POST /mud HTTP/
1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [09/Jun/2025 18:17:46] "P
OST /mud HTTP/1.1" 200 -
[

ing-action-type></filteringAction></configuration-acti
on><configuration-condition><filtering-configuration-c
ondition><packet-filter-condition><destination_dnsname
>raspberrypi-3-modelB-TFG.umu</destination_dnsname><d
estination_port>any</destination_port><direction>INBOU
ND</direction><protocol_type>6</protocol_type><source_
dnsname>bootstrapping-server.umu</source_dnsname><sour
ce_port>any</source_port></packet-filter-condition></f
iltering-configuration-condition></configuration-condi
tion><name>filtering_rule</name></configuration-rule><
name>filtering_rule_set</name></rule-set-configuration
></configuration><name>mspl_filtering</name><type>ipv4
-mspl-type</type></mspl>
-----
-----
Press ENTER if you want to reconfigure the device (sig

```

Figura 4.5: PoC: Consecuencias del *bootstrapping* (continuación)

Las figuras 4.4 y 4.5 buscan mostrar que el estado de todas las entidades que aparecían en la figura 4.2 se ha visto actualizado. Sin embargo, tratar de comprender qué ha ocurrido con estas imágenes sería imposible, pues ni siquiera se puede ver toda la información que han generado. Por ello, vamos a mostrar el estado de cada una de estas entidades por separado.

Comenzamos entonces con las entidades relacionadas con el *Bootstrapping-Server*. En concreto, la figura 4.2 mostraba que este se dividía en dos entidades: *BOOTSTRAPPING* y *AAA SERVER*. Estas corresponden a las entidades que se definieron en la figura 2.1, siendo *BOOTSTRAPPING* el controlador central y *AAA SERVER* la infraestructura AAA.

La figura 4.6 muestra la información generada por la entidad *BOOTSTRAPPING*. En ella, podemos confirmar que actúa como el controlador central para realizar el *bootstrapping*, pues es la entidad que actúa como intermediario entre el dispositivo IoT y el *AAA SERVER*. En la figura aparece un registro sobre los principales eventos que ocurren mientras se completa el *bootstrapping*. Entre estos destaca la redirección de mensajes EAP entre el dispositivo IoT y el *AAA SERVER*, y el envío de información hacia el *Device and Domain Manager* una vez se ha completado la autenticación del dispositivo. Es interesante destacar que entre la información que se envía al *Device and Domain Manager* se incluye el MUD URL que se debería usar para la configuración de políticas de seguridad en el dispositivo. Como se mencionó en la sección 4.3.2.2, la URL que se puede observar no tiene un significado real, es una cadena de prueba que permite simular el uso de un MUD URL.

```
[BOOTSTRAPPING]
Connection from ('192.168.18.67', 57010) established.
Received: b'\x01\x02\x00(https://www.example.com/yourmudfile.json'
Received hex: 0102002868747470733a2f2f777772e6578616d706c652e636f6d2f796f75726d756466696c652e6a736f6e
Message length: 44
Mud Url length: 40
Mud Url: https://www.example.com/yourmudfile.json
Type: 1
Code: 2
Calling high end bootstrapping process...
C program output captured:
AA Manager: Connection established with the device
Server: Construted EAP request identity...
[EAP] request identity:
- Code: 01
- ID: 01
- Length: 7
- Type: 01
AA Manager: Sent EAP request

AA Manager: Received EAP message (length: 11)
AA Manager: Redirecting message to AAA Server
AA Manager: [AAA] Socket created
AA Manager: [AAA] Connected to AAA server 127.0.0.1:1813
AA Manager: [AAA] EAP-Message sent, waiting for response...
AA Manager: [AAA] Response received from AAA server
AA Manager: [EAP] Sending message of length 32:
01 01 00 20 2F 00 AE 5C 91 36 CE C4 11 DE 24 87 F2 83 F4 26 4E FA 61 61 61 2D 73 65 72 76 65 72
AA Manager: [EAP] Request forwarded successfully to client

AA Manager: Received EAP message (length: 60)
AA Manager: Redirecting message to AAA Server
AA Manager: [AAA] Socket created
AA Manager: [AAA] Connected to AAA server 127.0.0.1:1813
AA Manager: [AAA] EAP-Message sent, waiting for response...
AA Manager: [AAA] Response received from AAA server
AA Manager: [EAP] Sending message of length 102:
01 00 00 66 2F 80 AE 5C 91 36 CE C4 11 DE 24 87 F2 83 F4 26 4E FA 0A FE D8 99 DE 45 DE 42 14 37 64 D5 FC
AA Manager: [EAP] Request forwarded successfully to client

AA Manager: Received EAP message (length: 102)
AA Manager: Redirecting message to AAA Server
AA Manager: [AAA] Socket created
AA Manager: [AAA] Connected to AAA server 127.0.0.1:1813
AA Manager: [AAA] EAP-Message sent, waiting for response...
AA Manager: [AAA] Response received from AAA server
AA Manager: [EAP] Authentication successful:
[EAP] Success:
- Code: 03
- ID: 00
- Length: 5
- Type: 2f
AA Server: [EAP] Authentication finished
AA Server: Sending data to DDM
Generated UUID: a96a18fc-a9b3-404d-88e0-2448a9466d3e
Payload:
{
  "uuid": "a96a18fc-a9b3-404d-88e0-2448a9466d3e",
  "device": "high_end_device-192.168.18.67",
  "ip_address": "192.168.18.67",
  "mud-url": "https://www.example.com/yourmudfile.json"
}
Data processed successfully
POST request successful
AA Manager: [EAP] Request forwarded successfully to client
AA Manager: [EAP] Received 'OK' response from client
AA Manager: [EAP] Authentication finished!
Authentication was successful.
```

Figura 4.6: PoC: Consecuencias del *bootstrapping* en el controlador central

