



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

**Implementación de una librería de seguridad mediante
el módulo de plataforma confiable sobre Sistemas de
Agentes Móviles**

**Implementation of a security library using the trusted
platform module on Mobile Agent Systems**

Realizado por:

Iván García Aguilar

Tutorizado por:

Antonio Jesús Muñoz Gallego

Departamento:

Lenguajes y Ciencias de la computación

MÁLAGA, Julio, 2020



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

**Implementación de una librería de seguridad mediante
el módulo de plataforma confiable sobre Sistemas de
Agentes Móviles**

**Implementation of a security library using the trusted
platform module on Mobile Agent Systems**

Realizado por:

Iván García Aguilar

Tutorizado por:

Antonio Jesús Muñoz Gallego

Departamento:

Lenguajes y Ciencias de la computación

MÁLAGA, Julio, 2020

Fecha de defensa:

Fdo.: El Secretario del Tribunal

Resumen:

El principal objetivo de este trabajo fin de grado, ha sido establecer e implantar un protocolo que permita la comunicación de forma segura, entre agentes móviles distribuidos a través de la red, haciendo uso de hardware criptográfico. En la actualidad, la seguridad corresponde con uno de los aspectos vitales a tener en cuenta. No existen protocolos de comunicación que hagan uso del enfoque acontecido en dicho trabajo. La solución establecida es de gran utilidad en diversos campos, tanto para empresas, así como entornos judiciales entre otros.

Haciendo uso de una serie de conceptos del ámbito de la seguridad previamente establecidos, en concreto mediante un hardware criptográfico denominado TPM, y una serie de funcionalidades que son clave para el diseño de nuestra solución, como es por ejemplo el uso de claves asimétricas, ha sido posible implantar el nuevo enfoque determinado.

Con el objetivo de utilizar dicho protocolo, se ha implementado una aplicación en java, haciendo uso del framework de desarrollo denominado Jade, abstrayendo la complejidad del proceso en base al protocolo definido. Para poner a prueba dicha aplicación, se ha establecido un posible caso práctico, ligado con el entorno judicial y la cadena de custodia.

Palabras claves: Claves asimétricas, TPM, Java, Jade, Cadena de Custodia.

Abstract:

The main goal of this Undergraduate Thesis has been to establish and implement a protocol, that allows secure communication between mobile agents distributed through the network using cryptographic hardware. Currently, security is one of the vital aspects to consider. There are not communication protocols that make use of this approach. The developed solution is very useful in various fields, both for companies, as well as judicial environments, etc.

Using a series of previously established security concepts, specifically through cryptographic hardware called TPM, and a series of functionalities that are the key to the design of our solution, such as the use of asymmetric keys, it has been possible to implement the determined new approach.

In order to use this protocol, an application has been implemented in java, making use of the development framework called Jade, abstracting the complexity of the process based on the defined protocol. To test this application, a possible practical case has been established, related to the judicial environment and the chain of custody.

Keywords: Asymmetric Keys, TPM, Java, Jade, Chain of Custody.

Índice general

1	Introducción:	13
1.1	Motivación:	13
1.2	Objetivos:	14
2	Conceptos Previos y Estado del arte:	19
2.1	Conceptos Previos:	19
2.1.1	Jade:	19
2.1.1.1	Arquitectura de las Plataformas:	20
2.1.1.2	Arquitectura de los Servicios:	23
2.1.1.3	Arquitectura de los Agentes:	26
2.1.1.4	Movilidad:	28
2.1.2	TPM:	29
2.1.2.1	Arquitectura del TPM:	31
2.1.2.2	Creación de claves:	33
2.1.2.3	Atestación del Sistema:	35
2.1.2.4	Especificación TPM 2.0:	38
2.2	Estado del Arte:	39
3	Metodología de trabajo:	41
4	Desarrollo del Protocolo Seguro :	45
4.1	Fases de Desarrollo:	45
4.1.1	Primer Prototipo:	45
4.1.2	Segundo prototipo:	46
5	Resultados:	69

5.1	Problemas y soluciones desarrolladas:	69
5.1.1	Problemas de sellado en el primer prototipo:	69
5.1.2	Atestación del sistema no actual:	70
5.1.3	Migraciones no autorizadas por la CA:	71
5.1.4	Interfaz Gráfica:	72
5.2	Caso de Uso:	73
6	Conclusión y líneas futuras:	79
7	ANEXO 1 - INSTALACIÓN Y EJECUCIÓN DE JADE:	83
8	ANEXO 2 - INSTALACIÓN DEL TPM:	91
	Bibliografía	93

Capítulo 1

Introducción:

La seguridad constituye uno de los principales puntos en la actualidad en cualquier sistema software. En un contexto cambiante como el que vivimos actualmente, en el cual, aparecen nuevos ataques diariamente, surge la necesidad de aportar nuevas soluciones que permitan mitigar a los mismos. Por ello, a continuación, se ha realizado una introducción del ámbito sobre el cual gira dicho proyecto, determinando los aspectos que han motivado su diseño e implementación, así como la solución que se ha conseguido desarrollar.

1.1 Motivación:

Cada año, surgen nuevos ataques, siendo cada vez estos más sofisticados. Algunos parecen estar motivados por razones económicas, o bien para conseguir cierta información sensible. Estos ataques tienen como objetivos tanto a usuarios como a empresas, dando lugar así a grandes perdidas, propiciando por ello la necesidad de establecer mecanismos para protegerse frente a los mismos cuando sea posible, o mitigar su potencial impacto.

Este proyecto, tiene como objetivo principal el diseño e implementación de un protocolo de comunicación seguro entre agentes móviles, basado en una arquitectura hardware concreta TPM. Se ha optado por resolver el problema del código móvil seguro, a través del hardware criptográfico, así como un framework de desarrollo denominado Jade¹. En este caso, se ha utilizado la

¹<https://jade.tilab.com/>

implementación de un módulo del fabricante Infineon ², implantado en una raspberry pi, como escenario de validación del protocolo propuesto, con el fin de lograr una solución sencilla, eficaz y con pocos recursos.

Se quiere resaltar el hecho de que esta implementación puede utilizarse sobre cualquier plataforma junto al uso del TPM bajo la especificación 2-0. Existe un repositorio destinado al desarrollo de nuevas funcionalidades con respecto al módulo de plataforma confiable. Todas estas funciones, así como el framework de desarrollo, disponen de una licencia de código libre propiciando así la creación de nuevas soluciones.

Bajo estas premisas descritas anteriormente, podemos determinar por tanto, que existen necesidades latentes en el desarrollo e implementación de nuevas soluciones destinadas al ámbito de la ciberseguridad. Las funcionalidades que ofrece el módulo de plataforma confiable, se encuentran en una versión primitiva, sumando a ello el hecho de que para construir soluciones sobre TPM, es necesario tener conocimientos profundos en seguridad, además de entender la arquitectura y funcionamiento de la plataforma en su totalidad.

1.2 Objetivos:

Objetivo principal:

El principal objetivo de este TFG, es diseñar e implantar un protocolo de comunicación seguro, entre agentes localizados en diversas plataformas, distribuidos a lo largo de la red. El paradigma del agente móvil realiza migraciones seguras entre plataformas una vez finalizada su respectiva tarea u objetivo, con el fin de determinar integridad, autenticación y confidencialidad

²<https://www.infineon.com/>

sobre los datos que gestiona el agente.

Para conseguir dicho propósito, se han establecido una serie de subobjetivos específicos.

Subobjetivos específicos:

- **Creación de un nuevo servicio de comunicación:** A través de este servicio, se permite la comunicación entre plataformas distribuidas en la red, de forma paralela, es decir, sin interrumpir el flujo de ejecución habitual de cada una de ellas.
- **Implementación del cifrado asimétrico:** Se ha implementado un criptosistema de clave pública para cifrar la información y poder establecer un intercambio de mensajes con la plataforma segura centralizada, antes de realizar la migración, así como firmar los datos, garantizando la autenticidad de la misma.
- **Uso de las funcionalidades del TPM:** A través del uso del sellado de la información, además de la firma, es posible conseguir la integridad de los datos (sealing), así como la autenticidad del origen de los mismos, con el objetivo de posteriormente validarlos dentro de la plataforma segura.
- **Caso de uso:** Se ha determinado un caso de uso real para la validación de dicho protocolo. Por ello, se presenta un contexto relacionado con el entorno judicial y la cadena de custodia, estableciendo así una posible solución para la transmisión de información segura entre plataformas, verificando previamente, que los dispositivos tanto de origen como destino no han sido manipulados por una entidad no autorizada para ello.

Estructura del documento:

El presente documento, está estructurado en varios capítulos, que aglutinan todos los pasos seguidos a lo largo del desarrollo del proyecto.

CAPÍTULO 2 - CONCEPTOS PREVIOS Y ESTADO DEL ARTE:

En este capítulo, se describe de forma detallada, todas las tecnologías utilizadas durante la elaboración del proyecto, definiendo el framework de desarrollo, estableciendo las nociones básicas del mismo, así como del módulo confiable, librerías y de las funcionalidades que se han utilizado.

CAPÍTULO 3 - METODOLOGÍA DE TRABAJO:

En este capítulo, se detalla la metodología usada, describiendo las pautas que se han tomado, así como las reuniones acontecidas con el tutor con el fin de establecer continuas mejoras durante la evolución del proyecto.

CAPÍTULO 4 - FASES DE DESARROLLO:

Se han establecido las fases acontecidas durante todo el proyecto, considerando una serie de mejoras que se han ido integrando en la evolución del mismo. Este punto, constituye con uno de los capítulos más extensos, pues se ofrece una descripción detallada del protocolo, así como los pasos seguidos para la consecución de su completo desarrollo y testeo.

CAPÍTULO 5 - RESULTADOS Y PRUEBAS:

Dentro de este capítulo, se expone un caso de uso de un problema real que ayuda a resolver la solución desarrollada en el proyecto. Además, se describe tanto un análisis del problema, así como una serie de pruebas realizadas.

CAPÍTULO 6 - CONCLUSIÓN Y LÍNEAS FUTURAS:

Por ultimo, se exponen una serie de conclusiones tras la finalización del proyecto, y se establecen las bases para una posible continuación del desarrollo de dicho trabajo en el futuro.

Capítulo 2

Conceptos Previos y Estado del arte:

2.1 Conceptos Previos:

Para la realización de este proyecto, se ha utilizado un framework de desarrollo denominado Jade, así como diversas funcionalidades provistas gracias al módulo de plataforma confiable. Dentro de este capítulo, se van a describir por ello, tanto el funcionamiento de Jade, así como las operatividad del TPM.

2.1.1 Jade:

Corresponde con un framework de desarrollo empleado con el objetivo de desplegar principalmente agentes, utilizando Java como lenguaje de programación. Gracias a este framework de código abierto [2], es posible por tanto la creación de aplicaciones punto a punto, a través de una serie de agentes, donde cada uno de ellos desempeña una tarea o labor en concreto.

Una de las principales ventajas de este framework, es que permite la distribución de agentes en diversas plataformas, con diferentes sistemas operativos (multiplataforma). Además, propicia la creación de sistemas multiagentes a través de una serie de servicios, que cumplen con todas las especificaciones establecidas por la FIPA, cuyas siglas hacen alusión a la fundación para agentes físicos inteligentes.

Con todo esto lo que se pretende es garantizar la interoperabilidad de los nuevos servicios desarrollados, aportando un modelo en el cual se gestiona desde la creación, hasta la destrucción, así como la comunicación entre agentes.

Puesto que el protocolo desarrollado requiere del intercambio de mensajes entre plataformas, se ha seguido las pautas y especificaciones propuestas por este estándar.

2.1.1.1 Arquitectura de las Plataformas:

De acuerdo al estándar FIPA descrito, una plataforma está formada por una serie de elementos. Se adjunta una imagen de dicha arquitectura:

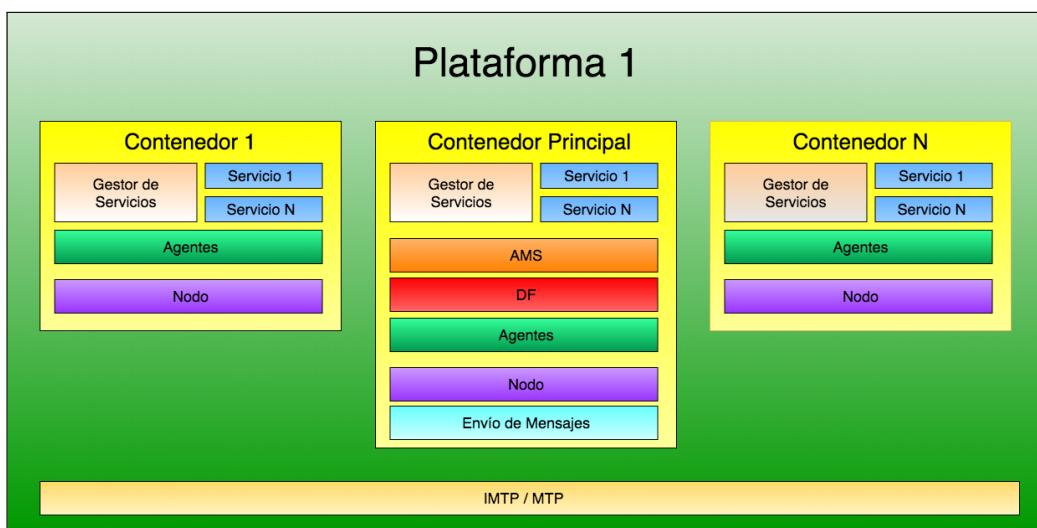


Figura 1: Arquitectura de una plataforma en JADE.

A continuación, se describe cada uno de los elementos que componen dicha arquitectura:

- **Agent Management System - AMS:** Este elemento se encarga principalmente de la gestión de la plataforma, ya que tiene información acerca del estado de la misma en todo momento, así como de los agentes que administra. Una de las tareas de este elemento es la modificación de los estados de cada uno de los agentes, además de la creación y

destrucción de los mismos. Se encarga también de gestionar tanto el canal de comunicación, así como los recursos del sistema. Es el encargado directo de la movilidad de los agentes entre plataformas, punto importante a tener en cuenta a la hora de validar dichas migraciones de acuerdo al protocolo desarrollado.

Además, provee de un servicio denominado ANS cuyas siglas hacen alusión al Agent Name Service. Dicho servicio se encarga de mantener un listado con los agentes que componen la plataforma, almacenando para cada uno de ellos tanto el nombre como la dirección asociada, permitiendo así la búsqueda de uno de los agentes en caso de que fuese necesario. Este servicio también es conocido como páginas blancas.

- **Directory Facilitator - DF:**

Este elemento es el encargado de gestionar un servicio denominado páginas amarillas, permitiendo buscar a un agente, de acuerdo a las capacidades del mismo, en lugar de por su nombre. Un agente incluido en la plataforma, puede registrarse en dicho directorio, determinando los servicios que ofrece, para que en caso de que se requiera por parte de otro agente, pueda ser localizado fácilmente.

- **Agent Communication Channel - ACC:**

Este elemento es fundamental, pues su cometido principal es la gestión de los mensajes entre agentes de diversas plataformas. Tiene como función el encaminamiento de mensajes desde el agente en la plataforma origen hasta la plataforma destino.

En un envío inter-plataforma, dicho mensaje será encaminado entre los ACC que conforman tanto la plataforma origen como destino. Este modelo de comunicación es asíncrono, es por ello por lo que se gestionan los mensajes de acuerdo a una cola para entrada, y otra para salida.

Destacar que los mensajes deben seguir unas especificaciones del tipo ACL, *Agent Communication Language*, definido por FIPA. Este tipo de mensaje, posee una serie de campos predefinidos, los cuales es necesarios tener en cuenta a la hora de poder desarrollar dicho protocolo.

Alguno de los campos más significativos se enuncian a continuación:

- **Sender:** Emisor de dicha solicitud.
- **Receiver:** Receptores del mensaje.
- **Content:** Contenido e información que gestiona.
- **Language:** Lenguaje en el cual se expresa la información del mensaje.
- **Ontology:** Ontología usada en el mensaje.
- **Performative:** Tipo de mensaje, con el objetivo de determinar el propósito que el emisor desea conseguir. Entre dichos tipos se encuentran por ejemplo *AGREE*, *CANCEL*, *CONFIRM*, etc...
- **Communication Type:** Dentro de esta opción, podemos encontrar opciones como *reply-with* para responder a un agente en concreto, *reply-by* con el objetivo de filtrar los mensajes en base a un tramo horario o fecha preestablecido o *in-reply-to* para expresar una acción anterior al mensaje correspondiente entre otros.

- **Message Transport Protocol - MTP:**

El envío de mensajes entre agentes localizados en diversas plataformas, corresponde con una de las tareas críticas, puesto que entra en juego los aspectos de rendimiento y escalabilidad. Dicho protocolo por tanto, se utiliza a nivel inter-plataforma.

- **Internal Message Transport Protocol - IMTP:**

Este elemento se encarga de gestionar el envío y recepción de los mensajes a nivel intra-plataforma.

La estructura de un entorno real, estaría formado por un conjunto de plataformas. En cada una de ellas, permanecerían localizados una serie de agentes que se comunicarían a través del intercambio de mensajes.