```

[AAA SERVER]
AAA Server: Socket Created
AAA Server: Bind done
AAA Server: Waiting for incoming connections...
AAA Server: Connection accepted from client
AAA Server: Received EAP message (length: 11)
AAA Server: Received Access Request (EAP-Response/ID):
[EAP] Access-Request:
- Code: 02
- ID: 01
- Length: 11
- Type: 01
AAA Server: [EAP] Identity received: user1
AAA Server: [EAP] Identity validation successful
- RAND_S generated for RRF Request
AAA Server: Constructing EAP Challenge:
[EAP] Access-Challenge - EAP PSK 1:
- Code: 01
- ID: 01
- Length: 32
- Type: 2f
- Flags: 00
- Rand_s: ae5c9136cec411de2487f283f4264efa
- ID_s: aaa-server
AAA Server: [EAP] Sending message of length 32:
01 01 00 20 2f 00 ae 5c 91 36 ce c4 11 de 24 87 f2 83 f4 26 4e fa 61 61 61 20 73 65 72 76 65 72
AAA Server: [EAP] Access Challenge (EAP PSK 1) sent successfully (length: 32)
AAA Server: No message received or receive failed (recv_len: 0)
AAA Server: Client disconnected or error in communication
AAA Server: Connection accepted from client
AAA Server: Received EAP message (length: 60)
AAA Server: Received Access Challenge Response (EAP PSK 2):
AAA Server: [EAP] Received message of length 60:
02 00 3c 00 2f 40 ae 5c 91 36 ce c4 11 de 24 87 f2 83 f4 26 4e fa 7b db 38 c6 a7 bc ab fc 60 23
[EAP] Access-Request - EAP PSK 2:
- Code: 02
- ID: 00
- Length: 15360
- Type: 2f
- Flags: 40
- Rand_s: ae5c9136cec411de2487f283f4264efa
- Rand_p: 7bdb38c6a7bcabfc6023ee54fd4a8f04
- MAC_C: f34313ea0fff12f890b01ed9dae07ad1
- ID_p: user1
AAA Server: Checking integrity of MAC_P...
- Derived AK: Scf1917625b1165fd0a955f5767cbea0
Computed MAC_P: f34313ea0fff12f890b01ed9dae07ad1
- MAC_P validation successful
AAA Server: Starting first Key derivation
- Derived KDK: c905db040bc16375bfbdfa800ce4aea7
- Derived TEK: 8cff7828a1950f77c5fe3c6d866fedba
AAA Server: ID_S Generated: 6161612d736572766572
AAA Server: Computed MAC_S: 0afed899de45de42143764d5fca867e9
AAA Server: Added ID_S to PChannel Data: 0a6161612d736572766572
AAA Server: Added TEK to PChannel Data: 8cff7828a1950f77c5fe3c6d866fedba
AAA Server: Added Nonce to PChannel Data: 67a19c3fd1d60d3e
AAA Server: Final PChannel Data before Encryption: 0a6161612d7365727665728cff7828a1950f77c5fe3c
AAA Server: Encryption context initialized successfully.
AAA Server: Generated IV: 22b9b8a3dd98d2fcbef8efae964b74f5
AAA Server: Final Encrypted Data: e2ba3ab1292390ee88c7a80dc70af130b8996e20011d386bf869ac196a34
AAA Server: Full EAP-Response Message: 01000662f80ae5c9136cec411de2487f283f4264efa0afed899de45
AAA Server: Total Message Length: 102
AAA Server: EAP-Response/Success message sent to client
AAA Server: No message received or receive failed (recv_len: 0)
AAA Server: Client disconnected or error in communication
AAA Server: Connection accepted from client
AAA Server: Received EAP message (length: 102)
AAA Server: Received Access Challenge Response (EAP PSK 4):
[EAP] Access-Request - EAP PSK 4:
- Code: 02
- ID: 00
- Length: 102
- Type: 2f
- Flags: c0
- Rand_s: ae5c9136cec411de2487f283f4264efa
AAA Server: Encrypted PChannel Data: 22b9b8a3dd98d2fcbef8efae964b74f5e2ba3ab1292390ee88c7a80dc7
- Derived MSK: c4ee3a550eb3000761dcf5ac10df26e55acc3068f80c6c3b95f1ba3608e350337c288e
- Derived EMSK: 2e065523dfa794574d9de9c03c4d79dbc6c4209192b5107dc551f73acb9722c8ac6b58c
AAA Server: Constructing EAP Message:
[EAP] Access-Success:
- Code: 03
- ID: 00
- Length: 5
- Type: 2f
AAA Server: [EAP] Access Accept sent successfully (length: 5)
AAA Server: [EAP] Authentication finished!

```

Figura 4.7: PoC: Consecuencias del *bootstrapping* en el AAA server

En la figura 4.7 podemos ver las acciones que se llevan a cabo en el AAA SERVER durante el proceso de *bootstrapping*. La figura muestra con detalle la autenticación EAP-PSK descrita en la sección 2.1.1.2. Para indicar el momento en el que comienza cada uno de los pasos descritos en dicha sección se ha remarcado el mensaje que da comienzo a cada etapa¹³. Además, es interesante destacar que tal y como se expuso en dicha sección el proceso de autenticación EAP-PSK concluye con la obtención de material criptográfico. En la figura podemos apreciar esto al final de la imagen, donde podemos ver que se derivan dos claves criptográficas definidas para este protocolo: MSK y Extended Master Session Key (EMSK)

El siguiente paso es mostrar cómo se ha visto actualizada la información que muestra el intermediario entre el *Bootstrapping-Server* y el *Device and Domain Manager*, el *Post-Server*. Esto se puede apreciar en la figura 4.8. En ella podemos ver que ha recibido la información del dispositivo que acaba de realizar el *bootstrapping*. Entre esta información tenemos la dirección IP del dispositivo IoT, un MUD URL y dos identificadores del mismo.

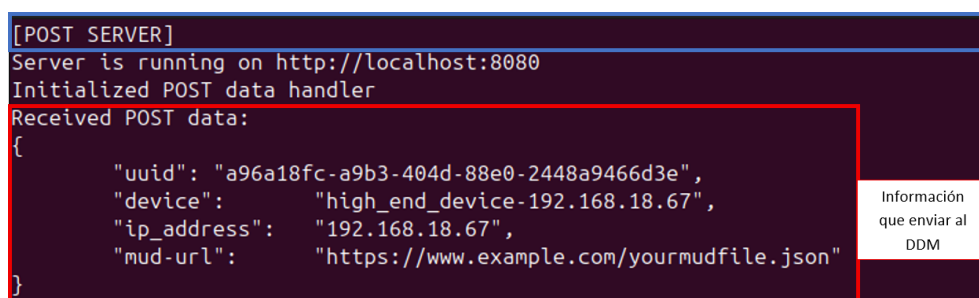


Figura 4.8: PoC: Consecuencias del *bootstrapping* en el *Post-Server*

El hecho de que el *Post-Server* haya recibido una actualización implica que esta información sea reenviada al *Device and Domain Manager*, provocando que se recupere el fichero MUD asociado al dispositivo que ha realizado el *bootstrapping*. Por tanto, las entidades MUD Manager, MUD Server y *Device and Domain Manager* actualizan la información que muestran, indicando cómo se recupera el fichero MUD y se realiza su traducción a lenguaje MSPL. Esto se puede observar en las figuras 4.9, 4.11 y 4.10.

¹³Destacar que el mensaje “AAA Server: EAP-Response/Success message sent to client” marcado en la imagen es un error del desarrollador del código. El mensaje que se está enviando en realidad es el mensaje EAP PSK 3.

```
[DDM]
* Serving Flask app 'DeviceDomainManager'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:4321
* Running on http://192.168.18.43:4321
Press CTRL+C to quit
```

Mensaje recibido del Post Server

```
{'uuid': 'a96a18fc-a9b3-404d-88e0-2448a9466d3e', 'device': 'high end device-192.168.18.67', 'io address': '192.168.18.43', 'mudurl': 'https://www.example.com/yourmudfile.json', 'mudfile': {'ietf-access-control-list:acis': [{'acl': {'protocol': 6}, 'tcp': {'destination-port': {'operator': 'eq', 'port': 443}}, 'ietf-mud:direction-in': {'ace': [{'actions': {'forwarding': 'accept'}, 'matches': {'ipv4': {'protocol': 6, 'source-ipv4-network': '192.168.18.0/24'}}, 'name': 'inbound-stricted-communication', 'type': 'ipv4-acl-type'}}]}, 'ietf-mud:direction-out': {'outbound-stricted-communication'}}], 'is-supported': True, 'last-update': '2024-12-18T21:07:00Z', 'eminfo': 'raspberrypi pi 3 model B current MUD File', 'to-device-policy': {'access-lists': {'access-list-name': 'ALLOW', 'rule-set-configuration': {'configuration-rule': {'configuration-action': {'software-protection': {'name': 'mspl_change_signature', 'type': 'software-mspl-type'}, 'configuration': {'capability': 'Firmware-Integrity-Check', 'type': 'firmware-integrity-check-type'}, 'filtering-condition': {'packet-filter-condition': {'packet-filter-condition': {'protocol': 6, 'source-dnsname': 'bootstrapping-server.umu', 'source_port': 'any'}}}, 'name': 'filtering_rule-192.168.18.43-404d-88e0-2448a9466d3e'}}}}}
```

MUD File Traducción a MSPL