2.1.1.2 Arquitectura de los Servicios:

Uno de los aspectos fundamentales a la hora de desarrollar nuevos servicios dentro de Jade, se basa en conocer la arquitectura. Un servicio está compuesto por una serie de elementos, los cuales se enunciarán a continuación:

- **Filtros:** Estos elementos son vitales a la hora del procesamiento de comandos verticales. Podemos encontrar dos tipos de filtros, conocidos como *Incoming* y *Outgoing filters*, con el fin de filtrar comandos de entrada y salida respectivamente.
- **Sumideros:** Elementos cuya tarea se basa primordialmente en consumir los comandos generados por el servicio.

- **Helper:** Correspondiente a una interfaz, en la cual se especifican las acciones o métodos que puede realizar el agente, con el fin de interactuar con un servicio en concreto.
- **Slice:** La clase *Slice* y *Proxy* son usadas para la gestión de los comandos intercambiados entre nodos de la misma plataforma.

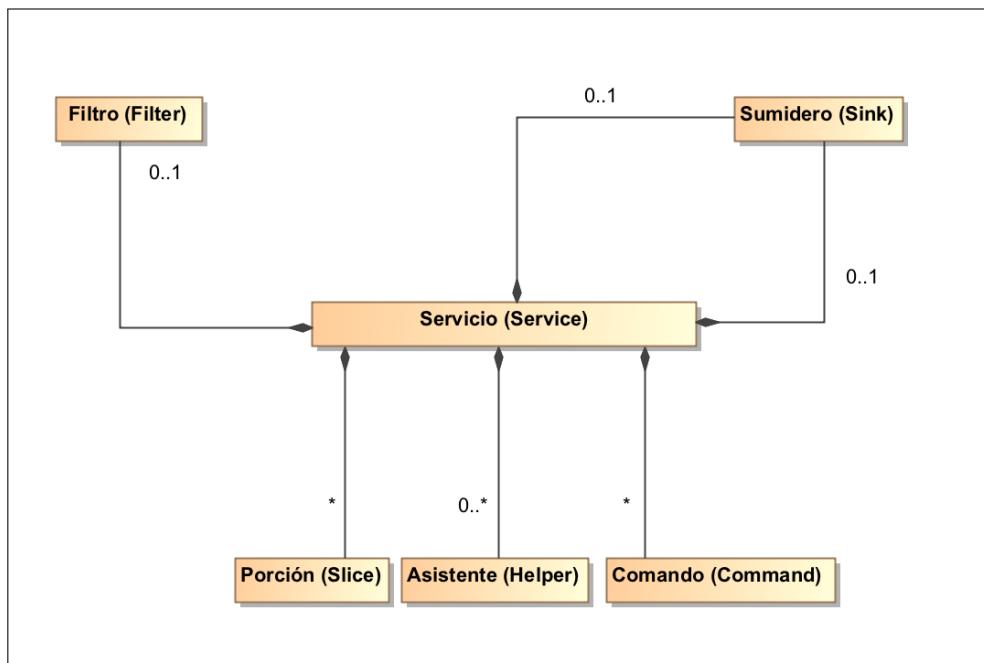


Figura 2: Componentes de un servicio.

Dentro de un servicio, podemos encontrar dos tipos de comandos:

Comandos Verticales: Estos comandos son creados en un servicio en concreto, con el objetivo de ser tratado por el mismo dentro de un mismo nodo, dotando al agente de cierta funcionalidad. Estos filtros tal y como se establecía anteriormente, son seleccionados de acuerdo al servicio.

Comandos Horizontales: Los comandos horizontales, en cambio, son generados de la misma forma dentro de los servicios, sin embargo, se utilizan para interaccionar con servicios de nodos remotos. De acuerdo a la especificación establecida a la hora de crear un nuevo servicio, cuando se recibe un comando horizontal dentro de un nodo remoto, este se procesa y posteriormente, se genera un nuevo comando vertical con el objetivo de que sea tratado en la correspondiente capa del servicio.

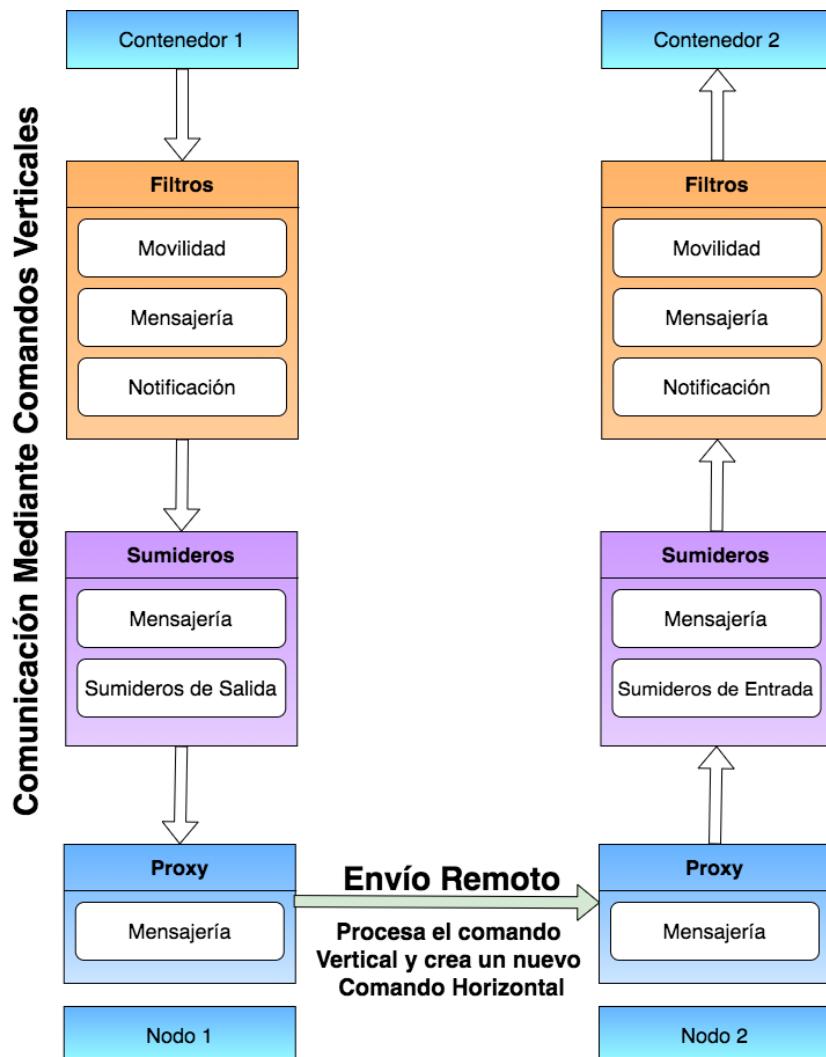


Figura 3: Esquema de comportamiento de los comandos generados.

2.1.1.3 Arquitectura de los Agentes:

En Jade, cada agente es procesado mediante un thread. Además, posee un buzón de mensajes, en el cual se van almacenando las mismos para posteriormente, tratarlos sin tener que interrumpir su flujo de procesamiento. Los nuevos agentes definidos, deben extender de la clase Agent y además, deben ser serializables, para su posterior migración inter-plataforma una vez solicitado por el mismo.

Una breve aproximación de la arquitectura de estos agentes sería la siguiente:

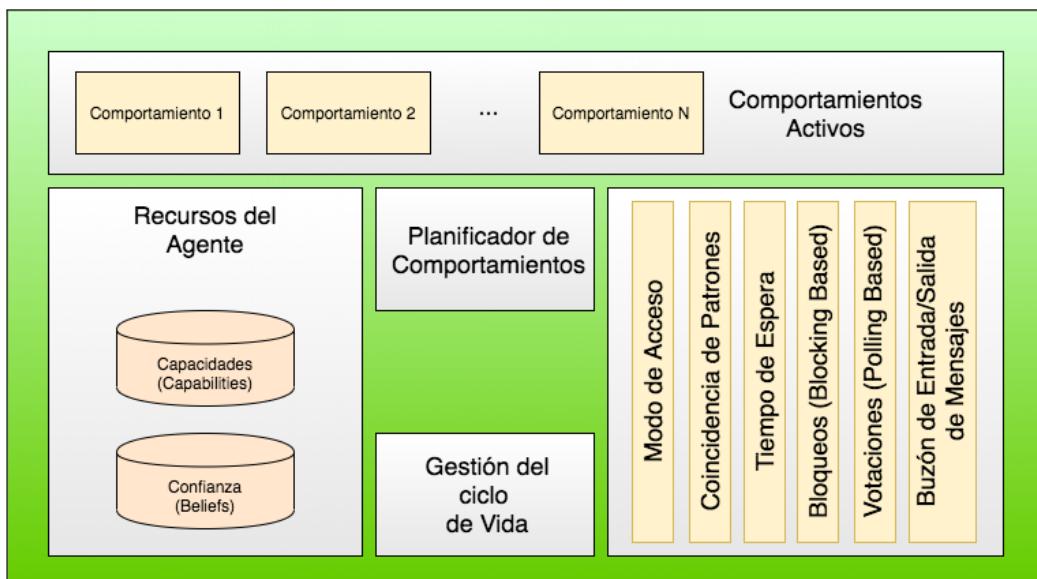


Figura 4: Arquitectura de los Agentes.

Para facilitar tareas de desarrollo a la hora de definir nuevas funcionalidades, surge el concepto de comportamientos, conocidos como *Behaviours*. Gracias a los mismos, es posible realizar nuevas funciones. Estas, pese a que den la sensación de actuar en paralelo, comparten el thread definido para

el agente. En caso de requerir parallelización total, será necesario crear un comportamiento que extienda de la clase *ThreadedBehaviour*.

Los servicios predefinidos por Jade son numerosos, adaptándose a la gran mayoría de necesidades, y pueden ser clasificados en simples o compuestos. Cada categoría, alberga a un conjunto de comportamientos, los cuales pueden verse en la siguiente imagen adjunta:

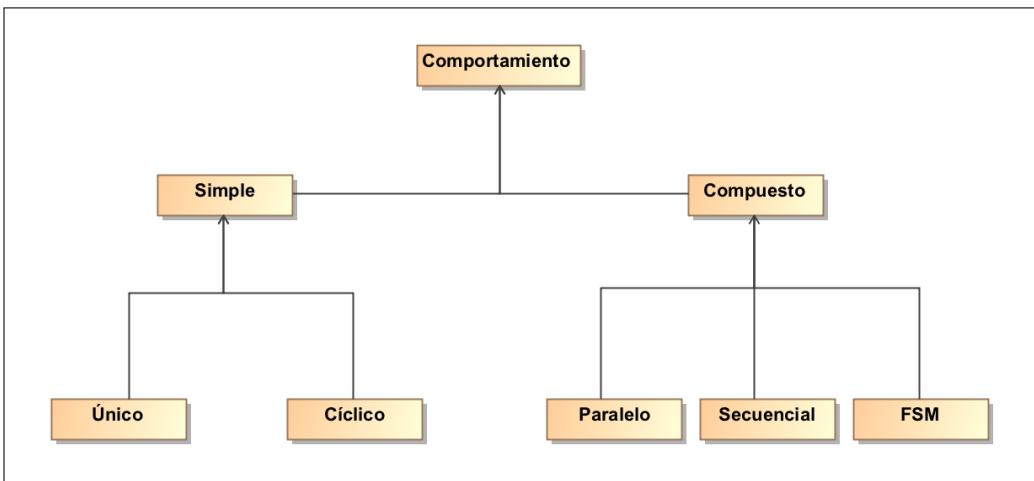


Figura 5: Tipos de comportamientos.

Comportamientos de tipo simple, son utilizados para la realización de tareas atómicas, entre los cuales se encuentra *OneShotBehaviour* (único), para la ejecución de un proceso una sola vez, o *CyclicBehaviour* (cíclicos) que tal y como su nombre indica, permite la ejecución de tareas en base a un periodo definido de forma cíclica. En cambio, los comportamientos de tipo complejo, son utilizados para la definición de tareas compuestas, pudiendo definir subcomportamientos secuenciales, paralelos, etc...

Para la ejecución del protocolo el cual concierne dicho informe, se han usado principalmente dos tipos de comportamientos extras, para el envío y

recepción de mensajes:

- **ReceiverBehaviour:** Recibe mensajes y ejecuta la operación solicitada por el mismo.
- **SenderBehaviour:** Envía mensajes creados por un agente en concreto.

Ambos servicios trabajan de forma totalmente paralela, es decir, no interrumpe el flujo de ejecución de las tareas del agente, y solo se activan en el caso de que se desea enviar un mensaje, o el agente reciba uno de ellos.

2.1.1.4 Movilidad:

Cuando un agente solicita realizar una migración a un contenedor, a nivel inter-plataforma debe migrar su código al completo. Utilizando el framework de desarrollo base, solo es posible migrar agentes dentro de una misma plataforma. Para realizar este proceso, se hace uso de dos métodos en concreto:

- `doMove(Location destiny)`: Es utilizado para mover a un agente al destino introducido como parámetro.
- `doClone(Location destiny)`: Realiza un proceso similar al método descrito anteriormente, con la salvedad de que realiza una copia en la plataforma destino.

Por otra parte, este framework ofrece una clase denominada *Movable*, con cuatro métodos, utilizados antes y después tanto del proceso de migración como

clonado. Estos métodos reciben el nombre de *afterMove()*, *afterClone()*, *beforeMove()* y *beforeClone()*.

Con respecto al protocolo definido, será necesario la comunicación a nivel inter-plataforma, es por ello por lo que se utilizará un complemento que ofrece Jade denominado IPMS³ cuyas siglas hacen referencia a *Inter Platform Mobility Service*.

A efectos prácticos, el comportamiento es similar al descrito a nivel intra-plataforma, sin embargo, se realizan procesos alternos. Para permitir dicha migración, previamente se genera un jar del agente, el cual contiene todas las librerías y contenido del mismo. Acto seguido, se envía de forma serializada esta información a través de un mensaje de tipo ACL.

Una de las consideraciones a tener en cuenta es que no se establece ningún tipo de seguridad en los mensajes intercambiados a través de la red. Gracias al uso de claves asimétricas, se ha resuelto dicho problema. Se entrará en mayor detalle en puntos posteriores.

2.1.2 TPM:

Las siglas TPM, hacen referencia a *Trusted Platform Module* y corresponde con un criptoprocesador seguro. Las ventajas que aporta el mismo son numerosas, alguna de las principales funcionalidades son las siguientes:

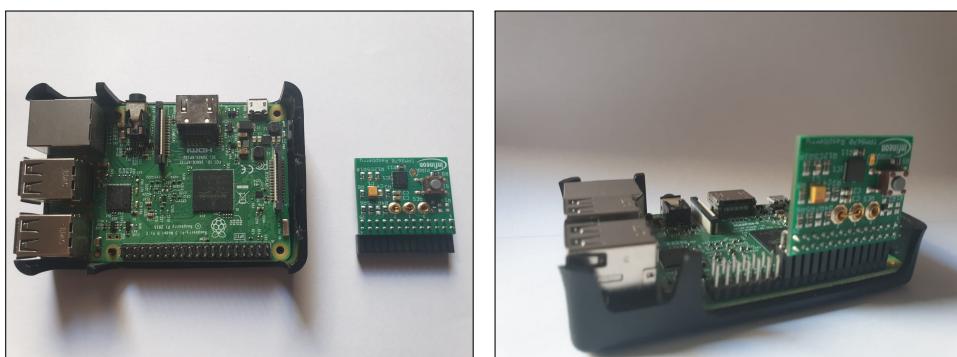
- Almacenamiento seguro de información y claves. Dicho dispositivo hace uso de dos tipos de memoria, una de ellas volátil para almacenar mediciones que se vayan realizando a lo largo de la ejecución del dispositivo en caso de que se requiera, o bien una memoria no volátil en la cual se

³<https://jade.tilab.com/dl.php?file=ipmsAddOn-1.5.zip>

almacenar claves.

- Ejecución de funciones criptográficas avanzadas, como el cifrado o la firma de datos.
- Funciones para garantizar la integridad del dispositivo, gracias al sellado de la información, conocido como *seal*. De acuerdo a los valores PCR, los cuales se entrará en detalle posteriormente, sella los datos. En caso de que se desee acceder de nuevo a la misma, el estado del sistema no debe haber variado, ya que de ocurrir dicha premisa, no se permitirá la lectura la información.

La mayoría de las funciones proporcionadas hoy día en el ámbito de la seguridad, se realizan a nivel software. Gracias a TPM, es posible ofrecer una solución que combine tanto el hardware como el software. Dicho criptoprocesador está integrado directamente con el dispositivo, en este caso, con una raspberry pi. Para este proyecto, se ha usado un TPM del fabricante Infineon⁴. Una vez acoplada a la raspberry pi, el sistema queda de la siguiente forma:



(a) Raspberry y TPM

(b) Dispositivo Seguro

Figura 6: Piezas que componen nuestro sistema

⁴<https://www.mouser.es/new/infineon/infineon-slm9670-board/>

2.1.2.1 Arquitectura del TPM:

La arquitectura del dispositivo está compuesta por una serie de componentes. Cada uno de estos, han sido diseñados y establecidos con el objetivo de ofrecer fiabilidad y seguridad al dispositivo en el que se implanta, ofreciendo por ello una serie de funcionalidades como el sellado de información o la firma digital de datos entre otras. Estas funcionalidades son esenciales para la implementación del protocolo seguro que se ha desarrollado. A continuación, se incluye una imagen en la que se muestran las distintas partes que conforman el TPM.