```
b'<?xml version="1.0" encoding="UTF-8" ?><mspl><name type="str">mspl_example</name><configuration type="dict"><default-action type="str">deny</default-action><configuration-rule type="dict"><filtering-action type="dict"><filtering-action type="str">allow/filtering-action type="dict"><packet-filter-condition type="dict"><destination-address type="str">192.168.100.0/24/</destination-address></packet-filter-condition></filtering-configuration-condition></configuration-condition></external-data><configuration-action type="dict"><filtering-action type="dict"><filtering-action type="dict"><packet-filter-condition type="dict"><source-address type="str">6/</source-address></packet-filter-condition></filtering-configuration-condition></configuration-condition></mspl>'<?xml version="1.0" encoding="utf-8" ?><mspl><configuration><capability>software-protection</capability><software-protection type="USE-HMAC"><software-protection type="dict"><software-protection-action type="dict"><name>charMSPLChangeSignature</name><type>software-mspl-type</type></mspl><?xml version="1.0" encoding="utf-8" ?><mspl><configuration><filteringAction><filtering-action type=ALLOW</filtering-action type=</filteringAction></configuration></mspl>
```

Detecta política de cambio de firma.
Espera confirmación para aplicarla

Figura 4.9: PoC: Consecuencias del *bootstrapping* en el DDM

```
[MUD SERVER]
* Serving Flask app 'mud_server'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production de
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.18.43:5000
Press CTRL+C to quit
127.0.0.1 - - [09/Jun/2025 18:17:46] "GET /mud/prueba HTTP/1.1" 200 -
```

Figura 4.10: PoC: Consecuencias del *bootstrapping* en el MUD Server


```
[MUD MANAGER]
* Serving Flask app 'MUDManager'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSG
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8888
* Running on http://192.168.18.43:8888
Press CTRL+C to quit

{'mudurl': 'https://www.example.com/yourmudfile.json', 'device identifier': 'a96a18fc-a9b3-404d-88e0-2
{"mudurl": "https://www.example.com/yourmudfile.json", "mudfile": {"ietf-access-control-list:acls": {
{"destination-ipv4-network": "192.168.100.0/24", "protocol": 6}, "tcp": {"destination-port": {"operat
utput-rule"}}}, "name": "outbound-stricted-communication", "type": "ipv4-acl-type"}, {"aces": {"ace":
ipv4-network": "192.168.100.0/24", "tcp": {"ietf-mud:direction-initiated": "to-device", "source-port
icted-communication", "type": "ipv4-acl-type"}}}, "ietf-mud:mud": {"cache-validity": 48, "extensions"
"name": "outbound-stricted-communication"}}}], "is-supported": true, "last-update": "2024-12-18T21:07
om/yourmudfile.json", "mud-version": 1, "systeminfo": "raspberry pi 3 model B current MUD File", "to-
cation"}}}], "umu-mspl-list:mspls": {"mspl": [{"configuration": {"capability": "software-protection"
re-protection-action": {"software-protection-type": "USE-HMAC"}, "name": "change_signature_rule"}, "
ware-mspl-type"}, {"configuration": {"capability": "Filtering_L4", "rule-set-configuration": {"config
": "ALLOW"}, "configuration-condition": {"filtering-configuration-condition": {"packet-filter-condit
any", "direction": "INBOUND", "protocol_type": "6", "source_dnsname": "bootstrapping-server.umu", "so
ame": "mspl filtering", "type": "ipv4-mspl-type"}}}], "device identifier": "a96a18fc-a9b3-404d-88e0-2
127.0.0.1 - - [09/Jun/2025 18:17:46] "POST /mud HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [09/Jun/2025 18:17:46] "POST /mud HTTP/1.1" 200 -
```

Figura 4.11: PoC: Consecuencias del *bootstrapping* en el MUD Manager

Además, el *Device and Domain Manager* registra en su base de datos al dispositivo que ha realizado el proceso de *bootstrapping*. Esto se puede apreciar en la figura C.1 del anexo C¹⁴.

Una vez completado el proceso de *bootstrapping*, si volvemos a las figuras 4.3 y 4.9, podemos ver que ambas han quedado listas para realizar un proceso de reconfiguración en el dispositivo IoT. En concreto, podemos aplicar la política de cambio de algoritmo de firma. Esto es lo que se puede observar en las figuras 4.12 y 4.13.

¹⁴Esta foto ha sido añadida en el anexo C por falta de legibilidad en formato vertical.

```
[DWM]
* Serving Flask app 'DeviceDomainManager'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:4321
* Running on http://192.168.18.43:4321
Press CTRL+C to quit
{'uuid': 'a96a18fc-a9b3-404d-88e0-2448a9466d3e', 'device': 'high_end_device-192.168.18.67', 'ip_address': '192.168.18.67', 'mudurl': 'https://www.example.com/yourmudfile.json', 'mudfile': {'ietf-access-control-list:acls': {'acl': [{'aces': {'protocol': 6}, 'tcp': {'destination-port': {'operator': 'eq', 'port': 443}, 'ietf-mud:direction-initiated': 'from-device'}, 'ace': [{'actions': 'forwarding', 'accept'}], 'matches': {'ipv4': {'protocol': 6, 'source-ipv4-network': '192.168.100.0/24', 'name': 'input-rule'}}], 'name': 'inbound-stricted-communication', 'type': 'ipv4-acl-type'}}], 'ietf-mud:mud': {'cache': {'name': 'outbound-stricted-communication'}}}], 'is-supported': True, 'last-update': '2024-12-18T21:07:00+01:00', 'model-info': 'raspberrypi 3 model B current MUD File', 'to-device-policy': {'access-lists': {'access-list': [{'name': 'inbound-striction', 'rule-set-configuration': {'configuration-rule': {'configuration-action': {'software-protection-action': {'signature': 'mspl_change_signature', 'type': 'software-mspl-type'}, 'configuration': {'capability': 'Filtering_L4', 'rule-set': 'ALLOW'}}], 'configuration-condition': {'filtering-configuration-condition': {'packet-filter-condition': {'destination-address': '192.168.100.0/24', 'source-dnsname': 'bootstrapping-server.umu', 'source_port': 'any'}}], 'name': 'filtering_rule'}}], 'name': 'filtering_rule'}}]}
name: inbound-stricted-communication
name: outbound-stricted-communication
-----
b'<?xml version="1.0" encoding="UTF-8" ?><mspl><name type="str">mspl_example</name><configuration type="dict"><capability type="dict"><default-action type="str">deny</default-action><configuration-rule type="dict"><output-rule type="dict"><filtering-action type="dict"><filtering-action-type type="str">allow</filtering-action-type></filtering-action><packet-filter-condition type="dict"><destination-address type="str">192.168.100.0/24</destination-address><destination-port type="str">443</destination-port></packet-filter-condition></filtering-action></configuration-rule></configuration></capability></dict></mspl><?xml version="1.0" encoding="utf-8" ?><mspl><configuration><capability>software-protection</capability><rule-set>USE-HMAC</software-protection-type></software-protection-action></configuration-action><name>change_signature_rule</mspl_change_signature/name><type>software-mspl-type</type></mspl><?xml version="1.0" encoding="utf-8" ?><mspl><configuration><filteringAction><filtering-action-type>ALLOW</filtering-action-type></filteringAction></configuration-action><configurationCondition><packet-filter-condition type="dict"><source-address type="str">192.168.100.0/24</source-address></packet-filter-condition></filteringConfigurationCondition></configuration-condition></configuration></mspl></mspl>'
-----
Press ENTER if you want to reconfigure the device (signature)
-----
Connected to 192.168.18.67:5025
Received: b'OK'
```

Figura 4.12: PoC: DDM tras aplicar el cambio de algoritmo de firma