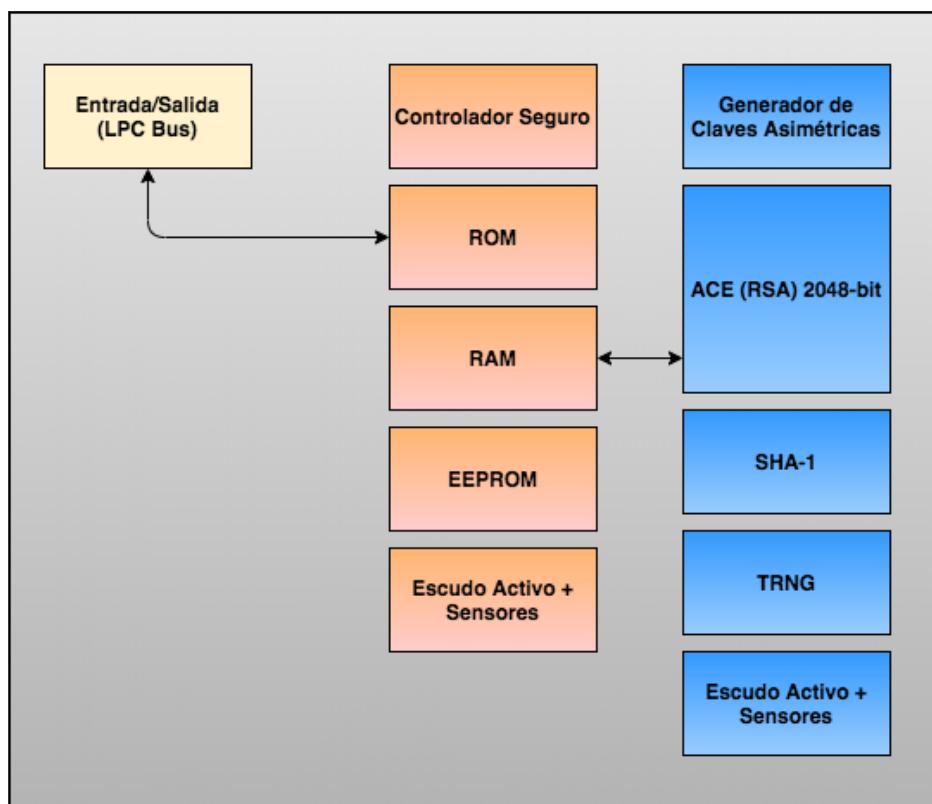


Figura 7: Componentes TPM⁵

⁵http://www.cs.unh.edu/~it666/reading_list/Hardware/tpm_fundamentals.pdf

- **I/O:** Se encarga del manejo de información la cual tiene lugar en los buses de comunicación que componen el dispositivo. Realiza una serie de funciones, como por ejemplo la codificación y decodificación de la información, así como la imposición de una serie de políticas que garantizan el acceso pertinente a la información.
- **Secure Controller:** Gestiona el flujo de ejecución interno dentro del dispositivo. Otras tareas corresponden con la verificación y ejecución de comandos para garantizar el correcto funcionamiento del dispositivo y la información que gestiona.
- **ROM:** Dentro de dicho componente, se almacena el firmware denominado TCG.
- **RAM:** Almacenamiento de información volátil dentro del dispositivo.
- **EEPROM:** Maneja información definida por el usuario. Dentro de este componente, se guardan por ejemplo las claves generadas por el dispositivo, así como una serie de certificados entre otros.
- **Asymmetric Key Generator:** Proporciona una serie de funciones orientadas al cifrado de la información, permitiendo el uso de claves RSA de hasta 2048 bits..
- **ACE:** Operaciones asimétricas de cifrado.
- **Motor Sha.1:** Es fundamental en uno de los puntos esenciales dentro de este proyecto. Utilizado para realizar mediciones de integridad dentro del dispositivo.
- **TRNG:** Generador de números aleatorios y claves..

- **Tick counter:** Se encarga de auditar los comandos acontecidos dentro del dispositivo.

Establecidos los principales bloques, a continuación se han desarrollado cada una de las partes utilizadas en este proyecto, entrando en mayor nivel de detalle en la generación de claves, así como la atestación del sistema.

2.1.2.2 Creación de claves:

Dentro del dispositivo, se gestionan las diferentes claves entre las cuales podemos encontrar las siguientes:

Endorsement Key: Más conocido bajo las siglas EK, hacen referencia a una clave única implementada para cada dispositivo. Esta clave está definida por el fabricante, de acuerdo a una condiciones seguras establecidas por el mismo, en este caso Infineon. Destacar que este tipo de clave se encuentra almacenada dentro del dispositivo, y no puede ser sustituida por una nueva, es única e intransferible. Normalmente son utilizadas para la creación de claves AIK.

Identity Keys: Las AIK son claves utilizadas para la firma de información. Está compuesto de una parte privada y una pública. El principal propósito de las mismas, es la firma de información generada en el dispositivo. En este caso, ha sido usada para la firma de los valores PCR con objeto de verificar la atestación o estado del sistema. Este tipo de claves son similares a las EK, con la peculiaridad de que puede haber múltiples almacenadas en un solo dispositivo, todas ellas definidas por el propio usuario.

Storage Root Key: SRK es otro tipo de clave que gestiona el dispositivo. De nuevo, se almacena dentro del mismo y no puede ser sustituida. Corresponde con una clave del tipo RSA formada por 2048 bits.

Signature Keys: Claves del tipo RSA utilizadas para la firma de información.

Storage Keys: Las claves de almacenamiento son utilizadas para encriptar una serie de elementos que forman la jerarquía de claves del dispositivo. Este tipo de claves son creadas en la iniciación del módulo seguro.

Una vez establecidas los tipos de claves que gestiona el dispositivo. Tanto la clave EK como la SRK se almacenarán en la memoria no volátil del dispositivo. Con ello lo que se consigue es que no se pueda tener acceso directo a las mismas, es decir, operaciones de sellado de información o firma será generada dentro del propio procesador criptográfico, cargando las claves necesarias almacenadas en ciertos índices de esta memoria.

Esta gestión de claves se realiza a través de software de acuerdo a la especificación del TSS en una serie de índices conocidos como ranuras o *slots*.

En caso de que se desee eliminar las claves almacenadas dentro del sistema, será necesario hacer una invalidación, con ello, lo que se pretende es el restablecimiento de la memoria del dispositivo, así como los valores PCR los cuales será necesario un reinicio del mismo. Al realizar dicha invalidación, el valor de la clave EK no se verá afectado. El proceso que sigue las operaciones efectuadas en el TPM es el siguiente:

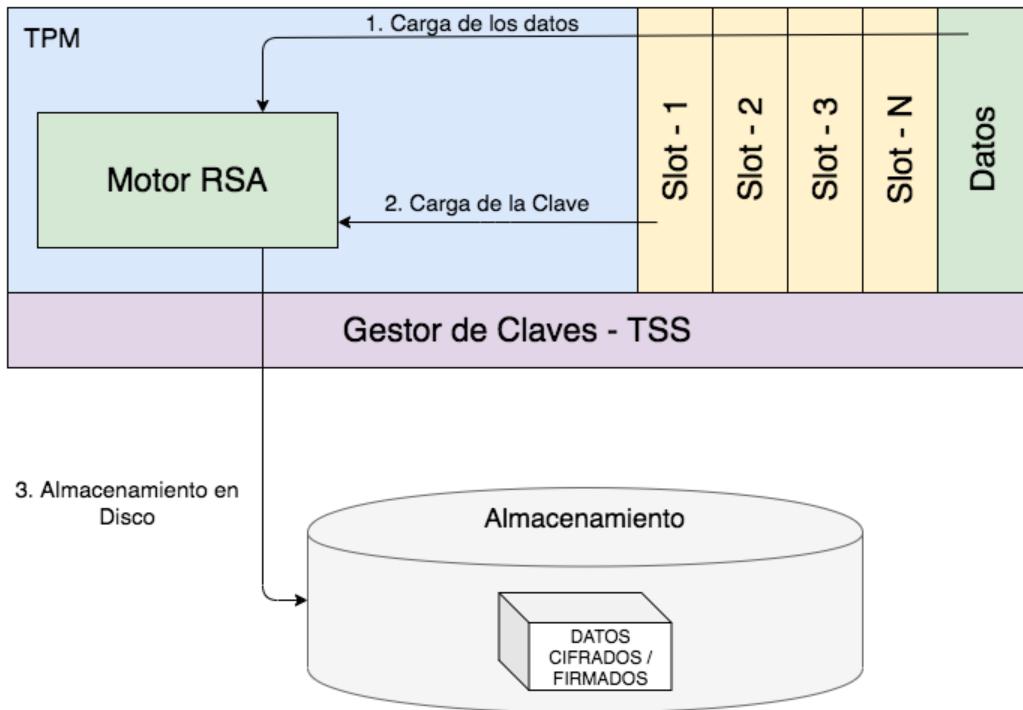


Figura 8: Cifrado de la información dentro del TPM.

2.1.2.3 Atestación del Sistema:

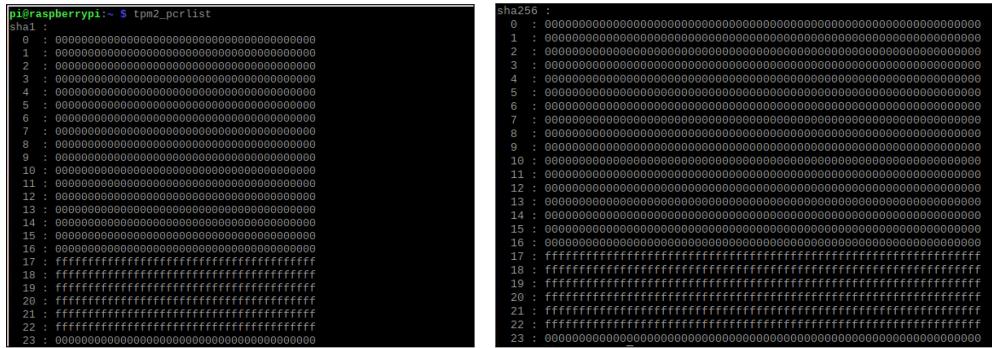
Como se ha mencionado anteriormente, el TPM proporciona una serie de funcionalidades, entre las cuales se encuentra por ejemplo el cifrado de información mediante el uso de claves asimétricas. En esta ocasión, el procesador ha sido usado para atestar el estado actual del sistema.

Este proceso toma el nombre de Sellado del inglés *Sealed*. Mediante esta operación, es posible sellar la información de acuerdo al estado actual de la plataforma. En caso de que al recibir el mensaje, este haya sufrido alguna modificación, la operación de descifrado no se podrá realizar, ya que la

generación de claves es diferente al estado anterior.

Puesto que se desea la atestación de forma remota de las plataformas, se ha optado por la función denominada *quote*. Esta operación, garantiza el sellado y firma de la información, mediante el uso de una clave AIK. Para ello, consultará unos bancos de registros denominados PCR.

Este registro corresponde con un área de memoria dentro del propio dispositivo. Gracias a la firma de los mismos, es posible certificar el estado del sistema actual. En el caso del TPM actual, hace uso de 24 registros. Se asignan a diversas capas del software, incluyendo entre ellas desde el código de arranque o inicialización del sistema, , así como el sistema operativo o aplicaciones en ejecución dentro del equipo. Normalmente, se suelen utilizar los índices impares para registrar el estado de los archivos de configuración, mientras que los pares se utilizan para registrar la ejecución del software.



(a) Sha1

```
pi@raspberrypi:~ $ tpm2_pcrlist
sha1 :
0 : 000000000000000000000000000000000000000000000000000000000000000
1 : 000000000000000000000000000000000000000000000000000000000000000
2 : 000000000000000000000000000000000000000000000000000000000000000
3 : 000000000000000000000000000000000000000000000000000000000000000
4 : 000000000000000000000000000000000000000000000000000000000000000
5 : 000000000000000000000000000000000000000000000000000000000000000
6 : 000000000000000000000000000000000000000000000000000000000000000
7 : 000000000000000000000000000000000000000000000000000000000000000
8 : 000000000000000000000000000000000000000000000000000000000000000
9 : 000000000000000000000000000000000000000000000000000000000000000
10 : 000000000000000000000000000000000000000000000000000000000000000
11 : 000000000000000000000000000000000000000000000000000000000000000
12 : 000000000000000000000000000000000000000000000000000000000000000
13 : 000000000000000000000000000000000000000000000000000000000000000
14 : 000000000000000000000000000000000000000000000000000000000000000
15 : 000000000000000000000000000000000000000000000000000000000000000
16 : 000000000000000000000000000000000000000000000000000000000000000
17 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
18 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
19 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
20 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
21 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
22 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
23 : 000000000000000000000000000000000000000000000000000000000000000
```

(b) Sha256

```
sha256 :
0 : 000000000000000000000000000000000000000000000000000000000000000
1 : 000000000000000000000000000000000000000000000000000000000000000
2 : 000000000000000000000000000000000000000000000000000000000000000
3 : 000000000000000000000000000000000000000000000000000000000000000
4 : 000000000000000000000000000000000000000000000000000000000000000
5 : 000000000000000000000000000000000000000000000000000000000000000
6 : 000000000000000000000000000000000000000000000000000000000000000
7 : 000000000000000000000000000000000000000000000000000000000000000
8 : 000000000000000000000000000000000000000000000000000000000000000
9 : 000000000000000000000000000000000000000000000000000000000000000
10 : 000000000000000000000000000000000000000000000000000000000000000
11 : 000000000000000000000000000000000000000000000000000000000000000
12 : 000000000000000000000000000000000000000000000000000000000000000
13 : 000000000000000000000000000000000000000000000000000000000000000
14 : 000000000000000000000000000000000000000000000000000000000000000
15 : 000000000000000000000000000000000000000000000000000000000000000
16 : 000000000000000000000000000000000000000000000000000000000000000
17 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
18 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
19 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
20 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
21 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
22 : ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff ffffff
23 : 000000000000000000000000000000000000000000000000000000000000000
```

Figura 9: Listado de índices PCR del sistema

Al inicializar el TPM en un dispositivo en concreto, este inicializa los bancos a 0, tal y como se puede ver en las imágenes 9a y 9b. Además, no es posible modificar directamente esta información, es decir, no se puede establecer un

valor deseado para cada uno de ellos. Dichos valores se modifican al realizar una operación conocida como *extend*, o extensión.

$PCR_VALUE[Index] =$ $DIGEST(PCR_OLD_VALUE \parallel DATA_EXTEND)$
--

Figura 10: Modificación de uno de los índices del sistema.

La formula de la figura 10, toma el valor antiguo establecido para dicho índice y realiza una concatenación con la nueva información definida en el *extend*, almacenando así el nuevo valor en dicho índice o registro. En la imagen que se adjunta a continuación, es posible obtener una posible configuración o uso de los registros PCR.

Número de índice PCR	Asignación
0	BIOS.
1	Configuración de la BIOS.
2	Opciones ROM.
3	Configuración de las opciones ROM.
4	MBR (Registro de arranque principal).
5	Configuración del MBR.
6	Transiciones de estados en el sistema y gestión de eventos.
7	Medidas específicas del fabricante de la plataforma.
8-15	Sistema operativo estático.
16	Depuración.
23	Soporte de aplicaciones.

Figura 11: Configuración PCR.

2.1.2.4 Especificación TPM 2.0:

Para poder implementar el protocolo seguro propuesto, se ha hecho uso de software establecido por el TCG (*Trusted Computing Group*), que no es más que una organización sin ánimo de lucro, encargada de desarrollar nuevos estándares relacionados siempre con el ámbito de la seguridad.

Dentro del repositorio ⁶, se proporciona un acceso a una serie de documentación para cada una de las funciones, determinando los parámetros necesarios a introducir para la correcta ejecución de las mismas. En este caso, se ha hecho uso de las funcionalidades provistas por el módulo denominado *tpm2-tools*. Se incluye una guía de instalación de este software como anexo a dicho documento 8.

Con respecto a las capas que componen este software, encontramos las siguientes:

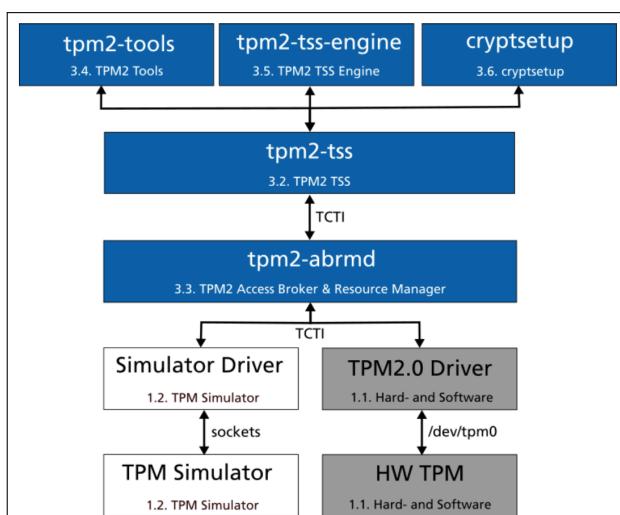


Figura 12: Capas TSS.

⁶<https://github.com/tpm2-software/tpm2-tools>

2.2 Estado del Arte:

El ámbito de la ciberseguridad corresponde con una tarea compleja, ligado a la gran cantidad de ataques que van surgiendo diariamente y con atacantes cada vez más preparados, está proponiendo un escenario difícil de abordar. A pesar de todo ello, se han realizado una serie de avances, proporcionando nuevos métodos que garanticen la seguridad en los dispositivos, así como en el proceso de comunicación, sin embargo, no se puede establecer una solución como infalible.

Por otro lado, con respecto a al TPM, a pesar de llevar bastantes años en el mercado y estar presente en un porcentaje elevado de dispositivos actualmente, no existen grandes avances en este ámbito. Las utilidades de este dispositivo siguen siendo limitadas y constituye una rama interesante a la hora de ofrecer seguridad mediante hardware y software. Pensamos que el principal motivo por el que esto está sucediendo está vinculado al hecho de que el uso de TPM para resolver problemas de seguridad no es un simple *plug and play*, sino que requiere de un diseño de la solución con conocimientos profundos de seguridad y del propio dispositivo.

Con el desarrollo de este trabajo, se ha conseguido aportar una nueva solución para la migración de agentes entre dispositivos previamente atestados por una entidad certificadora.

Capítulo 3

Metodología de trabajo:

Para el desarrollo de dicho proyecto, se han realizado una serie de iteraciones, fundamentadas en metodologías ágiles, en concreto, el modelo en espiral [5].

La elección de dicho modelo se basa principalmente en los requerimientos necesario para abordar dicho proyecto. Partimos de una implementación no definida en la actualidad. Además, se requiere un estudio previo tanto de las funcionalidades que ofrece el TPM, así como Jade, de los cuales no se poseen conocimientos previos antes de realizar dicho trabajo.

Gracias a este modelo, es posible alternar entre fases de estudio y desarrollo de forma ágil y eficaz. En el transcurso de dicho trabajo, se han ido desarrollando una serie de versiones, sobre las cuales se han trabajado con el fin de perfeccionar las mismas, a través de diversas mejoras.

Las fases de trabajo establecidas de acuerdo a este modelo, han sido las siguientes:

1. Planificación de los hitos a conseguir.
2. Estudio de la materia.
3. Reunión con el tutor para poner en común los conocimientos adquiridos y los hitos a realizar con los mismos.
4. Desarrollo y testeo de cada una de las nuevas funcionalidades implementadas.

5. Cumplimentación del informe.

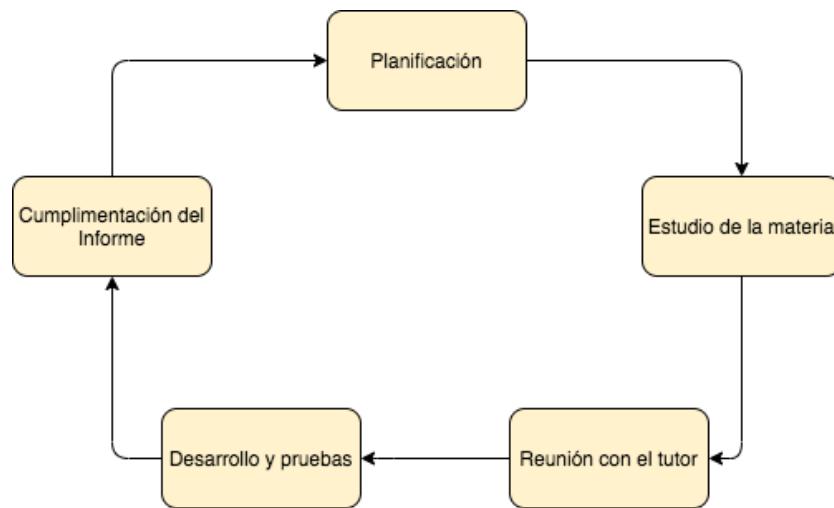


Figura 13: Fases que componen la metodología usada.

Una vez planificado la temática de dicho proyecto, y establecido un breve estudio del contexto en el cual se localizaba el mismo, se realizó una primera reunión con el tutor, con el objetivo de definir un objetivo final a conseguir una vez completado dicho trabajo.

A lo largo del proyecto, se han realizado dos versiones, con el objetivo de solucionar una serie de problemas acontecidos relacionados con el desarrollo y las capacidades que aportaba el TPM. Cada una de estas fases, se han determinado en puntos posteriores a este documento. Para la primera versión del protocolo se realizó tres iteraciones del ciclo propuesto en la metodología, mientras que para la versión definitiva correspondiente con la segunda versión, se han realizado un total de 5 iteraciones.

Dada la complejidad de este dispositivo, unido con el conocimiento avanzado requerido para el desarrollo de nuevos servicios en Jade, ha desembocado

todo ello en un arduo trabajo de investigación.

Uno de los aspectos fundamentales a tener en cuenta en base de establecer dicha metodología, se basa en la información disponible de cada uno de estos elementos. En el caso del TPM, la información era abundante, pero de elevada complejidad. Por otro lado, con respecto a Jade, totalmente lo contrario. Los datos recabados eran escasos para nuevos desarrollos, lo cual, unido al escaso soporte y apoyo actual de dicho framework, ha dado lugar a un proceso de desarrollo e ingeniería inversa, con el fin de determinar como llevar a cabo el nuevo protocolo, en base a servicios base pre-establecidos.

Capítulo 4

Desarrollo del Protocolo Seguro :

Con este trabajo, se ha conseguido determinar un nuevo protocolo de comunicación seguro entre agentes móviles. Dentro de este punto, se va a desarrollar el funcionamiento del mismo, así como las diversas fases de desarrollo acaecidas a lo largo del proyecto. El código de este trabajo se encuentra en un repositorio de github ⁷.

La gran mayoría de soluciones establecidas dentro del ámbito de la seguridad, se basan en el uso de claves seguras. Con este nuevo protocolo, se ha establecido una solución, a través de un enfoque dual de claves asimétricas y atestación de los valores del sistema en el cual se ejecuta el agente. Partiendo del contexto en el que existen una serie de agentes ya desplegados en diversas plataformas, procedemos a determinar cada una de las fases.

4.1 Fases de Desarrollo:

4.1.1 Primer Prototipo:

En un principio, se estableció que para realizar una migración del agente de forma segura hacia otra plataforma, se utilizó una de las funciones que ofrece el TPM denominado sellado (Seal). Dicha función se encarga de cifrar al agente, en base a los valores PCR comentados anteriormente, es decir, se realiza una atestación local del sistema, y posteriormente se procede a cifrar dicha información en base a estos valores. Gracias a esto, se consigue

⁷<https://github.com/IvanGarcia7/Jade TPM Security>

determinar que el destino al que se dirige tiene el mismo estado que el sistema de origen. Una primera aproximación de este protocolo, por tanto, quedaría de la siguiente forma:

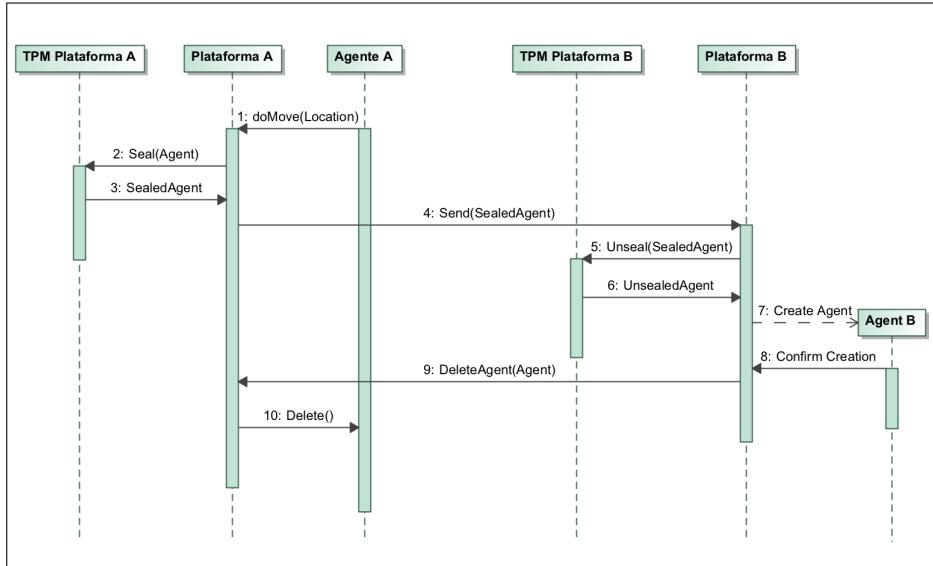


Figura 14: Primer prototipo del protocolo.

En base a una serie de problemas, que se explicarán en el siguiente punto 5, se ha procedido a desarrollar una nueva versión de este protocolo seguro.

4.1.2 Segundo prototipo:

Tras un estudio de la situación, así como de las funcionalidades que posee el TPM aplicadas a dicho contexto, se estableció un segundo prototipo. En esta ocasión, se ha realizado una separación de funcionalidades con respecto a las plataformas. Con dicha implementación, coexisten plataformas seguras, correspondientes a CA o entidades certificadoras por parte de la propia entidad, así como plataformas que deseen la migración remota de forma segura.

Una vez estudiado más a fondo dicha nueva versión, el protocolo se compone

de dos partes bien diferenciadas, la primera de ellas, corresponde con la etapa de validación. Los pasos que se ejecutan en dicha fase son los siguientes:

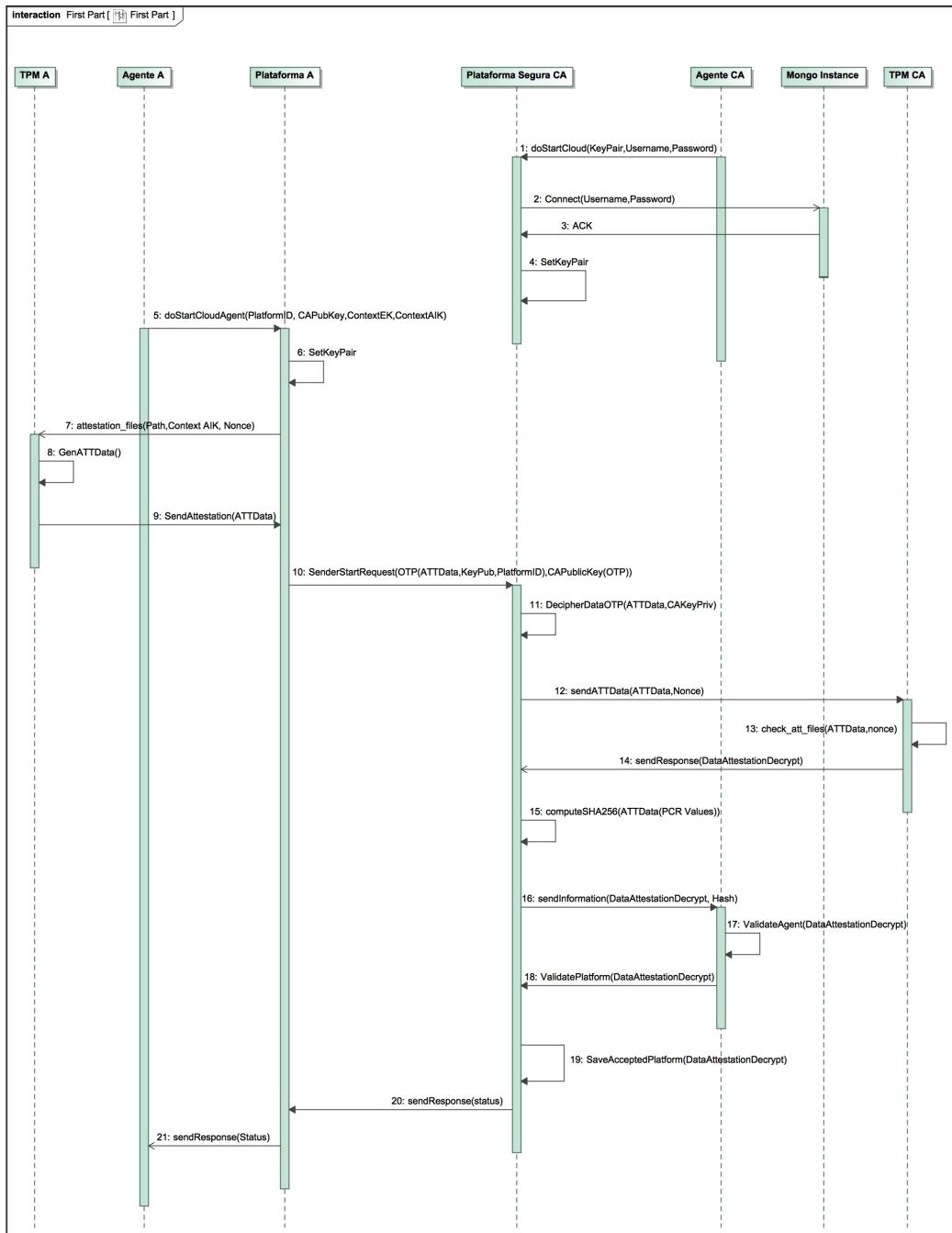


Figura 15: Primera parte del protocolo: Etapa de Validación

Tal y como se puede apreciar en la imagen, el protocolo comienza con una etapa de validación previa. Una entidad despliega una plataforma segura, encargada del proceso de validación de los agentes. Toda la información validada en las diversas etapas que compone el protocolo, es almacenada en una base de datos no relacional. En esta ocasión, se ha elegido MongoDB para la creación de un clúster. Para poder obtener acceso a esta base de datos, es necesario establecer tanto el nombre como la contraseña. Validado los credenciales de acceso, se creará un par de claves RSA de 2048 bits en caso de que no haya sido configurada previamente y la plataforma estará lista para aceptar peticiones externas.

Para facilitar el manejo y la realización de pruebas, se ha diseñado una interfaz gráfica *responsive*, la cual permite realizar acciones tanto desde la CA, así como de cada plataforma. Sin embargo, el uso de Jade está enfocado como servicio lanzado a través de comandos desde terminal.

En el panel superior, se han habilitado una serie de opciones. Las funciones establecidas son las siguientes:

- **Username:** Al presionar sobre dicho botón, se muestra la ventana de inicio de la plataforma.
- **Request List:** Accede a un repositorio en MongoDB, y carga todas las plataformas validadas previamente. También es posible visualizar aquellas que están pendientes, además de determinar una serie de valores Hashs aceptables para validar la plataforma directamente.
- **CRUD:** Permite validar o eliminar plataformas establecidas en el repositorio de MongoDB. Es posible además determinar el tiempo máximo de espera a la hora de procesar mensajes provenientes de plataformas

externas involucradas en el proceso de migración.

- **Information:** Dentro de esta ventana, se muestra el intercambio de mensajes acontecido en el proceso de atestación.
- **Clear:** Limpia el área de mensajes de la ventana activa.
- **Exit:** Termina la ejecución del agente y cierra la plataforma.

Al iniciar la plataforma correspondiente con la CA, aparece la siguiente interfaz gráfica diseñada para este protocolo en concreto:

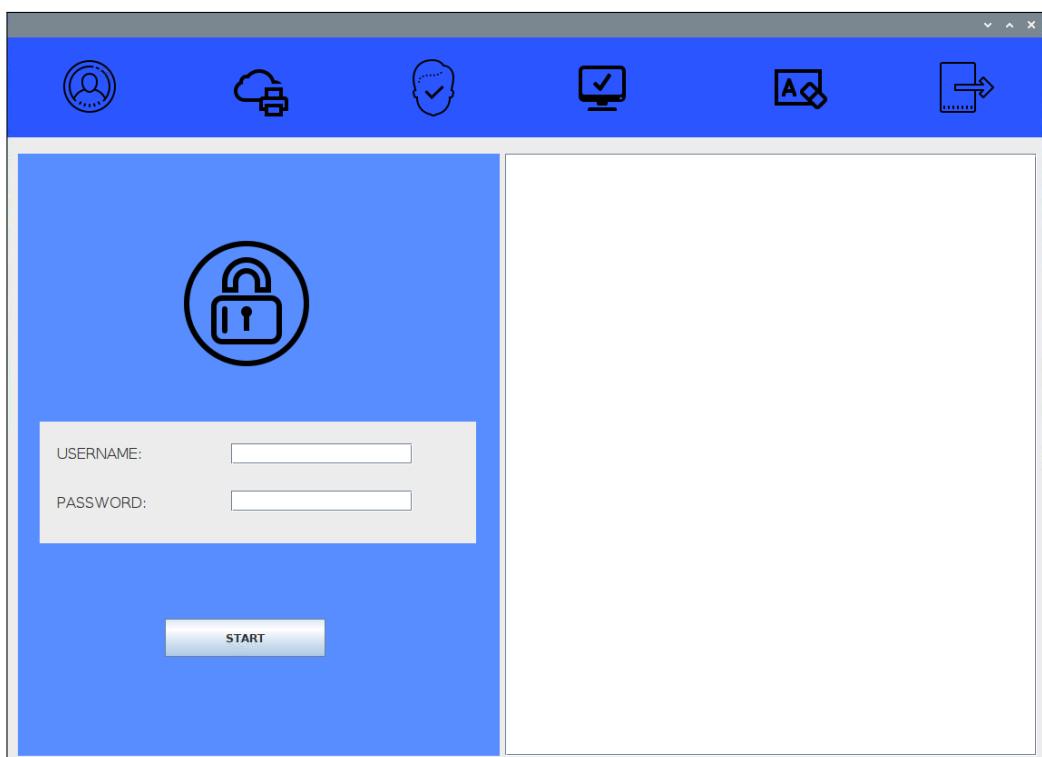


Figura 16: Interfaz de ayuda para la CA

Con el objetivo de ver en todo momento que procesos se está ejecutando, se han establecido una serie de *Logs*. Comenzando por el área de inicio, una

vez que se ha introducido el usuario y contraseña que permite acceder a la instancia de Mongo, es posible ver en el panel derecho la carga de claves (pública y privada) en caso de que no se hubiera inicializado previamente, con el objetivo de posteriormente poder cifrar los mensajes intercambiados en la comunicación con cada una de las plataformas involucradas en el proceso de migrado. Ambas claves son mostradas con el fin de obtener una visión del funcionamiento del protocolo al iniciar la plataforma.

```
INITIALIZING THE KEY PAIR RECEIVE
NAME OF THE CONTAINER: Main-Container

*****PUBLIC*****
PUBLIC KEY: Sun RSA public key, 2048 bits
params: null
modulus: 249212033900203196114864708805677802393566926593065177358493815890475193 ...
public exponent: 65537
*****PUBLIC*****

*****SECRET*****
PRIVATE KEY: SunRsaSign RSA private CRT key, 2048 bits
params: null
modulus: 249212033900203196114864708805677802393566926593065177358493815890475193 ...
private exponent: 23420691182000114518379895753620857724983109656825117626206559065 ...
*****SECRET*****
```

Figura 17: Inicialización de la CA

Para realizar dicho proceso, el agente ha tenido que generar un comando vertical, el cual ha sido atendido en el sumidero denominado *Command-SourceSink*. Posteriormente, se procesa la petición creando un comando Horizontal y enviándolo previamente al AMS del contenedor principal. Una vez que las claves han sido inicializadas previamente, ya es posible atender peticiones.