```

$ python3 era_testing.py
Certificate to store: Dunny
Alleged size: 5
Certificate contents: 0x41 75 6d 6d 79
Invoking TA to install PSK
Create socket
Configure server address structure
Bind socket to port
Listen for incoming connections
Accept a connection
Receive data
Finished receiving
Client: Identity: useral
Client: EAP-Response length: 11
Message sent successfully
Size of received first EAP message: 32
Contents of first 16 bytes of received buffer (first EAP message):
0x01 01 00 20 2f 00 ae 5c 91 36 ce c4 11 de 24 87
EAP-PSK first message: received header:
Code: 1
ID: 1
Length: 2000
Type: 2f
Flags: 0
0xae 5c 91 36 ce c4 11 de 24 87 f2 83 f4 26 4e fa
0xae 5c 91 36 ce c4 11 de 24 87 f2 83 f4 26 4e fa
Invoking TA to install generate random buffer
TA generated random buffer
Invoking TA to sign
TA signed
CMAC: 0xf3 43 13 ea 0f ff 12 f8 90 b0 1c d9 da e0 7a d1
Message sent successfully
Invoking TA to sign
TA signed
Server CMAC: 0x0a fc d8 99 dc 45 de 42 14 37 64 d5 fc a8 67 e9
Invoking TA to sign
TA signed
Server authenticated successfully
Message sent successfully
Successfully sent fourth EAP message
Invoking TA to derive MSK
TA generated key
Successfully derived MSK
Message sent successfully

Bootstrapping successful
Connection from ('192.168.18.43', 51716) ESTABLISHED.
Received: b'\x02\x02\x00\x02\x01\x00'
Received hex: 020200020100
Message length: 6
bin length: 2
bin: b'\x01\x00'
Type: 2
Calling TA for reconfiguration
Opcode: b'\x01'
Data: b'\x00'
Invoking TA to change signature algorithm: 0
TA changed algorithm successfully
Reconfiguration successful

```

Figura 4.13: PoC: Raspberry tras aplicar el cambio de algoritmo de firma

De esta forma se pone fin a la prueba del escenario desplegado para este trabajo.

5. Evaluación de la PoC desarrollada

En la sección 4 hemos presentado la PoC desarrollada para este trabajo. En concreto, hemos realizado tres acciones: definir los elementos que la componen, explicar cómo se ha desplegado cada uno de esos elementos y finalmente, proporcionar un ejemplo de su correcto funcionamiento.

En esta sección buscamos completar la presentación de la PoC proporcionando una evaluación del sistema desplegado. En concreto, proporcionaremos medidas del tiempo de ejecución de las principales actividades que se realizan en el escenario propuesto. De este modo, estas pueden servir como base para futuras mejoras, pues las posibles evoluciones de la PoC deberían tener entre sus objetivos reducir el tiempo de ejecución del escenario. Además, las medidas que se presentarán en esta sección pueden ser de utilidad para comparar cómo de buenas, en términos de rendimiento, son otras alternativas para la gestión de dispositivos IoT.

Para realizar esta evaluación, el primer paso es tener claro el escenario que se ha desarrollado en este trabajo, pues solo así se puede valorar qué actividades merecen la pena ser evaluadas. Para ello, en la figura 5.1 se proporciona un diagrama de flujo que resume la PoC desarrollada. Como se puede observar en dicha figura, el escenario sigue un flujo secuencial, donde el primer paso es realizar el proceso de *bootstrapping* CoAP-EAP. Esa etapa concluye con el envío al DDM de información del dispositivo que ha completado el *bootstrapping* (pasos 2 y 3 del diagrama), permitiendo así que pueda comenzar la fase de reconfiguración del dispositivo. Una vez el DDM tiene la información del dispositivo que ha completado el *bootstrapping*, puede proceder a recuperar el fichero MUD que tiene asociado. Para ello, interactúa con el MUD Server a través del MUD Manager (paso 4 del diagrama). Así, tras recibir el MUD file (paso 5 del diagrama), puede procesarlo y mandar una solicitud de reconfiguración al dispositivo IoT si lo considera pertinente (paso 6 del diagrama). Dicha solicitud es procesada por el ERA del dispositivo y si se considera válida, es aplicada (paso 7 del diagrama).

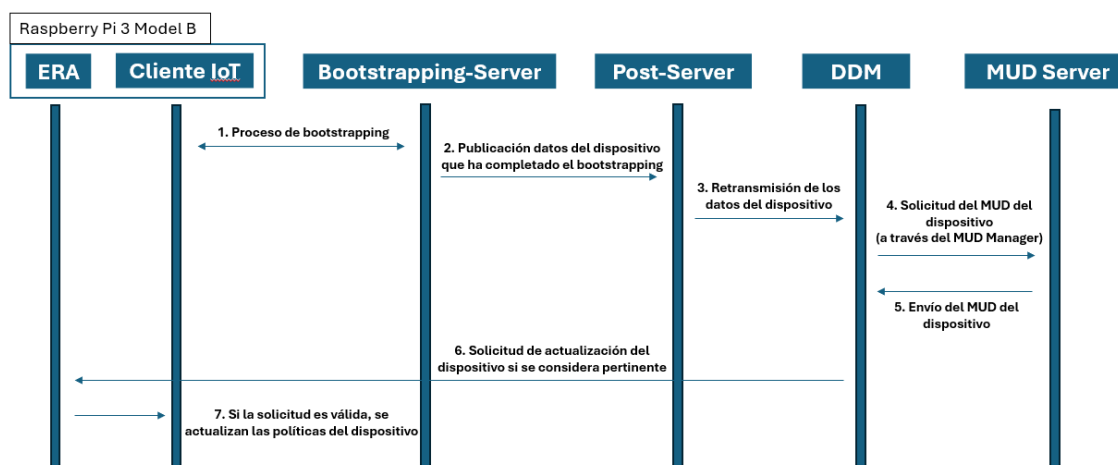


Figura 5.1: Flujo del escenario IoT desplegado

Por tanto, el escenario planteado en este trabajo divide su funcionalidad en dos fases: el proceso de *bootstrapping* (paso 1 del diagrama) y el proceso de reconfiguración (pasos del 2 al 7 del diagrama). De este modo, podemos realizar una primera evaluación de los tiempos que se tarda en ejecutar cada una de estas fases.

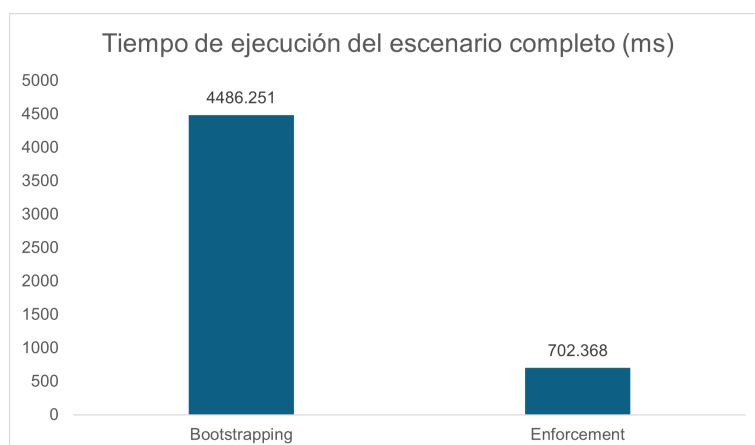


Figura 5.2: Tiempos de ejecución del *bootstrapping* y del *enforcement* en la PoC

La figura 5.2 muestra el tiempo que se tarda en ejecutar cada una de las fases¹ mencionadas. Como se puede observar, completar el *bootstrapping* es aproximadamente seis veces más lento que realizar una reconfiguración del dispositivo. Este resultado es lógico, pues el *bootstrapping* es un proceso complejo que debe preparar al dispositivo para unirse a la red IoT de un modo seguro. En esta fase se deben realizar tareas de auten-

¹En la imagen a la fase de reconfiguración se le ha llamado *enforcement*. Esto se hace para reforzar la idea de que estamos aplicando políticas de seguridad definidas en ficheros MUD.

ticación, autorización y obtención de claves criptográficas. Por tanto, esta etapa tiene una gran relevancia en el correcto funcionamiento del dispositivo durante el resto de su vida útil, por lo que es normal que tarde mucho más que realizar una reconfiguración una vez que el dispositivo ya ha sido preparado para poder hacer esto. Cabe destacar que, aunque el *bootstrapping* se puede considerar un proceso lento, solo se debe realizar una vez en todo el ciclo de vida del dispositivo, cuando este arranca. Por tanto, ocupar un periodo de tiempo considerable no es preocupante si esto garantiza que el dispositivo pueda operar y recibir actualizaciones de forma segura mientras se mantenga activo.

Es interesante destacar que en esta evaluación se ha comparado el tiempo de ejecución del proceso de *bootstrapping* con el de una reconfiguración que implica la aplicación de una única política de seguridad, el cambio de algoritmo de firma. Como sería de esperar, a mayor número de políticas a aplicar, mayor podría ser el tiempo requerido para realizar el proceso de reconfiguración. El estudio de la escalabilidad de dicho proceso ha quedado fuera del alcance de este trabajo. No obstante, en mejoras futuras, sería conveniente realizar evaluaciones que involucren aplicar un número mayor de políticas sobre un mismo dispositivo. De este modo, se podrá obtener una comparativa más precisa entre el tiempo requerido para completar el proceso de *bootstrapping* y el necesario para realizar una reconfiguración.

Finalmente, dado que el proceso de reconfiguración incluye los pasos del 2 al 7 de la figura 5.1, es interesante desglosar el tiempo de ejecución de la fase de reconfiguración que muestra la figura 5.2.

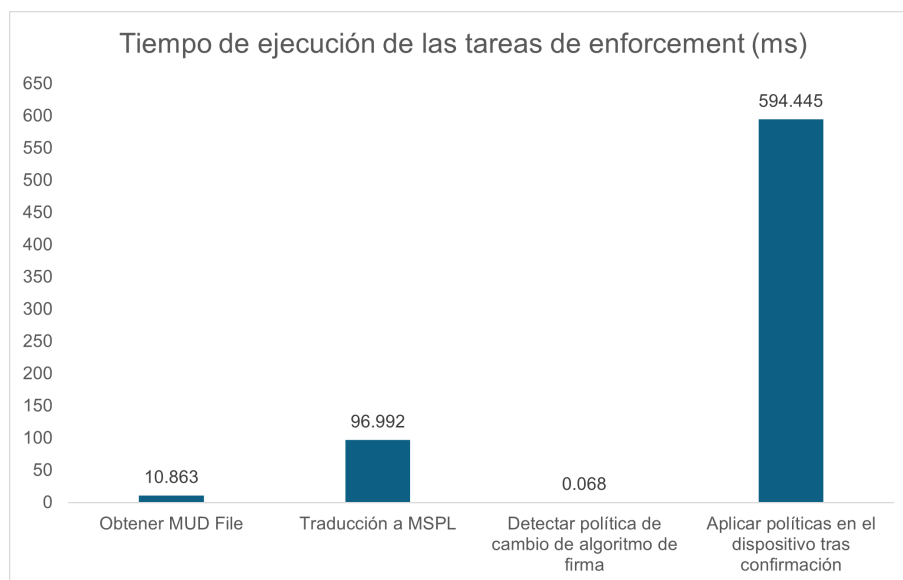


Figura 5.3: Desglose de los tiempos de ejecución del *enforcement* en la PoC

El desglose mencionado se puede observar en la figura **5.3**. Es importante hacer ciertas aclaraciones sobre esta para poder entenderlo.

En primer lugar, hemos llamado “Obtener MUD File” a los pasos 2, 3, 4 y 5 del diagrama de la figura **5.1**. El motivo de hacer esto es que como se expone en la sección **6**, una limitación de esta PoC es que todas las entidades del diagrama que no corresponden al dispositivo IoT están desplegadas sobre un mismo equipo. Por tanto, realizar mediciones separadas de las comunicaciones entre *Bootstrapping-Server* y *Post-Server* (paso 2 del diagrama), *Post-Server* y DDM (paso 3 del diagrama), y DDM y MUD Server (pasos 4 y 5 del diagrama), producen resultados despreciables. Por este motivo, se ha decidido agrupar ese proceso en una única medición.

Tras esto, hemos realizado una medición llamada “Traducción a MSPL”. Esta evaluación no se corresponde a ningún paso del diagrama de la figura **5.1**. Sin embargo, creemos que es importante tener controlado el tiempo que se tarda en traducir el fichero MUD definido en el anexo **A**. Del mismo modo, hemos añadido otra medición que no aparece en el diagrama: “Detectar política de cambio de algoritmo de firma”. El objetivo de esta es detectar cuánto tardamos en procesar el MSPL para detectar si hay que realizar una reconfiguración del dispositivo o no. En nuestro caso, dado que nuestra política de reconfiguración, como se puede observar en el anexo **B**, es muy fácil de detectar, esta tarea se realiza de forma prácticamente inmediata.

Por último, podemos ver la medición “Aplicar políticas en el dispositivo tras confirmación”. Esta etapa supone el grueso de la reconfiguración y es realizada una vez el administrador del DDM decide aplicarla (pasos 6 y 7 del diagrama de la figura **5.1**). Al igual que ocurre con el proceso de *bootstrapping*, es lógico que esta etapa sea la que más tiempo ocupa dentro de la reconfiguración, pues las operaciones que implica son críticas para el dispositivo IoT, por lo que el ERA que tiene desplegado debe hacer uso del TEE para garantizar la seguridad.

Con esto ponemos fin al proceso de evaluación de la PoC que se propone en este trabajo. Estas mediciones pueden servir de base para verificar que evoluciones futuras del escenario IoT planteado son mejores en términos de rendimiento.

6. Conclusiones y vías futuras

El trabajo presentado propone una PoC basada en el proyecto CERTIFY que está siendo desarrollado en la Universidad de Murcia. Dicho proyecto busca proporcionar una solución que permita gestionar de forma segura los largos ciclos de vida de los dispositivos IoT. Este trabajo permite realizar una primera aproximación a los problemas asociados a esta gestión y cómo poder solucionarlos basándonos en la propuesta de CERTIFY. En concreto, el trabajo desarrollado proporciona una comprensión del proceso de *bootstrapping* que se debe realizar en cualquier dispositivo IoT cuando este quiere unirse por primera vez a una red IoT. Para ello, se ha estudiado el modelo CoAP-EAP, el cual ha sido publicado como RFC en el IETF, lo que refuerza el uso de este modelo para realizar la etapa de *bootstrapping*. En este trabajo también se realiza una introducción a cómo gestionar los dispositivos IoT una vez han completado el proceso de *bootstrapping* y pasan a la siguiente fase de su ciclo de vida, operación. Para ello, se introduce el concepto de MUD, siendo clave su gestión para poder proteger a los dispositivos durante toda su vida útil, mediante la aplicación de las políticas que en este se definan.

De este modo, la PoC presentada ofrece un escenario capaz de administrar un determinado dispositivo IoT, una Raspberry Pi 3 Model B. Aunque la gestión que se ofrece respecto a todo su ciclo de vida es limitada, ofrece un buen punto de partida para familiarizarse con los conceptos y herramientas que permiten profundizar en la gestión de dispositivos IoT que propone CERTIFY.

Uno de los aportes más relevantes de este trabajo ha sido la integración práctica de varias de las componentes desarrolladas para CERTIFY, que hasta el momento no habían sido probadas de forma conjunta. Esta integración ha permitido identificar incompatibilidades y errores no detectados previamente, contribuyendo así directamente a la evolución del proyecto. Entre los hallazgos más significativos se incluyen:

- Detectar problemas en el proceso de *bootstrapping* que se estaba desarrollando. Las personas a cargo de las implementaciones del cliente y servidor estaban tomando caminos incompatibles debido al uso de un TEE para el cliente.
- Detectar la incompatibilidad de la Raspberry Pi 4 Model B con OP-TEE.
- Definir una guía para poder utilizar Python en una Raspberry Pi 3 Model B que haga uso de OP-TEE. Hasta el momento, se desconocía cómo poder integrar este

lenguaje de programación en la Raspberry una vez que se comenzaba a utilizar OP-TEE, lo cual suponía un grave inconveniente para poder utilizar el ERA desarrollado.

- Detectar incompatibilidades entre el servidor de *bootstrapping*, el DDM y el ERA. Estas impiden aplicar las políticas de comportamiento definidas en ficheros MUD durante el proceso de *bootstrapping*.

Además, la evaluación de rendimiento en tiempos de ejecución realizada en este trabajo permite establecer medidas de referencia útiles para futuras mejoras de la PoC presentada. En particular, la medición del tiempo requerido para completar el proceso de *bootstrapping* puede ser utilizada como base para comparar el modelo CoAP-EAP con otras alternativas existentes, facilitando así la elección de la opción más viable en función del rendimiento.

No obstante, el trabajo también presenta ciertas limitaciones que deben ser abordadas en futuros desarrollos, especialmente para permitir un despliegue más generalizado y robusto. Estas quedan resumidas del siguiente modo:

1. Los dispositivos IoT se caracterizan por tener capacidades muy variadas. Por este motivo debemos hacer uso de protocolos ligeros. En este escenario, debido a problemas encontrados con el trabajo con OP-TEE se dejó de utilizar CoAP para el proceso de *bootstrapping*. Esto implica el uso de TCP para transportar los mensajes EAP, lo que puede suponer problemas con ciertos dispositivos IoT. Si queremos un escenario más generalista, se debe conseguir el uso de CoAP con OP-TEE (u otro TEE si así fuese necesario).
 2. En el escenario desplegado, la aplicación de las políticas del MUD tiene lugar una vez que ocurre el *bootstrapping*, pues el ERA y el *Bootstrapping Server* no están preparados para la intervención del DDM durante el proceso de *bootstrapping*. Esto supone una fuerte limitación en el escenario, pues ha llevado a incluir una entidad innecesaria, el *Post-Server*, y eliminar la automatización del DDM.
 3. Los ficheros MUD pueden ir siendo actualizados con el paso del tiempo. Es necesario ampliar la funcionalidad del DDM y MUD Manager para hacer comprobaciones periódicas del estado de los MUDs asociados a los dispositivos IoT que se estén gestionando. En caso de que estos hayan sido modificados se debería repetir el proceso de actualización del comportamiento de los dispositivos afectados.