Desde el punto de vista del resto de plataformas, al ejecutar un agente, la interfaz mostrada variará con respecto a la del CA, ofreciendo otras funciones totalmente distintas.

- **Start:** Inicializa la plataforma, estableciendo un par de claves RSA, en caso de que no se hubiera definido previamente.
- **Migrate:** De acuerdo a la dirección de la plataforma, así como su identificador en las casillas denominadas *AID Platform* y *Address Platform* respectivamente, inicializa el protocolo de comunicación.
- **Clear:** Limpia el área de mensajes de la ventana activa.
- **Exit:** Termina la ejecución del agente y cierra la plataforma.

A la hora de inicializar la plataforma, es necesario establecer dos contextos, en el cual el TPM almacene la clave EK del dispositivo, así como la parte privada del AIK generado para la firma. Con ello lo que se pretende, es poder facilitar la simulación de escenarios utilizando un único dispositivo. El TPM almacena esta información dentro de la memoria no volátil. Una de las ventajas que ofrece este dispositivo, es la seguridad en cuanto a almacenamiento se refiere, pues no es posible acceder de forma directa a dicha información, los métodos y funciones como por ejemplo la firma de la información, se ejecutan en el interior del TPM.

Por otro lado, el proceso de carga de las claves RSA son similares a como se establece para la CA. Una vez presionado el botón de inicio, el *log* ofrece la siguiente información:

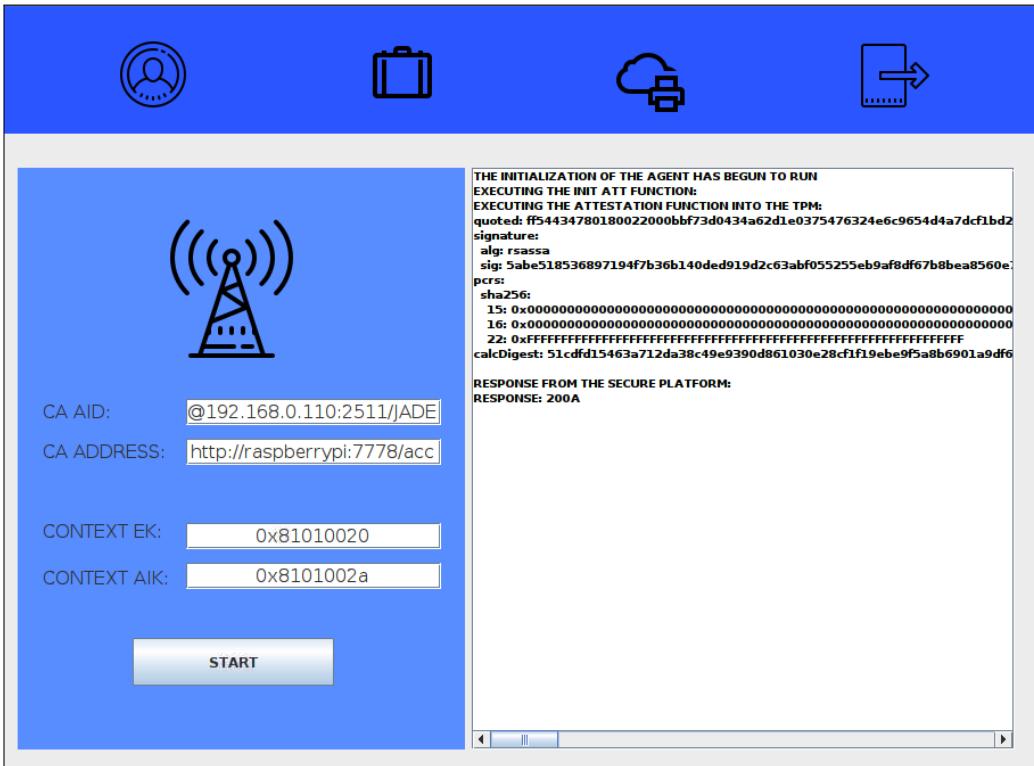


Figura 18: Atestación inicial de la plataforma

Tal y como se puede ver en la anterior imagen, el agente envía una petición a la plataforma, y será esta quien contacte con el TPM para generar una atestación inicial del sistema. En este caso, se han establecido los índices 15, 16 y 22 como prueba. Accediendo a la terminal del dispositivo e insertando el comando `tpm2_pcrlist`, es posible obtener un listado con los hash de estos valores.

En una primera atestación inicial, no se establece ningún tipo de reto, de esto se hablará con mayor detalle en puntos posteriores 5. Destacar que las funciones utilizadas en el TPM, generan la información en una serie de archivos. Para facilitar el manejo de los mismos, se serializa y se almacena en memoria de forma automática, con el fin de agilizar los procesos a través

de una función. En caso de que sea necesaria en ciertos procesos, generará un directorio temporal y los almacenará en los archivos establecidos en el listado anterior. Una vez que el *quote*, nombre que recibe el comando para firmar los datos que contienen el estado actual del sistema haya finalizado, se generan tres tipos de archivos:

- Message.File: Dentro del mismo se almacena el mensaje de atestación que compone los datos firmados en base a los valores PCR seleccionados.
- Signature.File : Registra la firma.
- PcrOutput.File: Almacena el listado de los valores PCR seleccionados.

En caso de requerir mayor información acerca de dicho comando, consultar la documentación oficial ⁸.

Llegados a este punto, la plataforma serializa la información generada en la atestación, creando un paquete denominado *RequestSecureATT*. El contenido del mismo es el siguiente:

$$\begin{aligned} & \textit{SymmetricKey}(\textit{PublicKeyAgent}, \textit{PlatformLocation}, \textit{AttestationData}), \\ & \textit{CAKeyPub}(\textit{SymmetricKey}) \end{aligned}$$

Figura 19: Contenido del paquete inicial generado por la plataforma para iniciar el proceso de atestación.

⁸https://github.com/tpm2-software/tpm2-tools/blob/master/man/tpm2_quote.1.md

No es posible utilizar la clave pública de la CA directamente, puesto que el tamaño de los archivos supera los 256 bytes, es por ello, por lo que previamente, se ha generado una clave de uso único simétrica para el cifrado de estos archivos. Dentro del mensaje, se incluye este OTP cifrado con la clave pública de la CA.

Se ha implementado un modo de operación diseñado para algoritmos simétricos, modificando así la forma de operar los bloques, aportando confidencialidad. Para ello, previamente, se especifica una inicialización de vector, para acto seguido, generar dicha clave. Este modo de operación toma el nombre de *Cipher Block Chaining* (CBC). Una vez que el contenido se ha cumplimentado correctamente, se serializa el mensaje y se envía la información a la CA.

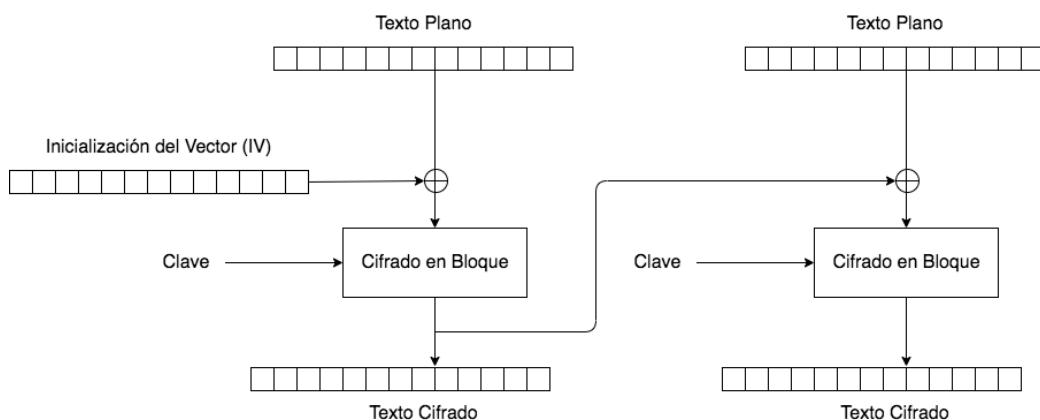


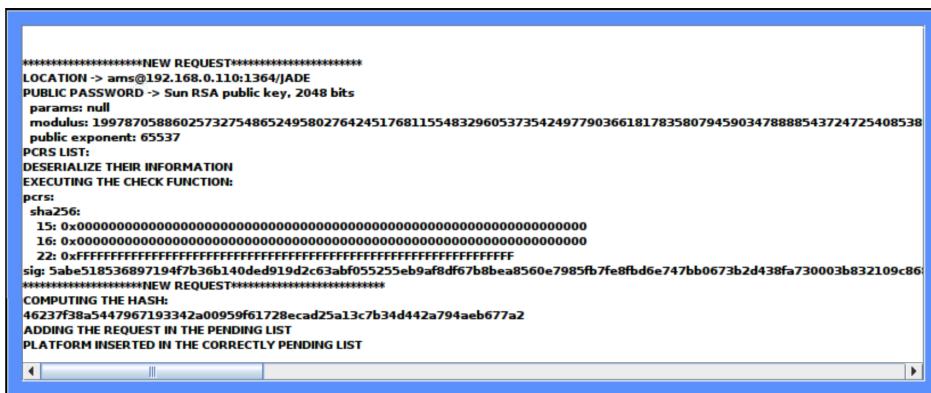
Figura 20: Cipher Block Chaining Mode. Fuente:⁹

En esta etapa, se inicia el proceso para comunicarse con la CA. Para poder enviar la información, es necesario contactar previamente con el AMS del contenedor principal. Este por tanto, será el encargado final de transmitir

⁹https://upload.wikimedia.org/wikipedia/commons/thumb/9/9d/CFB_encryption.svg/601px-CFB_encryption.svg.png

el mensaje, para ello, de nuevo, es necesario crear una serie de comandos verticales y horizontales, con el objetivo de comunicarse con el mismo.

Volviendo de nuevo al área de información de la CA, vemos como efectivamente, se ha recibido la petición por parte de la plataforma. El *log* muestra tanto la dirección de la plataforma, así como su clave pública. Además, es posible ver el valor de cada uno de los índices PCR establecidos.



```
*****NEW REQUEST*****
LOCATION -> ams@192.168.0.110:1364/JADE
PUBLIC PASSWORD -> Sun RSA public key, 2048 bits
params: null
modulus: 19978705886025732754865249580276424517681155483296053735424977903661817835807945903478888543724725408538
public exponent: 65537
PCRS LIST:
DESERIALIZE THEIR INFORMATION
EXECUTING THE CHECK FUNCTION:
pcrs:
sha256:
 15: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 16: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 22: 0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFFF0xFFFFFFFFFFFFFF
sig: 5abe518536897194f7b36b140ded919d2c63abf055255eb9af8df67b8bea8560e7985fb7fe8fb6e747bb0673b2d438fa730003b832109c86
*****NEW REQUEST*****
COMPUTING THE HASH:
46237f38a5447967193342a00959f61728ecad25a13c7b34d442a794aeb677a2
ADDING THE REQUEST IN THE PENDING LIST
PLATFORM INSERTED IN THE CORRECTLY PENDING LIST
```

Figura 21: Petición recibida en la CA.

Se genera un valor hash de forma automática en base a estos índices, para posteriormente almacenar la petición. En el área de plataformas registradas en el sistema, si presionamos sobre el botón peticiones pendientes, es posible localizar la solicitud de la plataforma, viendo como información la dirección de la misma, así como el código hash generado en base a los valores PCR, correspondientes con el estado del sistema.

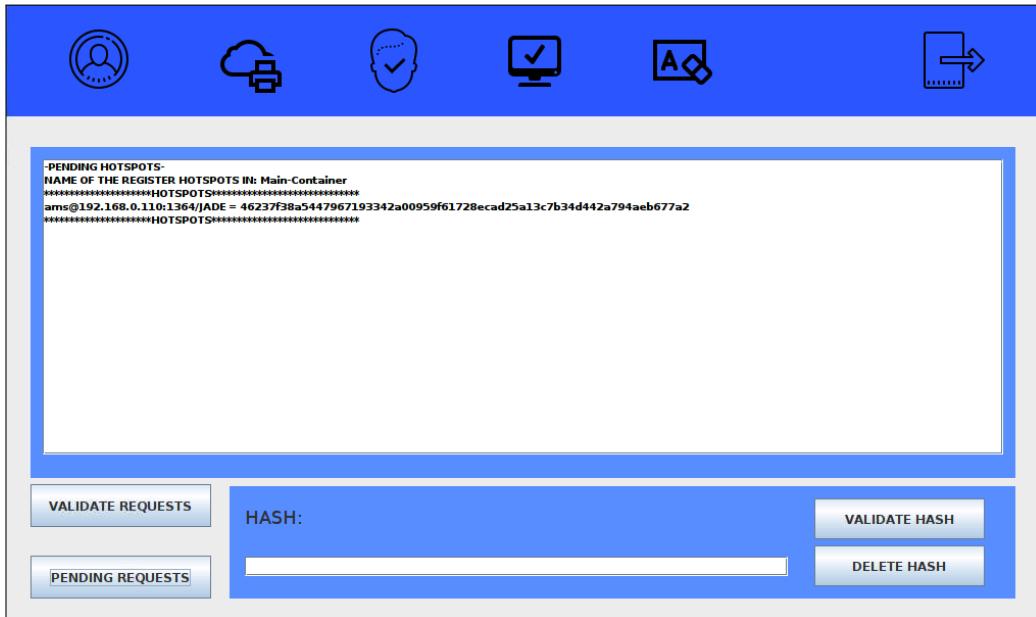


Figura 22: Listado de peticiones pendientes.

En el caso de que el hash coincida con uno de los valores almacenados previamente en la base de datos, la plataforma llevará a cabo la verificación de forma inmediata. Para ello, se ha habilitado un campo en el cual se permite la introducción o eliminación de ciertos valores. Dichos hashes están almacenados en un nuevo clúster de MongoDB designado para ello.

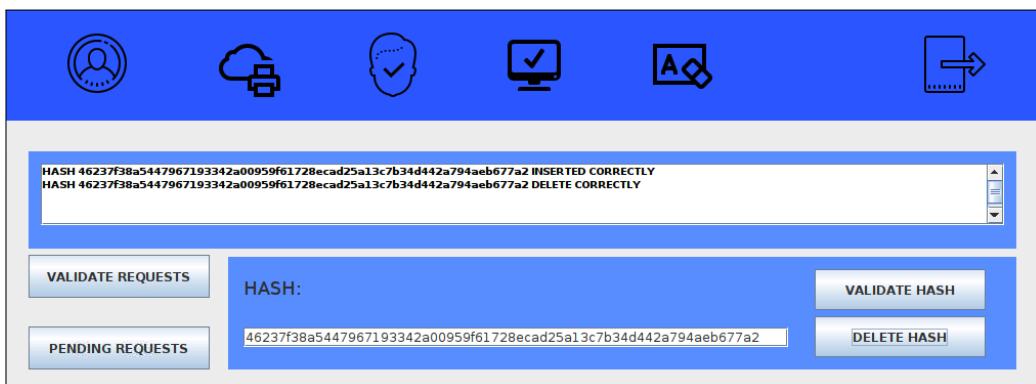


Figura 23: Introducción y eliminación de un Hash.

En caso contrario, solo resta que el responsable dentro de la CA, se encargue de verificar dicho hash, con el objetivo de finalmente incluirlo en el listado de plataformas confiables. Para realizar dicha acción, basta con dirigirse al área de gestión. Dentro de la misma, introduciendo el identificador de la plataforma, podrá validar nuevas solicitudes o eliminar peticiones ya incluidas dentro de la instancia de MongoDB.