 4. El ERA desplegado es activado una vez se completa el *bootstrapping*. Esto debería ser modificado para que se active durante dicho proceso, permitiendo aplicar las políticas de comportamiento definidas en los MUDs también en esta etapa y solucionar el problema mencionado en el punto 2.
-

5. En la prueba de este escenario todas las entidades que no pertenecían al dispositivo IoT se han desplegado sobre un mismo equipo físico. Para aumentar el realismo, este escenario debería desplegarse haciendo uso de equipos que estén físicamente separados.
6. Los mensajes que se envían hacia el ERA son mensajes sensibles. Estos deberían ir protegidos para evitar que un atacante pueda espiarlos. Este aspecto debe ser mejorado para proporcionar privacidad e integridad a esta comunicación.

En resumen, la PoC desarrollada representa un paso importante hacia la validación del enfoque propuesto por el proyecto CERTIFY. Aunque se han identificado algunas limitaciones, el trabajo realizado ofrece una base sólida sobre la que seguir construyendo y mejorando. Además, abre nuevas oportunidades para explorar cómo hacer que la gestión de dispositivos IoT sea más sencilla, segura y adaptable a distintos entornos.

Bibliografía

- [1] D. García Carrillo. *A CoAP-based bootstrapping service for large-scale Internet-of-things networks*. PhD thesis, Universidad de Murcia (UMU), Murcia, 2018.
- [2] Miguel Angel López Peña. *DevOps Aplicado a Sistemas IoT: Definición e Implementación de Procesos Continuos de Monitorización y Retroalimentación*. PhD Thesis, Universidad Politécnica de Madrid, Madrid, 2021. URL <http://oa.upm.es/68043/>.
- [3] Satyajit Sinha. IOT ANALYTICS, 2024. URL <https://iot-analytics.com/number-connected-iot-devices/>.
- [4] Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales., 2018. URL <https://www.boe.es/eli/es/lo/2018/12/05/3/con>.
- [5] CERTIFY, (s.f.). URL <https://certify-project.eu/>.
- [6] Eliot Lear, Ralph Droms, and Dan Romascanu. Manufacturer Usage Description Specification, 2019. URL <https://datatracker.ietf.org/doc/html/rfc8520>.
- [7] Sameh Khalfaoui. *Security bootstrapping for Internet of Things*. PhD thesis, Institut Polytechnique de Paris, Francia, 2022.
- [8] Dan García Carrillo. EAP-based Authentication Service for CoAP, 2024. URL <https://datatracker.ietf.org/doc/html/draft-ietf-ace-wg-coap-eap-12>.
- [9] Russ Housley and Dr. Bernard D. Aboba. Guidance for Authentication, Authorization, and Accounting (AAA) Key Management, 2007. URL <https://datatracker.ietf.org/doc/html/rfc4962>.
- [10] Dr. Bernard D. Aboba and Jonathan Wood. Authentication, Authorization and Accounting (AAA) Transport Profile, 2003. URL <https://datatracker.ietf.org/doc/html/rfc3539>.
- [11] John Vollbrecht, James D. Carlson, Larry Blunk, Bernard D. Aboba, and Henrik Levkowetz. Extensible Authentication Protocol (EAP), 2004. URL <https://datatracker.ietf.org/doc/html/rfc3748>.

-
- [12] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP), 2014. URL <https://datatracker.ietf.org/doc/html/rfc7252>.
- [13] Agustín Bassi. Introducción a CoAP, 2021. URL [https://www.gotoiot.com/pages/articles/coap_intro/index.html#:~:text=CoAP%20\(Constrained%20Application%20Protocol\)%20es,que%20puedan%20comunicarse%20sobre%20internet](https://www.gotoiot.com/pages/articles/coap_intro/index.html#:~:text=CoAP%20(Constrained%20Application%20Protocol)%20es,que%20puedan%20comunicarse%20sobre%20internet).
- [14] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999. URL <https://datatracker.ietf.org/doc/html/rfc2616>.
- [15] Eddy Wesley. Transmission Control Protocol (TCP), 2022. URL <https://datatracker.ietf.org/doc/html/rfc9293>.
- [16] User Datagram Protocol, 1980. URL <https://datatracker.ietf.org/doc/html/rfc768>.
- [17] Florent Bersani and Hannes Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method, 2007. URL <https://datatracker.ietf.org/doc/html/rfc4764>.
- [18] Itai Turbahn. Introduction to Trusted Execution Environments (TEEs), 2024. URL <https://www.dynamic.xyz/blog/trusted-execution-environments>.
- [19] What is a Trusted Execution Environment (TEE)?, 2019. URL <https://www.trustonic.com/technical-articles/what-is-a-trusted-execution-environment-tee/>.
- [20] What is a Trusted Application?, 2024. URL <https://licelus.com/insights/what-is-a-trusted-application#:~:text=A%20Trusted%20Application%20is%20a,%2C%20authentication%2C%20and%20secure%20storage>.
- [21] Introduction to Trusted Execution Environments. Technical report, 2018. URL <https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf>.
- [22] Jose L. Hernandez-Ramos, Sara N. Matheu, Angelo Feraudo, Gianmarco Baldini, Jorge Bernal Bernabe, Poonam Yadav, Antonio Skarmeta, and Paolo Bellavista. Defining the Behavior of IoT Devices Through the MUD Standard: Review, Challenges, and Research Directions. *IEEE Access*, 9:126265–126285, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3111477. URL <https://ieeexplore.ieee.org/document/9531614/>.
-

-
- [23] Martin Björklund. The YANG 1.1 Data Modeling Language, 2016. URL <https://datatracker.ietf.org/doc/html/rfc7950>.
 - [24] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format, 2017. URL <https://datatracker.ietf.org/doc/html/rfc8259>.
 - [25] Mahesh Jethanandani, Sonal Agarwal, Lisa Huang, and Dana Blair. YANG Data Model for Network Access Control Lists (ACLs), 2019. URL <https://datatracker.ietf.org/doc/html/rfc8519>.
 - [26] Stefano Sebastio, Sara Matheu, Antonio Skarmeta, Riccardo Orizio, Dimitris Karras, Daria Schumm, Burkhard Stiller, Rosella Omana Mancilla, Francesca Giampaolo, Simon Tuck, Thomas Bocek, Vincenzo Cuomo, Roland Atoui, Sreedevi Beena, Roberto Nardone, Alessandro Cilardo, Dinesh Sharma, Rohit Bohara, and Javier Parra-Domínguez. Cybersecurity Management Throughout the IoT Systems Lifecycle – The CERTIFY Approach. In Rashid Mehmood, Guillermo Hernández, Isabel Praça, Jaroslaw Wikarek, Roussanka Loukanova, Arsénio Monteiro Dos Reis, Antonio Skarmeta, and Eleonora Lombardi, editors, *Distributed Computing and Artificial Intelligence, Special Sessions I, 21st International Conference*, volume 1198, pages 281–290. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-76458-5 978-3-031-76459-2. doi: 10.1007/978-3-031-76459-2_26. URL https://link.springer.com/10.1007/978-3-031-76459-2_26. Series Title: Lecture Notes in Networks and Systems.
 - [27] Fulvio Valenza and Antonio Lioy. User-oriented Network Security Policy Specification.
 - [28] Alejandro Molina Zarca, Jorge Bernal Bernabé, Jordi Ortíz, and Antonio Skarmeta. Policy-based Definition and Policy for Orchestration FInal Report. Technical report, 2018. URL <https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5c03f6297&appId=PPGMS>.
 - [29] Sara N. Matheu, Alberto Robles Enciso, Alejandro Molina Zarca, Dan García-Carrillo, José Luis Hernández-Ramos, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Security Architecture for Defining and Enforcing Security Profiles in DLT/SDN-Based IoT Systems. 2020. URL <https://www.mdpi.com/1424-8220/20/7/1882>.
 - [30] Dan García Carrillo. draft-ietf-coap-eap-proof-of-concept, 2022. URL <https://github.com/dangarciacarrillo/draft-ietf-coap-eap-proof-of-concept>.
 - [31] OpenSSL Documentation, (s.f.). URL <https://docs.openssl.org/master/>.
 - [32] Universally Unique IDentifiers (UUIDs), 2024. URL <https://datatracker.ietf.org/doc/rfc9562/>.
-

-
- [33] Getting started with your Raspberry Pi, (s.f.). URL <https://www.raspberrypi.com/documentation/computers/getting-started.html>.
 - [34] About OP-TEE. URL <https://optee.readthedocs.io/en/latest/general/about.html>.
 - [35] Amit Nadiger. OPTEE, 2023. URL <https://www.linkedin.com/pulse/tee-optee-amit-nadiger>.
 - [36] Build and run, (s.f.). URL <https://optee.readthedocs.io/en/latest/building/prerequisites.html>.
 - [37] Jose Manuel Merlos Espín. CERTIFY-OP-TEE, 2024. URL <https://ants-gitlab.inf.um.es/jmanuel/certify-op-tee>.
 - [38] Sara Matheu, Stefano Sebastio, Antonio Skarmeta, Riccardo Orizio, Stefanos Vasi-leiadis, Vasilis Kalos, Katharina Müller, Thomas Grübl, Rosella Omana Mancilla, Simon Tuck, Thomas Bocek, Vincenzo Cuomo, Roland Atoui, Sreedevi Beena, Luigi Coppolino, Roberto Nardone, Dinesh Sharma, Rohit Bohara, Javier Parra-Domínguez, and Javier Prieto. Active Security for Connected Device Lifecycle: The CERTIFY Architecture. In Rashid Mehmood, Guillermo Hernández, Isabel Praça, Jaroslaw Wikarek, Roussanka Loukanova, Arsénio Monteiro dos Reis, Antonio Skarmeta, and Eleonora Lombardi, editors, *Distributed Computing and Artificial Intelligence, Special Sessions I, 21st International Conference*, pages 301–310, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-76459-2.
 - [39] Eric Rescorla. HTTP Over TLS, 2000. URL <https://datatracker.ietf.org/doc/html/rfc2818>.
 - [40] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Mes-sage Authentication, 1997. URL <https://datatracker.ietf.org/doc/html/rfc2104>.
 - [41] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, Jamal Hadi Salim, Da-vid Meyer, and Odysseas Koufopavlou. Software-Defined Networking (SDN): La-yers and Architecture Terminology, 2015. URL <https://datatracker.ietf.org/doc/html/rfc7426>.
 - [42] Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008. URL <https://www.w3.org/TR/xml/>.
 - [43] xml.etree.ElementTree - The ElementTree XML API. URL <https://docs.python.org/3/library/xml.etree.elementtree.html>.
 - [44] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture, 2006. URL <https://datatracker.ietf.org/doc/html/rfc4251>.
-

A. Anexo I. Fichero MUD generado

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://www.example.com/yourmudfile.json",
    "last-update": "2024-12-18T21:07:00+01:00",
    "cache-validity": 48,
    "extensions": [
      "umu-mspl-list:mspls"
    ],
    "is-supported": true,
    "modelname": "raspberry pi 3 model B",
    "systeminfo": "raspberry pi 3 model B current MUD File",
    "from-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "outbound-stricted-communication"
          }
        ]
      }
    },
    "to-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "inbound-stricted-communication"
          }
        ]
      }
    }
  },
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "outbound-stricted-communication",
        "type": "ipv4-acl-type",
        "aces": {
          "ace": [
            {
              "name": "output-rule",
              "matches": {
                "ipv4": {
                  "destination-ipv4-network": "192.168.100.0/24",
                  "protocol": 6
                },
                "tcp": {
                  "ietf-mud:direction-initiated": "from-device",
                  "destination-port": {
                    "operator": "eq",
                    "port": 443
                  }
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```

    },
    "actions": {
        "forwarding": "accept"
    }
}
]
}
},
{
    "name": "inbound-stricted-communication",
    "type": "ipv4-acl-type",
    "aces": {
        "ace": [
            {
                "name": "input-rule",
                "matches": {
                    "ipv4": {
                        "source-ipv4-network": "192.168.100.0/24",
                        "protocol": 6
                    },
                    "tcp": {
                        "ietf-mud:direction-initiated": "to-device",
                        "source-port": {
                            "operator": "any",
                            "port": "*"
                        }
                    }
                },
                "actions": {
                    "forwarding": "accept"
                }
            }
        ]
    }
}
]
},
"umu-mspl-list:mspls": {
    "mspl": [
        {
            "name": "mspl_change_signature",
            "type": "software-mspl-type",
            "configuration": {
                "capability": "software-protection",
                "rule-set-configuration": {
                    "name": "change_signature_rule_set",
                    "configuration-rule": {
                        "name": "change_signature_rule",
                        "configuration-action": {
                            "software-protection-action": {
                                "software-protection-type": "USE-HMAC"
                            }
                        }
                    }
                }
            }
        }
    ]
},
{
    "name": "mspl_filtering",
    "type": "ipv4-mspl-type",
    "configuration": {
        "capability": "Filtering_L4",
        "rule-set-configuration": {
            "name": "filtering_rule_set",

```

```
        "configuration-rule": [  
            {  
                "name": "filtering_rule",  
                "configuration-action": {  
                    "filteringAction": {  
                        "filtering-action-type": "ALLOW"  
                    }  
                },  
                "configuration-condition": {  
                    "filtering-configuration-condition": {  
                        "packet-filter-condition": {  
                            "source_dnsname": "bootstrapping-server.umu",  
                            "destination_dnsname": "raspberrypi-3-modelB-TFG.umu",  
                            "source_port": "any",  
                            "destination_port": "any",  
                            "direction": "INBOUND",  
                            "protocol_type": "6"  
                        }  
                    }  
                }  
            }  
        ]  
    }  
}
```

B. Anexo II. Traducción del fichero MUD a MSPL

```
<!-- MSPL 1 - Permitir tráfico con la red 192.168.100.0/24 -->
<?xml version="1.0" encoding="UTF-8"?>
<mspl>
  <name type="str">mspl_example</name>
  <configuration type="dict">
    <capability type="str">Filtering_L4</capability>
    <rule-set-configuration type="dict">
      <name type="str">rule_set963b7a3-1dcb-49e2-8ed6-3fb9a1c37c64</name>
      <default-action type="str">deny</default-action>
      <configuration-rule type="dict">
        <output-rule type="dict">
          <external-data type="dict">
            <priority type="int">1</priority>
          </external-data>
          <configuration-action type="dict">
            <filtering-action type="dict">
              <filtering-action-type type="str">allow</filtering-action-type>
            </filtering-action>
          </configuration-action>
          <configuration-condition type="dict">
            <filtering-configuration-condition type="dict">
              <packet-filter-condition type="dict">
                <destination-address type="str">192.168.100.0/24</destination-address>
                <destination-port type="str">eq 443</destination-port>
                <direction type="str">OUTBOUND</direction>
                <protocol-type type="str">6</protocol-type>
              </packet-filter-condition>
            </filtering-configuration-condition>
          </configuration-condition>
          <name type="str">output-rule</name>
        </output-rule>
        <input-rule type="dict">
          <external-data type="dict">
            <priority type="int">1</priority>
          </external-data>
          <configuration-action type="dict">
            <filtering-action type="dict">
              <filtering-action-type type="str">allow</filtering-action-type>
            </filtering-action>
          </configuration-action>
          <configuration-condition type="dict">
            <filtering-configuration-condition type="dict">
              <packet-filter-condition type="dict">
                <source-address type="str">192.168.100.0/24</source-address>
                <source-port type="str">any *</source-port>
                <direction type="str">INBOUND</direction>
                <protocol-type type="str">6</protocol-type>
              </packet-filter-condition>
            </filtering-configuration-condition>
          </configuration-condition>
        </input-rule>
      </configuration-rule>
    </rule-set-configuration>
  </configuration>
</mspl>
```

```

        </filtering-configuration-condition>
        </configuration-condition>
        <name type="str">input-rule</name>
    </input-rule>
</configuration-rule>
</rule-set-configuration>
</configuration>
</mspl>

<!-- MSPL 2 - Política de cambio de algoritmo de firma -->
<?xml version="1.0" encoding="utf-8"?>
<mspl>
    <configuration>
        <capability>software-protection</capability>
        <rule-set-configuration>
            <configuration-rule>
                <configuration-action>
                    <software-protection-action>
                        <software-protection-type>USE-HMAC</software-protection-type>
                    </software-protection-action>
                </configuration-action>
                <name>change_signature_rule</name>
            </configuration-rule>
            <name>change_signature_rule_set</name>
        </rule-set-configuration>
    </configuration>
    <name>mspl_change_signature</name>
    <type>software-mspl-type</type>
</mspl>

<!-- MSPL 3 - Permitir tráfico entrante desde servidor de bootstrapping de la UMU -->
<?xml version="1.0" encoding="utf-8"?>
<mspl>
    <configuration>
        <capability>Filtering_L4</capability>
        <rule-set-configuration>
            <configuration-rule>
                <configuration-action>
                    <filteringAction>
                        <filtering-action-type>ALLOW</filtering-action-type>
                    </filteringAction>
                </configuration-action>
                <configuration-condition>
                    <filtering-configuration-condition>
                        <packet-filter-condition>
                            <destination_dnsname>raspberrypi-3-modelB-TFG.umu</↵
↵ destination_dnsname>
                                <destination_port>any</destination_port>
                                <direction>INBOUND</direction>
                                <protocol_type>6</protocol_type>
                                <source_dnsname>bootstrapping-server.umu</source_dnsname>
                                <source_port>any</source_port>
                            </packet-filter-condition>
                        </filtering-configuration-condition>
                    </configuration-condition>
                    <name>filtering_rule</name>
                </configuration-rule>
                <name>filtering_rule_set</name>
            </rule-set-configuration>
        </configuration>
        <name>mspl_filtering</name>
        <type>ipv4-mspl-type</type>
    </mspl>