A continuación, se adjunta un ejemplo de validación, mostrando por pantalla la información almacenada para dicha petición, la cual equivale a las claves públicas de cifrado, firma, localización y hash inicial de la plataforma:

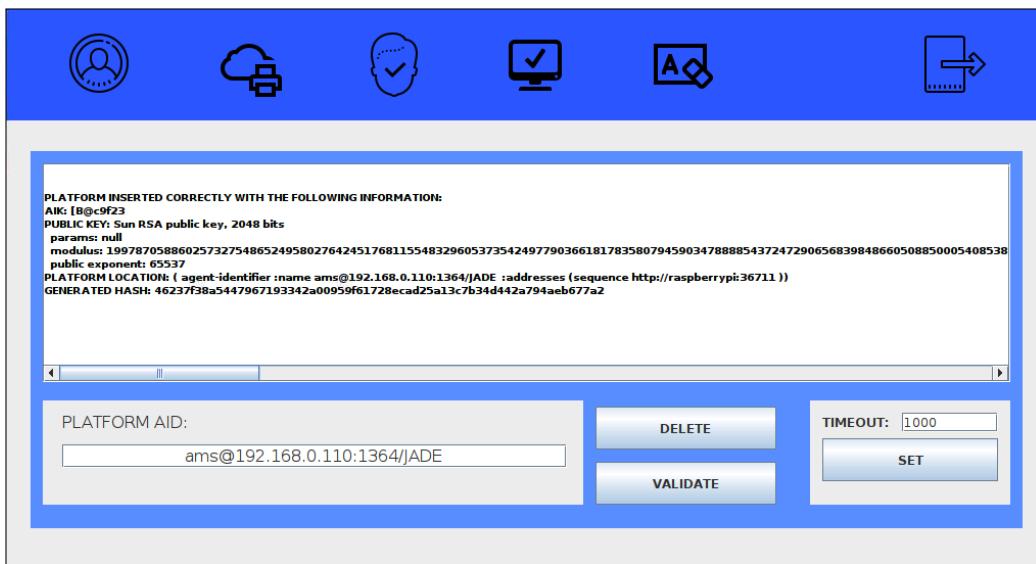


Figura 24: Validación de una de las peticiones.

Al validar una petición, vemos la información establecida para la misma. Para almacenar estos datos en el clúster de Mongo, es necesario serializarlos previamente. Dentro de MongoDB Compass, podemos visualizar el registro generado para dicha plataforma:

```

PUBLIC_AIK: Binary('r00ABXVyAAJbQqzzF/gGCFTgAgAAeHAA...', 0)
PUBLIC_KEY: Binary('r00ABXNyABRqYXZhLnNlY3VyaXR5Lktl...', 0)
PLATFORM_LOCATION: Binary('r00ABXNyABRqYWRLmNvcUUUGxhdGZv...', 0)
SHA256: "46237f38a5447967193342a00959f61728ecad25a13c7b34d442a794aeb677a2"

```

Figura 25: Registro almacenado en el clúster de MongoDB.

Además de validar una plataforma, también es posible definir el tiempo máximo de procesamiento de un paquete. Cuando se inicia el protocolo para atestar a ambas plataformas, la CA genera un límite de tiempo, determinando el retardo máximo a la hora de procesar las solicitudes de respuesta. En caso de que dicho tiempo haya sido superado, la petición se descartará. Este tiempo, está definido en milisegundos.

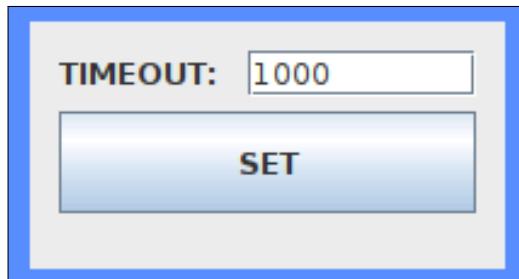


Figura 26: Casilla designada para el tiempo máximo de retardo.

Dirigiéndonos de nuevo al área de peticiones, si en esta ocasión presionamos sobre las solicitudes aceptadas, podemos ver que ahora la plataforma si está establecida como segura con un valor de estado definido.



Figura 27: Listado de Plataformas y Hashes Validados.

El proceso descrito anteriormente, corresponde con la primera etapa, a continuación, se muestra la segunda parte del protocolo. En esta parte, la CA inicia un proceso de atestación a ambas plataformas, tanto origen como destino, antes de proceder con la migración del agente, comprobando previamente el estado de estos sistemas.

Para poder llevar a cabo esta parte, se han establecido una serie de criterios, con el objetivo de garantizar la seguridad en todo momento. Cada uno de estos pasos se explicaran a continuación. Esta segunda etapa del protocolo es posible dividirla en dos grandes bloques. El primero de ellos correspondiente a la atestación previa de las plataformas, y en segundo lugar al proceso de migrado. La ejecución que se va a explicar a continuación, es totalmente invisible de cara al usuario. Se adjunta la imagen correspondiente a la segunda parte del protocolo.

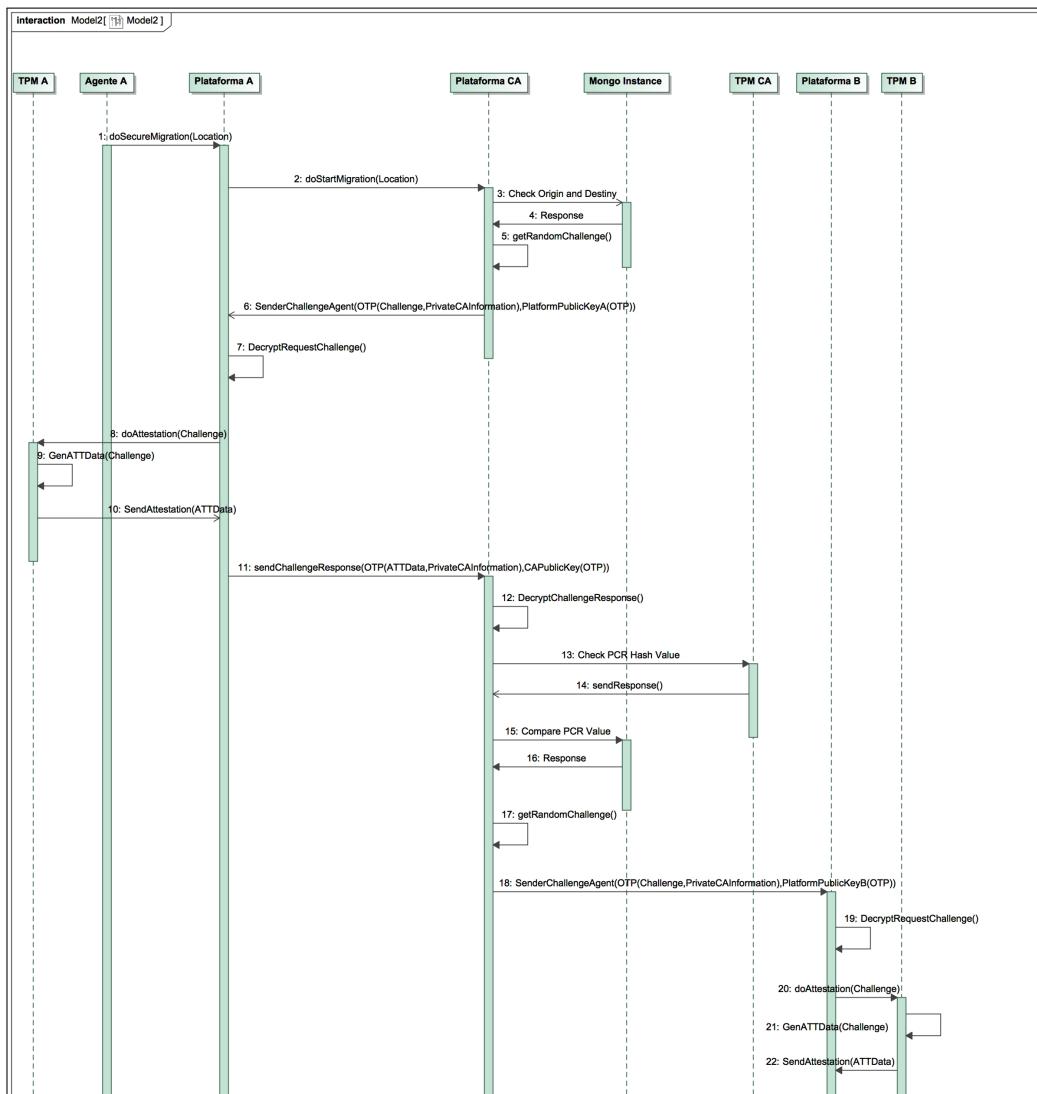


Figura 28: Segunda Parte del Protocolo Seguro.

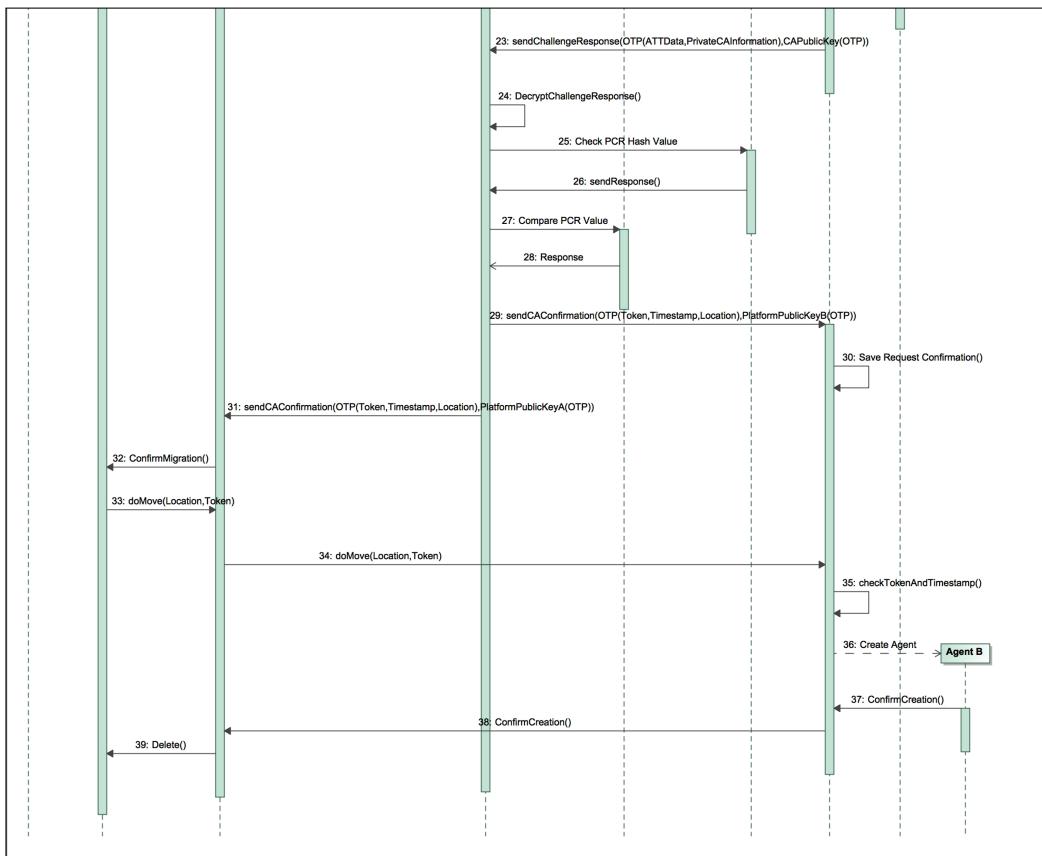


Figura 29: Segunda Parte del Protocolo Seguro.

Cuando un agente decide migrar, con simplemente ejecutar la función de migración, el intercambio de mensajes necesarios se ejecutará en segundo plano.

Partiendo de un agente localizado en una plataforma previamente registrada en la CA, este decide migrar. Se ha establecido un área, en la cual es posible determinar las plataformas por las que se desea migrar sucesivamente, con el objetivo de realizar una tarea en cada una de ellas. Introduciendo respectivamente tanto el identificador de la plataforma, así como la dirección de la misma y el servicio que desea ejecutar, al presionar sobre el botón para

añadir, se define la ruta a seguir. También es posible eliminar una ruta introducida por error, para ello, del mismo modo, se establece la información necesaria para identificar la plataforma y posteriormente, se presiona sobre el botón de borrado.

En el área derecha de la siguiente imagen, se muestra el estado del proceso de migración, en base a una serie de *logs* establecidos. Al presionar sobre el botón de migración, el sistema selecciona la primera ruta introducida. El resto de rutas, se almacenan en una lista dentro del propio agente, para realizar las migraciones de forma iterativa.

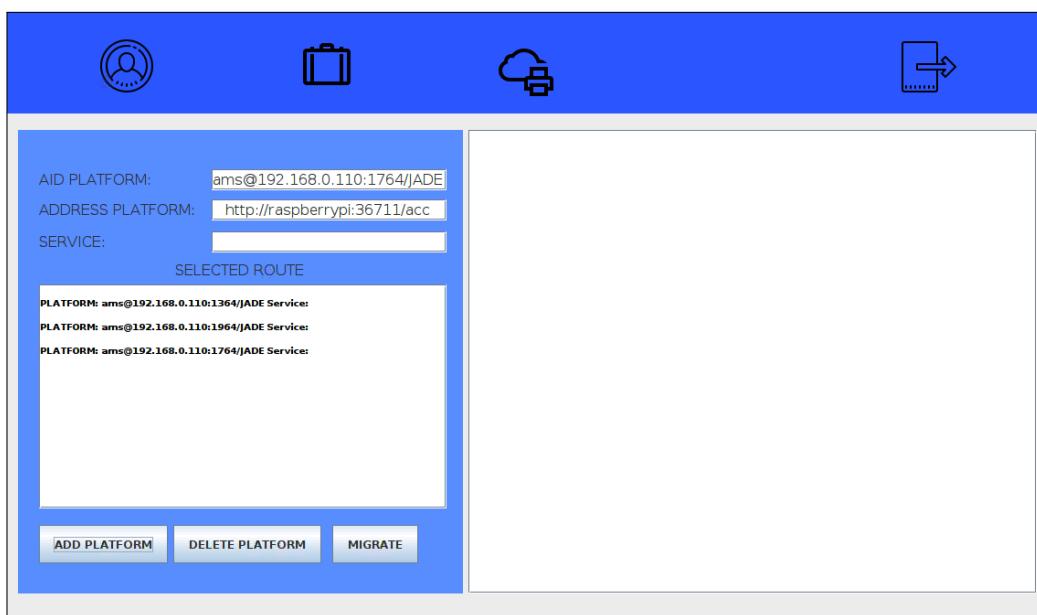


Figura 30: Rutas de Migración definidas.

En primer lugar, envía al CA una petición para migrar al primer destino seleccionado. Dicho mensaje tiene la siguiente forma:

```
SymmetricKey(PlatformLocationOrigin, PlatformLocationDestiny,  
AIDAgent), CAKeyPub(SymmetricKey)
```

Figura 31: Contenido del mensaje enviado a la entidad certificadora para comenzar el proceso de migrado seguro.

Es decir, genera un paquete, en el cual se incluye la localización a la que desea migrar. Esta información se cifrará con una clave simétrica generada, ya que el tamaño excede el límite de bytes permitido tal y como se ha comentado anteriormente.

Los procesos que se ejecutan en la CA, a excepción de la primera atestación de la mano del responsable en caso de que el valor hash no esté definido como seguro, son totalmente automáticos. Cuando esta reciba una nueva petición de migración, en primer lugar, comprueba que el estado de la plataforma es el aceptado en un principio. En un contexto en el cual existan multitud de plataformas que soliciten migrados concurrentemente, sería costoso establecer un registro de estas peticiones, para ello, como posible solución, se ha generado un paquete denominado *PrivateInformationCA*, el cual contiene información firmada con la clave pública de la CA. Dentro de la misma, podemos encontrar los siguientes datos:

```
SymmetricKey(PlatformLocationOrigin, PlatformLocationDestiny,  
AIDAgent), CAKeyPub(SymmetricKey)
```

Figura 32: Información privada cifrada con la clave pública de la entidad certificadora.

Almacenando por ello, la plataforma desde la cual proviene la petición, el destino solicitado, una marca de tiempo, un *challenge* o reto, y un bit de validación. Dicha información, se envía a la plataforma origen, junto con el reto. El bit de validación, es usado con el fin de conocer el proceso de atestación, tomando el valor 0 si aún no se ha atestado la plataforma origen, o 1 en caso contrario.

La CA recibe la petición y acto seguido comprueba si la plataforma se encuentra registrada. En caso de que así sea, comienza el proceso de atestación, tanto de la plataforma origen como destino, para ello, genera un *challenge*, es decir, una cadena de 32 bits, además de un paquete de información privada. Desde el punto de vista de la plataforma, recibe dicha petición, de la cual solo utilizará el *Challenge* para generar los nuevos valores de atestación del sistema. Al realizar la atestación, tal y como se establecía anteriormente, se generan una serie de archivos que posteriormente necesitan ser serializados.

El paquete con la información queda de la siguiente forma:

SymmetricKey(ATTData), CAPublicKey(OTP), PrivateInformationCA

Figura 33: Paquete de información enviada por la entidad certificadora a la plataforma para proceder con la atestación.

Llegados a este punto es donde entra en juego la marca de tiempo. La CA descifra la información privada generada en un principio, y recoge la marca de tiempo en la cual se creó. Posteriormente, se genera una nueva con el tiempo actual, y verifica que esta no supere un margen establecido por el sistema. Imponiendo estas medidas, por tanto podemos determinar que los valores de la atestación son actuales, y que además no ha sido posible

realizar un ataque de repetición debido al margen de tiempo definido. En el capítulo 5, se entrará en mayor detalle los beneficios conseguidos con dicha implementación.