```


C. Anexo III. Base de datos del *Device and Domain Manager*

En este anexo se incluye una imagen sobre el contenido de la base de datos del *Device and Domain Manager* una vez la Raspberry Pi 3 Model B ha completado el proceso de *bootstrapping*. El motivo de añadir este anexo es que en la sección 4.4 es imposible colocar la imagen de forma que esta pueda ser legible. Por ello, hacemos uso del formato apaisado en este anexo para poder visualizar el contenido de la base de datos.

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Attach Database Save Project Open Project Close Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: device

id ip mac model identifier mudurl mudfile mitigation-actions

Filter Filter Filter Filter Filter Filter Filter

1 1 192.168.18.67 NULL high_end_device-192.168.18.67 a96a18fc-a9b3-404d-88e0-2448a9466d3e https://www.example.com/yourmudfile.json {"ietf-access-control-list-ads": {"acl": [{"aces": "...

Figura C.1: PoC: Base de datos del DDM tras completar el bootstrapping

D. Anexo IV. Configurar SSH en una Raspberry Pi 3 Model B

Al trabajar con la Raspberry Pi 3 Model B una propiedad muy interesante es poder acceder mediante SSH a esta. Al haber desplegado OP-TEE, es necesaria una pequeña configuración para poder hacer uso de SSH. El objetivo de este anexo es exponer cómo hacer esto.

El primer paso es editar el fichero `/etc/ssh/sshd_config` de la Raspberry. En este debemos añadir las líneas:

```
PermitRootLogin yes
PasswordAuthentication yes
```

Esto permite utilizar al usuario *root*, definido en la Raspberry al desplegar OP-TEE, para la conexión SSH. Tras esto, debemos configurar una contraseña para dicho usuario:

```
passwd root
# Habrá que introducir dos veces la contraseña deseada para este usuario
```

Finalmente, habrá que reiniciar el servicio SSH:

```
killall sshd
/usr/sbin/sshd
```

De este modo, podremos hacer uso de SSH a través del usuario *root*. Para ello, se debe ejecutar en nuestro ordenador personal el siguiente comando:

```
ssh root@<dirección_IP_raspberry>
# Habrá que introducir la contraseña que se configuró para este usuario
```