Además, el TPM garantiza que la información no ha sido modificada, puesto que a la hora de realizar estos archivos de atestación, el proceso tiene lugar en el interior del dispositivo, utilizando la clave AIK privada al inicializar la plataforma, almacenada en memoria no volátil.

Una vez que la información ha sido correctamente verificada, se genera un hash del listado de valores PCR y se compara con los ya almacenados. Si estos coinciden, el proceso de migración continúa, y en este caso, se realizan los mismos pasos pero por parte de la plataforma a la que se desea migrar, para ello, se genera un nuevo reto, y se cambia el bit de validación incluida en la parte privada.

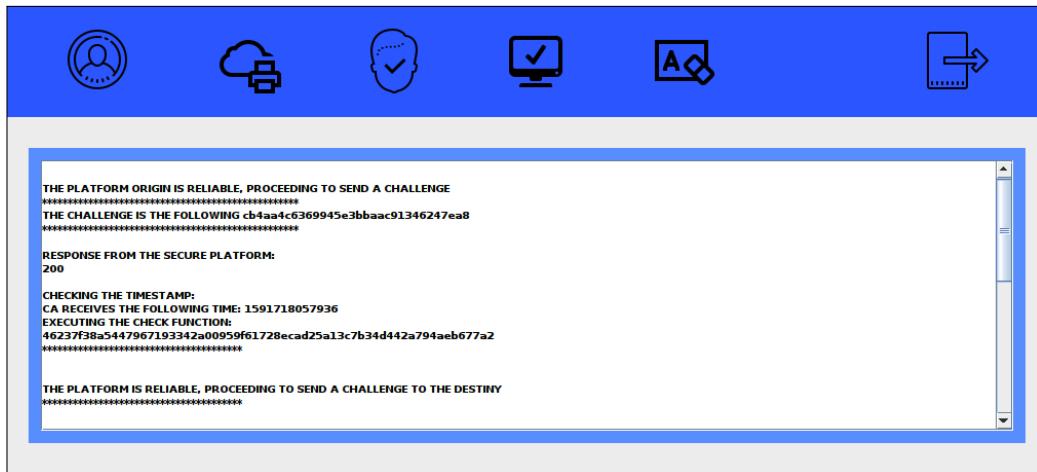


Figura 34: Ejemplo de Atestación correcta de la plataforma origen.

En caso contrario, la CA muestra en el área de mensajes que la plataforma se encuentra comprometida y la descarta como plataforma segura. Si en un

futuro se desea tener en cuenta a la misma, deberá de nuevo ser verificada manualmente.

En el momento en el que la CA verifique que ambos sistemas poseen el mismo estado, acorde con los valores validados por un responsable previamente, envía un mensaje tanto a la plataforma origen como destino, estableciendo una marca de tiempo, en el cual dicho proceso de migrado es válido, así como un identificador y la clave pública de la plataforma con la que se desea comunicar respectivamente.

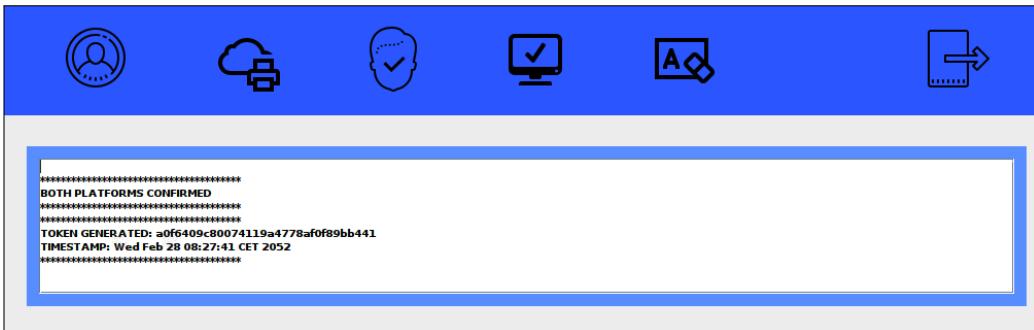


Figura 35: Mensaje mostrado tras validar ambas plataformas involucradas en el proceso de migración.

Por parte de la plataforma destino, tanto el identificador, como el tramo de tiempo válido, y la clave, los almacenará en un HashMap. En caso de la plataforma origen, comienza con el proceso de migrado. Utilizando el complemento que permite la migración inter-plataforma, ha sido necesario efectuar ciertas modificaciones.

Para llevar a cabo este proceso, en primer lugar se crea un jar del agente, que debe ser cifrado con la clave pública de la plataforma destino. Además, en el mensaje se debe incluir el identificador cedido por la CA. Una vez que la plataforma destino recibe el mensaje con la información necesaria, en

primer lugar comprueba que posea un identificador válido y que se cumpla la marca de tiempo establecida. En caso de que alguna de estas premisas no llegue a satisfacerse, el proceso de migración finaliza, impidiendo el traslado del agente. Por el contrario, si ambos requisitos se cumplen, el agente será descifrado y ejecutado en la plataforma.

Una vez recibido con éxito el mismo, se envía una confirmación para eliminarlo de la plataforma origen finalmente. Uno de los problemas encontrados en esta parte, corresponde con la migración entre plataformas sin haber obtenido un permiso previo de la CA. Este problema se desarrollará con mayor profundidad en el siguiente punto 5.

Por otro lado, en el agente desplegado para la inicialización de la plataforma destino, es posible ver el proceso que está teniendo lugar a lo largo del protocolo.

```
PROCESSING THE RESPONSE IN THE AGENT
PLATFORM RECEIVE A ZONE 1 REQUEST TO ATTESTATE THE ORIGIN

EXECUTING THE ATTESTATION FUNCTION INTO THE TPM:
*****CREATING THE ATTESTATION FILES WITH THE FOLLOWING INFORMATION:
SERVER TOKEN: 6a713c0d6a1b48bfad631e47ea39ca48

RESPONSE FROM THE SECURE PLATFORM:
RESPONSE ATTESTATION REQUEST: 200

PROCESSING THE RESPONSE IN THE AGENT
PLATFORM RECEIVE A CONFIRMATION DESTINY SECURE MOVE
***** CHECKING THE CONFIRM LIST *****
```

Figura 36: Información mostrada en la plataforma destino.

Capítulo 5

Resultados:

Dentro de este capítulo, se especifican en primer lugar los problemas acontecidos a lo largo del desarrollo, así como una serie de soluciones establecidas para la resolución de los mismos.

5.1 Problemas y soluciones desarrolladas:

5.1.1 Problemas de sellado en el primer prototipo:

Tras proceder al estudio de este primer prototipo, partíamos de una serie de premisas, destacando el ámbito restrictivo de dicho protocolo. Tal y como se planteaba, era necesario que ambos estados del sistema fueran exactamente idénticos, con lo cual reduciría considerablemente los ámbitos de aplicación del mismo.

Por ejemplo, en el caso de que entidades seguras dispusieran de diversos sistemas operativos o se realizaran cambios en el mismo, el proceso de migrado no sería posible, debido al sellado impuesto en un principio.

Para realizar dicho sellado, es necesario ejecutarlo directamente con la clave EK del dispositivo. Para su posterior descifrado, será necesario de nuevo dicha clave. Llegados a este punto, se decidió descartar esta implementación, puesto que esta clave pertenece única y exclusivamente a cada TPM. Además, su ámbito no está orientado al cifrado directo de la información, sino que se utiliza para generar claves AIK.

5.1.2 Atestación del sistema no actual:

A la hora de proceder a realizar la atestación del sistema en base a los valores PCR del mismo, las funciones que permiten actualmente dicho proceso almacenan los resultados en una serie de archivos. Tras realizar un estudio, esta situación puede suponer una posible vulnerabilidad del sistema.

Un atacante, no podría modificar los valores de la atestación como tal, puesto que estos se establecen en el interior del TPM, sin embargo, cabe la posibilidad de que obtenga los archivos de la atestación inicial del sistema. A la hora de realizar la atestación establecida por la CA, podría entregar valores desactualizados, permitiendo así la migración por parte de una plataforma comprometida. Se ha definido la existencia de un *challenge* o reto, correspondiente con una cadena de caracteres de 32 bits. Un posible ejemplo sería el siguiente:

fd0b592700734454b8a44b6d73f89e87

Figura 37: Ejemplo de reto generado por la entidad certificadora.

Gracias al mismo, es posible resolver este problema. A la hora de realizar la atestación, se introduce dicho *challenge*, obteniendo una firma del sistema de acuerdo a este valor. Cuando la CA reciba el mismo, en caso de que no se haya utilizado el mismo, la firma no podrá verificarse, por ello, rechazará la migración. Se consigue así evitar ataques de repetición por parte de una plataforma comprometida.

Para un entorno en el cual se han realizado 100 pruebas, el tiempo medio oscila en torno a los 579.67 ms, tal y como muestra la línea de tendencia. Puesto que el reto posee un tamaño de 32 bytes, habrá 2^{256} combinaciones posibles, las cuales, ligados a un tiempo medio de 580 ms aproximadamente en un entorno distribuido de raspberry junto con el límite de tiempo impuesto en la CA, no sería posible llevar a cabo este tipo de ataques. De acuerdo a estudios realizados, se han obtenido los siguientes resultados:

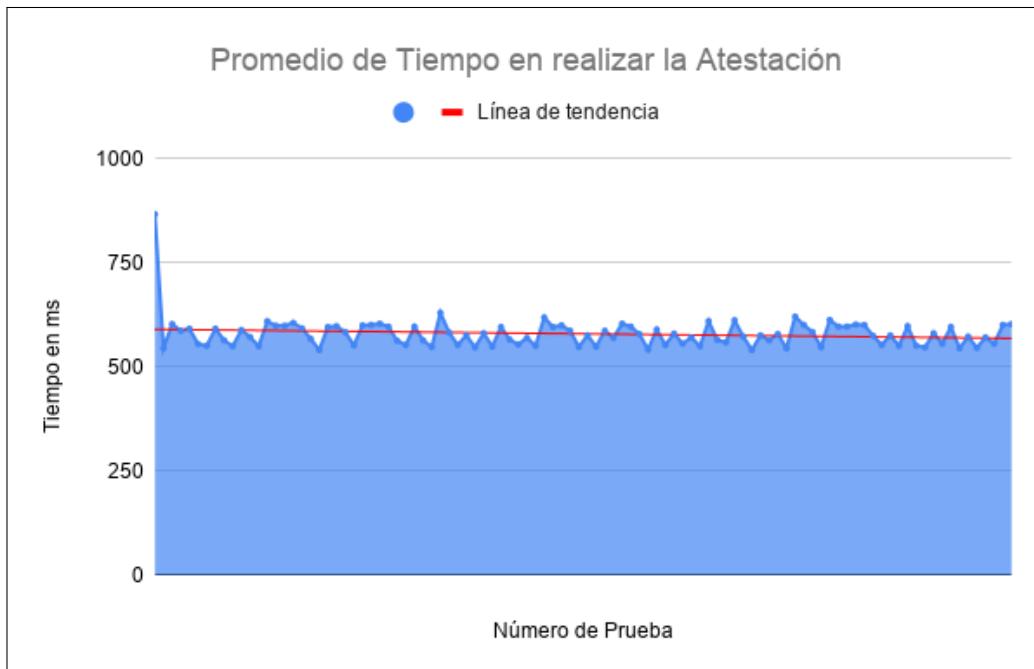


Figura 38: Estudio del tiempo promedio de ejecución

5.1.3 Migraciones no autorizadas por la CA:

Tal y como comentábamos en puntos anteriores, es posible que si una de las plataformas se encuentra comprometida, intente realizar la migración de forma directa con la plataforma destino.

Para solucionar este problema, se ha establecido un registro almacenado en el AMS de cada plataforma, almacenando el token, la plataforma de la cual se espera la migración de un agente y un *timestamp*. Este registro se envía previamente a la plataforma destino.

Posteriormente, se manda la validación a la plataforma origen para realizar el proceso de migración. Se iniciará por ello la función *doMove(Location)*, generando al agente en un jar cifrado, el cual se enviará junto al token.

Al llegar a la plataforma destino, localiza el token y comprueba tanto el origen, como el *timestamp* actual, con el objetivo de determinar si la solicitud es posible de acuerdo al periodo determinado por la CA. Si ambas condiciones se cumplen, el proceso de migrado se realiza con éxito, en caso contrario, se cancela la migración. De nuevo, implementamos el uso de un *challenge* de 32 bits, además de una marca de tiempo que imposibilite los ataques de repetición.

5.1.4 Interfaz Gráfica:

Uno de los aspectos que más problemas ha generado, ha sido la creación de interfaces gráficas para cada agente, todo ello debido principalmente al proceso de migrado. En una primera instancia, se desarrolló las interfaces utilizando javafx y su funcionamiento era correcto, exceptuando la parte final del protocolo, es decir, el proceso de migrado.

Se genera un jar del agente antes de migrar al mismo. Dicho jar, contiene tanto el código, como las clases que necesita para su correcta ejecución. Esta interfaz, requería de múltiples librerías, que en un principio, se establecieron antes de compilar de forma total el desarrollo de este proyecto. Pese a todo

ello, al momento de rastrear las librerías necesarias, no localizaba esta última, impidiendo la ejecución del protocolo. Se optaron por múltiples alternativas, destacando una de ellas, la cual se basaba en incluir directamente el código total de la librería en el proyecto, sin embargo, nada de esto dio resultado.

Una posible solución encontrada tras un proceso de investigación, fue utilizar componentes que procedan de JFrame. Tras descartar el desarrollo generado con javafx, se crearon de nuevo las interfaces a través de un complemento que ofrece Netbeans, con el fin de facilitar dicha tarea. Implementadas las mismas en el directorio del proyecto, dicho problema finalmente se resolvió.

5.2 Caso de Uso:

Tal y como se ha desarrollado a lo largo de este documento, es posible llevar a cabo un protocolo de migración totalmente autónomo a excepción de la primera validación en el sistema en caso de que el valor hash no este previamente establecido. Gracias al uso de claves RSA asimétricas, de 2048 bits, junto con el TPM, es posible garantizar la seguridad e integridad de la información.

Los usos para este protocolo son extensos, abarcando diversos ámbitos. En esta ocasión un posible escenario efectivo corresponde con el entorno judicial, en especial con la cadena de custodia de la información en el entorno digital. Para ello, se ha establecido el siguiente contexto. Supongamos que cierta información debe ser tratada en una serie de puntos. En cada una de estas sedes, se aplica un procedimiento único. Partiendo de unos datos en concretos, estos deben ser procesados para finalmente, poder obtener un resultado.

Este proceso debe realizarse, garantizando principalmente las medidas de seguridad, en concreto los pilares de la confidencialidad, integridad, autenticación y el control de acceso. El protocolo desarrollado cumple con todos estos ámbitos, pues gracias al uso de claves asimétricas, es posible cifrar la información para que solo pueda acceder a la misma la plataforma a la que va dirigida. Por otro lado, gracias a las funcionalidades que ofrece el TPM, es posible garantizar el resto de criterios planteados. El control de acceso, recaerá directamente en la información determinada por el empleado en la plataforma establecida como entidad autenticadora.

Para poner en marcha dicho ejemplo, por tanto, se ha diseñado un agente encargado de realizar funciones determinadas previamente. Gracias al establecimiento de un comportamiento, es posible comunicarse de forma directa con el agente una vez este ha conseguido realizar la migración.

En este ámbito, la entidad denominada como *Secure CA*, puede corresponder a una entidad judicial. Esta, proporciona el acceso a una serie de plataformas, identificándose cada una de estas como direcciones de confianza. Para este caso, partimos de cierta información, disponible por la plataforma A. Estos datos, necesitan ser tratados de forma iterativa tanto en la plataforma B como C. Al finalizar el proceso, en base a la ruta de migrado introducida, los datos ya procesados pueden volver de nuevo a la plataforma de inicio, o establecer como destino la entidad judicial.

Finalizado el protocolo, se obtendría el siguiente esquema de iteraciones entre plataformas:

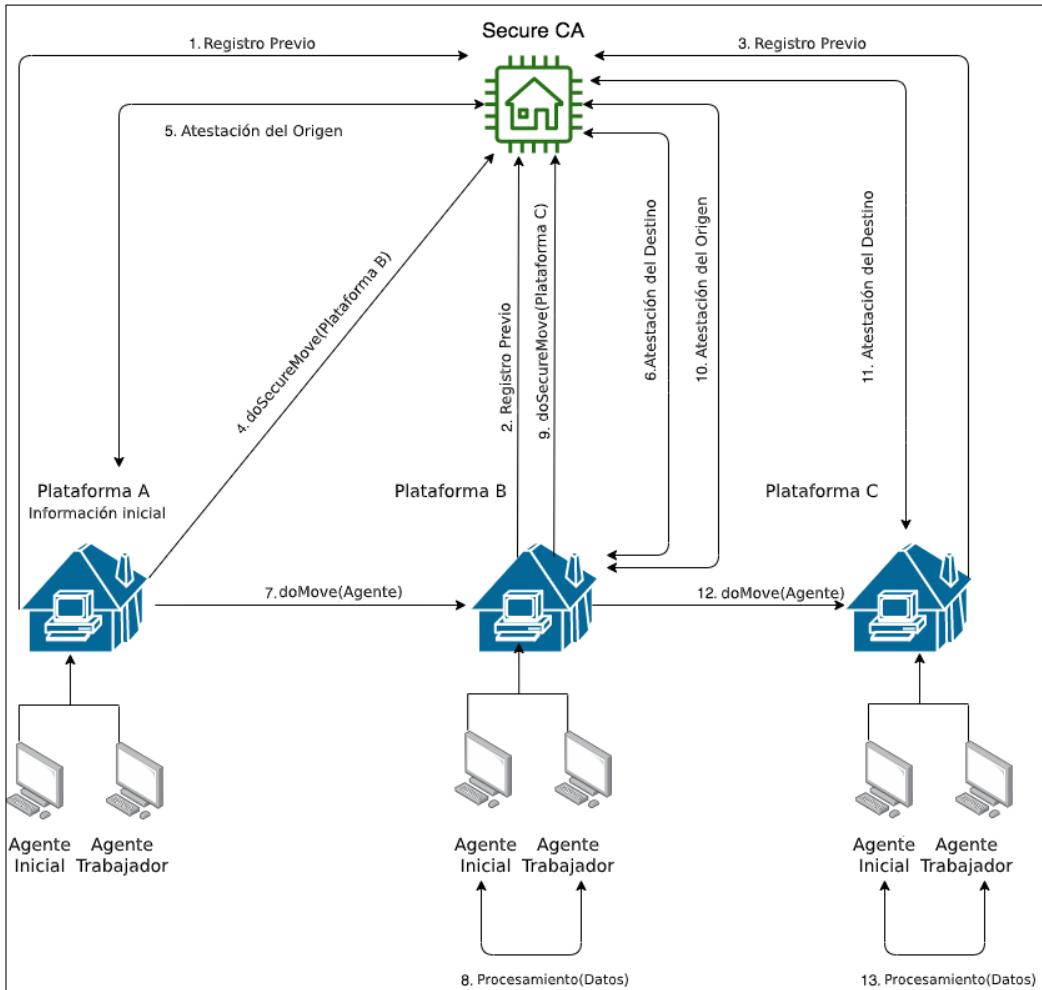


Figura 39: Ejecución del protocolo de migración segura

Gracias a este protocolo, es posible ofrecer garantías en primer lugar de que la información ha sido transmitida de forma segura entre plataformas, sin poner en riesgo la integridad de la misma. Además, aporta garantías de que esos datos no han sido tratados por entidades comprometidas, gracias al proceso de atestación del sistema en el cual el agente desea migrar. Con todo ello, por tanto se ha podido satisfacer el requisito de la seguridad, aportando además una ejecución en entornos distribuidos, en los cuales es posible que

cada plataforma, cuente con unas especificaciones, rendimiento y operaciones de ejecución diversas.

En el caso de que se cambie el estado registrado inicialmente en alguno de ellos, la plataforma CA mostrará el siguiente mensaje, informando posteriormente del pertinente error:



Figura 40: Mensaje mostrado en la CA en caso de que una plataforma se encuentre comprometida.

Para cada agente trabajador, localizado en las diversas plataformas desplegadas, se ha establecido un método de procesamiento simple el cual registra el paso de los datos por la plataforma. En base a las necesidades requeridas por el usuario en un futuro, la modificación del mismo se haría de forma rápida y eficaz.

Para el uso de este protocolo, es necesario un módulo de plataforma confiable que siga la especificación denominada TPM 2.0, de lo contrario, el funcionamiento no será adecuado [1].

Capítulo 6

Conclusión y líneas futuras:

En base a los estudios acontecidos a lo largo de este proyecto, es posible determinar que aún queda un largo camino en cuanto al aprovechamiento del módulo de plataforma confiable. Su uso en la actualidad es reducido con respecto a la gran cantidad de funcionalidades que aporta el mismo, por ende, este ámbito corresponde con uno de los posibles trabajos de investigación en un futuro.

El potencial por explotar, tanto por parte de Jade como del módulo de plataforma confiable es muy extenso, ya que la fusión de las ventajas de cada uno de ellos, propician la creación de un entorno distribuido seguro. Actualmente, se sigue avanzando en desarrollos pertinentes al TPM, generando cada vez nuevas funcionalidades con un mejor rendimiento.

En lo que a trabajo personal se refiere, la evolución de este proyecto es elevada, proporcionando mayores funcionalidades de cara a cada plataforma, con el objetivo de realizar procesos relacionados con el ámbito de la seguridad. En cuanto al TPM, una posible línea futura sería la creación de una librería enfocada a utilizar los diversos métodos que proporciona el mismo en Java, facilitando así la ejecución de ciertos procesos, como ha sido el caso del *quote* en este proyecto.

Como conclusión final, destacar que corresponde con una rama de gran interés, en la cual se ha partido de cero en la mayoría de conocimientos necesarios para el desarrollo del proyecto. En la parte de Jade, uno de los grandes

inconvenientes ha sido el escaso soporte proporcionado para el mismo. No existe mucha información en este aspecto, a la hora de generar nuevos servicios, solo se establece indicaciones para poner en práctica servicios que ya dispone [4]. En cuanto a errores se refiere, ocurre algo similar, pues al ser un entorno complejo, es posible no detectar de primera mano de donde proviene los mismos. Por otro lado, con respecto al TPM, si es cierto que existe gran cantidad de información acerca del mismo, sin embargo, requiere de grandes conocimientos para el desarrollo de nuevas funcionalidades, hecho el cual mezclado con el número de horas definido para el proyecto, puede dificultar su realización dentro del plazo establecido.

Pese a estos impedimentos, considero que se ha obtenido una visión clara de cada una de las partes, asentando las bases de conocimientos para posibles desarrollos en un futuro centrados en este ámbito.

ANEXO 1 - INSTALACIÓN Y EJECUCIÓN DE JADE:

Dentro de este anexo, se van a establecer nociones básicas a la hora de ejecutar Jade. En primer lugar, dicho framework no requiere de grandes especificaciones, en este caso, solo es necesario tener instalado una versión JRE actualizada en el sistema. Para facilitar la ejecución de este software, se entrega un jar, el cual contiene las funciones desarrolladas para el proyecto.

En caso de que de requiera desplegar una serie de plataformas, será necesario lanzar comandos a través de la terminal del sistema, para ello, se debe introducir la siguiente estructura:

```
java -cp {librerias} jade.Boot {Opciones} –services {Servicios}  
–agents {Agentes}
```

Si deseemos desplegar el ejemplo establecido en el punto anterior, será necesario desplegar una plataforma correspondiente con la CA, así como 3 plataformas que posibiliten la migración entre ellas. Atendiendo al rol que represente cada plataforma, será necesario el uso de ciertas librerías que se establecerán a continuación:

- **CA:** Se requiere cargar la librería que contiene los agentes de ejemplo, la librería de migración segura, así como mongo para el manejo de información en el clúster establecido.
- **Plataforma:** Es necesario cargar la librería con los agentes de ejemplo, la librería de migración segura, y *commons* para la serialización de los agentes, momentos anteriores a la migración.

Con respecto a las opciones que ofrece este framework, son numerosas. A continuación se describen las más significativas para poder ejecutar el proyecto realizado.

- **-gui:** Se utiliza para desplegar la interfaz gráfica habilitada por defecto en Jade. En esta ocasión, no será necesaria activar dicho opción puesto que se ha desarrollado una en concreto la cual se inicializa con la ejecución del agente.
- **-host:** Se utiliza para designar la dirección de host que va a tomar el contenedor.
- **-port:** Establece el número de puerto del contenedor.
- **-mtp:** Permite el establecimiento de una dirección MTP para comunicarse con otros agentes, habilitando así un protocolo para el transporte de mensajes.
- **-name:** Determina el nombre que va a tomar la plataforma.
- **-container:** Define el nombre del contenedor.

En la opción de servicios, es necesario cargar las clases desarrolladas con el objetivo de inicializar los mismos. En el caso de la CA solo será necesario lanzar el servicios de migración segura, mientras que para el resto de plataformas, además deberán activar los servicios de movilidad permitiendo así la migración.

Finalmente en el área Agentes, será necesario definir el nombre del agente y la clase desarrollada para el mismo, tomando la siguiente estructura:

Nombre_Del_Agente:Ruta_Clase_Agente

Establecidas las nociones básicas de este framework, en caso de que se desee lanzar el ejemplo establecido en el caso de uso, será necesario ejecutar los siguientes comandos:

CA:

```
java -cp TPM.jar:test.jar:mongo.jar jade.Boot -gui -host  
localhost -port 2511 -services jade.core.SecureCloud.  
SecureCloudTPMService -agents CA:vom.CAPlatform
```

Plataforma A:

```
java -cp commons.jar:TPM.jar:test.jar jade.Boot -container-name  
P1 -gui -host localhost -port 1564 -mtp jade.mtp.http.  
MessageTransportProtocol\(\http://raspberrypi:36811\) -  
services jade.core.mobility.AgentMobilityService\;jade.core.  
migration.InterPlatformMobilityService\;jade.core.SecureAgent  
.SecureAgentTPMService -agents A1:vom.CAAgent\;WORKERA:vom.  
AgentWorker
```

Plataforma B:

```
java -cp commons.jar:TPM.jar:test.jar jade.Boot -container-name  
P2 -gui -host localhost -port 1364 -mtp jade.mtp.http.  
MessageTransportProtocol\(\http://raspberrypi:36711\) -  
services jade.core.mobility.AgentMobilityService\;jade.core.  
migration.InterPlatformMobilityService\;jade.core.SecureAgent  
.SecureAgentTPMService -agents A2:vom.CAAgent\;WORKERB:vom.  
AgentWorker
```

Plataforma C:

```
java -cp commons.jar:TPM.jar:test.jar jade.Boot -container-name P3 -gui -host localhost -port 1864 -mtp jade.mtp.http. MessageTransportProtocol\(http://raspberrypi:36911\) - services jade.core.mobility.AgentMobilityService\;jade.core.migration.InterPlatformMobilityService\;jade.core.SecureAgent .SecureAgentTPMService -agents A3:vom.CAAgent\;WORKERC:vom. AgentWorker
```

Al lanzar los comandos, se abren las vistas establecidas para cada uno de los agentes. Finalmente, solo queda definir los saltos para el agente localizado en la plataforma A. En el área de migración, se introducen los siguientes datos:

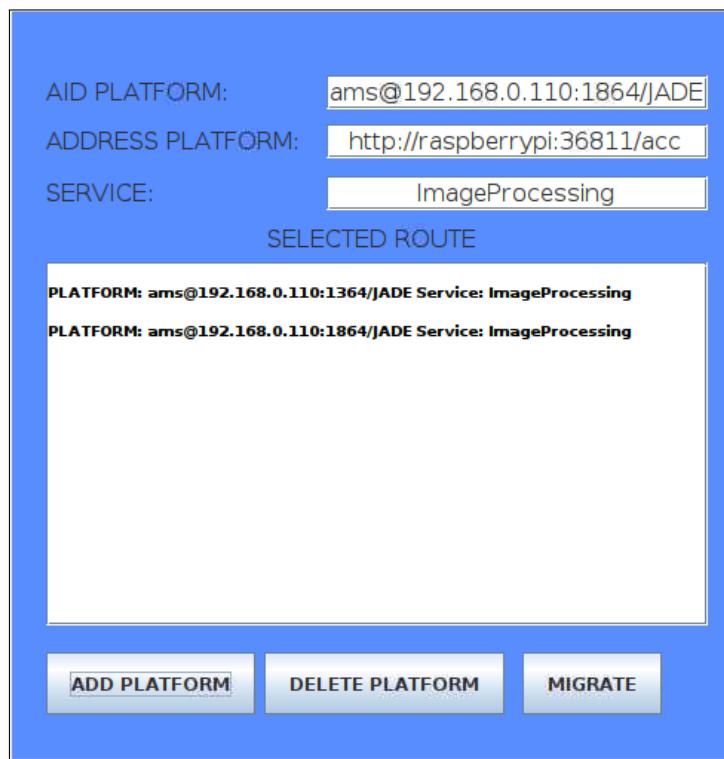


Figura 41: Ruta de migraciones definida para el agente localizado en la plataforma A.

En este caso, como nombre de servicio se introduce *ImageProcessing*, puesto que así ha sido establecido en los agentes denominado como *Workers*. Al finalizar el proceso, en este caso, se concatena en un String designado en el agente el nombre de la plataforma por la cual ha sido procesado. Si el servicio no se indica, realizará una migración a la plataforma destino y no ejecutará ningún proceso adicional.

Como alternativa, también es posible establecer este contexto haciendo uso de la interfaz gráfica proporcionada por defecto, para ello, solo es necesario ejecutar el comando establecido anteriormente con el parámetro -gui.

```
java -cp TPM.jar jade.Boot -gui
```

Dicha opción despliega una plataforma. Una vez ejecutada, muestra la siguiente ventana:

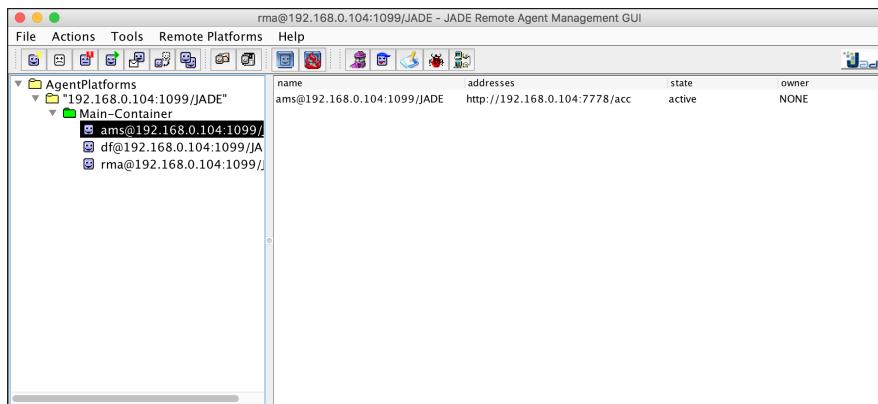


Figura 42: Interfaz por defecto de Jade.

Tal y como se puede apreciar en el AMS de la plataforma, tomará un nombre y dirección disponible. Alguna de las opciones que tenemos disponibles son por ejemplo la ejecución de un nuevo agente, para ello, presionando click

izquierdo sobre el contenedor en el cual se desea ejecutar el despliegue, habilitará la siguiente ventana:

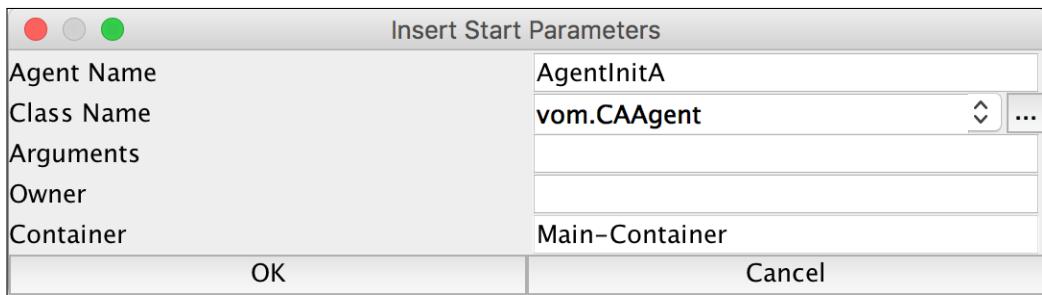


Figura 43: Ventana para desplegar un nuevo Agente.

Cumplimentada esta información en la cual se incluye el nombre, la clase o el contenedor al cual va dirigido, presionamos sobre el botón OK y el agente comenzará su ejecución. Al presionar sobre un Agente, es posible modificar el estado del mismo, permitiendo la suspensión o eliminación del mismo. Además, habilita opciones como por ejemplo el envío de mensajes, clonado o migración. Estos dos últimos, están establecidos para actuar en la misma plataforma.

En caso de requerir mayor detalle de comandos o ejecución de Jade, hacer referencia a la guía de administración [3].

ANEXO 2 - INSTALACIÓN DEL TPM:

Tal y como se establecía anteriormente, para ejecutar las funciones pertinentes establecidas en el protocolo, será necesario instalar un conjunto de librerías que se enuncian a continuación.

- TPM Software Stack 2.0.
- TPM2 Access Broker & Resource Manager.
- TPM2 Tools.
- TPM2 TSS Engine.
- Cryptsetup.

Para realizar la instalación de estos componentes, es necesario hacer referencia a la guía oficial del fabricante ¹⁰. Una vez finalizado este proceso, será posible ejecutar una serie de pruebas. Todas ellas se lanzan a través de la terminal del sistema como comandos. Una posible prueba sería por ejemplo, la generación de números aleatorios.

```
pi@raspberrypi:~ $ cd /home/pi/usageExamples  
pi@raspberrypi:~/usageExamples $ tpm2_getrandom 4  
0xDF 0x18 0x85 0x78
```

Figura 44: Creación de números aleatorios haciendo uso del TPM.

¹⁰<https://www.infineon.com/TPM-TSS-AppNote/>

Bibliografía

- [1] Will Arthur and David Challener. *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015.
- [2] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.
- [3] Fabio Bellifemine et al. “Jade administrator’s guide”. In: *TILab (February 2006)* (2003).
- [4] Fabio Bellifemine et al. “Jade programmer’s guide”. In: *Jade version 3* (2002), pp. 13–39.
- [5] Barry W Boehm. “A spiral model of software development and enhancement”. In: *Computer* 21.5 (1988), pp. 61–72.



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga