eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ **T**rabajo **F**in de **G**rado ▪

Ingeniería de Software

Solución Para La Actualización Segura De Firmware De Dispositivos IoT Sobre Arquitectura Distribuida Basada En Blockchain

---

Jesús Rugarcía

Junio 2023

Dirección

Maider Azanza

Resumen

La cantidad de dispositivos del Internet de las Cosas es cada vez mayor. Su limitada capacidad, sin embargo, los convierte en objetivos de ataques, ya que no pueden implementar protocolos de seguridad avanzados. Por este motivo, es necesario actualizar los dispositivos cada vez que se detecta una vulnerabilidad. El proceso de actualización, no obstante, también es vulnerable y es utilizado en muchos ataques.

El objetivo de este TFG es presentar una solución para llevar a cabo la actualización de dispositivos del Internet de las Cosas de manera segura, utilizando para ello un sistema construido alrededor de una arquitectura distribuida basado en una red blockchain.

Se presentan el diseño y la implementación realizados, y se da una valoración sobre el resultado tras realizar pruebas. Adicionalmente se incluyen los desafíos enfrentados en el proyecto y se muestran la planificación y el seguimiento y control del proyecto.

Índice

Índice de Figuras

1 Introducción

La industria está evolucionando rápidamente en los últimos años. Nuevas tecnologías como la Inteligencia Artificial o el Big Data están revolucionando diversos campos. El enfoque hacia el dato y la automatización han permitido que florezca el internet de las cosas o Internet of Things (IoT en adelante). El IoT está compuesto por una gran diversidad de aparatos conectados a redes como internet, como pueden ser sensores, vehículos, cámaras de seguridad, routers o impresoras.

La expansión acelerada de esta tecnología, sin embargo, conlleva riesgos de seguridad muy alarmantes. La limitada capacidad de los dispositivos IoT dificulta la implementación de medidas de seguridad eficaces, lo que los convierte en objetivos muy interesantes para agentes maliciosos. Por ejemplo, en algunos casos estos dispositivos pueden ser utilizados para realizar ataques masivos, como en el ejemplo de las redes Mirai o Meris, que se utilizaron para denegar servicios utilizados por millones de personas, o pueden ser utilizados para invadir la privacidad de individuos capturando las imágenes de cámaras de seguridad o cámaras para bebés.

Cuando un fallo de seguridad es descubierto, es necesario actualizar el software controlador (firmware) de los dispositivos afectados. El proceso de actualización es, sin embargo, uno de los puntos más vulnerables, y es comúnmente utilizado para realizar ataques. Por ejemplo, se puede utilizar para obtener la imágen de un dispositivo, un archivo binario que contiene el firmware compilado del dispositivo, lo que permite encontrar sus vulnerabilidades mediante un proceso de ingeniería inversa. También puede ser aprovechado para instalar imágenes de la creación de un atacante, lo que le permite controlar el dispositivo e introducir nuevas vulnerabilidades.

El objetivo de este Trabajo de Fin de Grado (TFG en adelante) es diseñar una solución de software que avance hacia la meta de conseguir la actualización segura de dispositivos IoT. Adicionalmente, se plantea la implementación de algunos de los elementos diseñados.

La solución se basará en una arquitectura distribuida construida alrededor de una red blockchain permisionada. Con este enfoque se garantiza la persistencia de los datos utilizados y se eliminan algunos problemas como la imposibilidad de actualizar un dispositivo después de la desaparición del fabricante, además de proteger las actualizaciones mediante el uso de técnicas criptográficas. Por otro lado, el uso de una red permisionada añade una capa de confianza inexistente en redes públicas, ya que todos los participantes de la red cuentan con un mínimo nivel de confianza hacia el resto de participantes y se excluyen participantes externos.

El trabajo se realizará en conjunto con Ceit, un centro de investigación tecnológica con 40 años de historia, dentro del grupo de análisis de datos y gestión de la información o Data Analysis and Information Management (DAIM).

2 Antecedentes

En este capítulo se describen los conceptos relacionados con el proyecto, así como un breve estudio de los proyectos realizados con anterioridad y que representan el estado del arte. Finalmente, se describe la entidad colaboradora.

2.1 Conceptos relevantes

En este apartado se hace una breve descripción de los conceptos más relevantes utilizados en el proyecto: el Internet of Things, la blockchain y la criptografía.

2.1.1 Internet of Things (IoT)

El internet de las cosas es el conjunto de objetos físicos con capacidad de procesamiento, a menudo poseedores de sensores o actuadores, interconectados para el intercambio y procesamiento de información a través de una red como internet. Abarca muchos campos como la domótica, la medicina o la industria, y se ha visto expandida recientemente gracias la evolución y convergencia de muchas tecnologías como el Machine Learning y el Big Data.

Los dispositivos conectados al internet de las cosas se clasifican en tres categorías según el RFC 7228 [5]:

- Clase 0: Memoria RAM inferior a 10 KiB, Memoria Flash inferior a 100 KiB. Estos dispositivos tienen restricciones severas a la hora de comunicarse de manera segura con internet, por lo que suelen conectarse con otros dispositivos intermediarios como gateways o servidores.
- Clase 1: Memoria RAM de alrededor de 10 KiB, Memoria Flash de alrededor de 100 KiB. Estos dispositivos son capaces de ejecutar protocolos de seguridad ligeros, pero no tienen la capacidad suficiente para emplear algunos más avanzados como HTTP sin un gateway.
- Clase 2: Memoria RAM de alrededor de 50KiB, Memoria Flash de alrededor de 250 KiB. Estos dispositivos tienen una capacidad suficiente para implementar protocolos más avanzados, pero siguen teniendo ciertas restricciones.

Los dispositivos mencionados suelen agruparse como dispositivos restringidos, y a menudo son acompañados por gateways, dispositivos con mayor capacidad que se encargan de conectar y gestionar la red.

El IoT juega un papel fundamental en el desarrollo de la industria. El uso de dispositivos conectados a la red permite la recolección de datos en tiempo real y facilita la automatización de distintos procesos. Algunos de los sectores más beneficiados son la producción industrial y la sanidad, pero cualquier sector en el que sea relevante la monitorización de datos provenientes del mundo físico puede beneficiarse en gran medida de esta tecnología.

2.1.1.1 Firmware

Los dispositivos IoT necesitan de una serie de comandos o instrucciones para ejecutar sus tareas. El código o software que controla su funcionamiento a bajo nivel, es decir, a nivel de máquina, se conoce en castellano como soporte lógico inalterable o firmware en inglés. Este firmware se escribe en lenguajes de programación que generalmente están orientados a un programador humano. Por ello, deben ser compilados para traducirlos a un lenguaje utilizable por el dispositivo. Al resultado de esta compilación se le llama imagen, y se trata de un archivo binario utilizable por el dispositivo. Cabe mencionar que una imagen de firmware se compila para ser utilizada por un dispositivo con un tipo de procesador concreto, como pueden ser Arm o x86.

2.1.2 Blockchain

Se trata de una tecnología de tipo ledger descentralizado (Distributed Ledger Technology, DLT). Un ledger se asemeja a una base de datos, pero a diferencia de esta, almacena información sobre los cambios realizados además del estado actual de la información.

Se dice que es descentralizado porque se replica a lo largo de una serie de participantes de red o nodos, donde cada uno de estos nodos es responsable del mantenimiento de la red. Cada uno de los nodos tiene una copia del ledger idéntica a las demás, que se actualiza de manera independiente.

Todos los nodos que forman parte de una red DLT contienen un bloque básico, una agrupación de transacciones o información de un tamaño preestablecido. Cuando se dan transacciones en la red, todos los nodos deben establecer un consenso basado en una serie de reglas previamente acordadas para declararlas válidas. Dichas transacciones se agrupan en nuevos bloques, que son almacenados en el ledger formando una cadena mediante referencias hash ordenadas. Es destacable mencionar que el proceso ocurre sin intermediarios.

Los datos de esta red pasan por técnicas criptográficas para asegurarse que una vez en la red, no pueden ser modificados, lo que se conoce como la propiedad de inmutabilidad. Adicionalmente, aunque los contenidos de un asset, nombre que toman los objetos guardados en el ledger, cambien como resultado de una transacción, los valores anteriores pueden ser recuperados examinando la cadena de bloques, en la que quedan registrados todas las modificaciones al ledger desde su creación.

La seguridad de la red se mantiene con los sistemas de prueba o consenso, de los que hay muchos tipos. Dichos sistemas ponen en común las decisiones tomadas por distintos nodos sobre la validez de una determinada transacción, evitando que un participante malicioso pueda modificar los contenidos de la red de manera unilateral.

Para realizar tareas dentro de la red, como transacciones o queries, se utilizan contratos inteligentes, que otorgan acceso limitado a la red. Permiten encapsular la información y automatizar ciertas tareas. Son estos contratos inteligentes los que permiten añadir una lógica de negocio a la red, creando la posibilidad de implementar aplicaciones basadas en estas redes. De manera simplificada, se podría decir que un contrato inteligente es un programa ejecutado dentro de la red. Debido al volumen de operaciones que suelen adquirir las redes blockchain, es recomendable que sean lo más

ligeros posible. Por otro lado, deben tener resultados deterministas para que los nodos puedan tener consenso sobre las transacciones que surgen de su ejecución.

Cuando los procesos de la red o el almacenamiento son internos a la cadena de bloques, decimos que se trata de un proceso on-chain. Cuando ocurren procesos fuera de la red o se utiliza un almacenamiento externo pero queda reflejado en la red, decimos que se trata de un proceso off-chain. Los procesos on-chain tienen la ventaja de la persistencia, pero pueden tener costes de procesamiento o velocidad significativos.

Aunque el público general relaciona el concepto de blockchain con los proyectos de criptodivisas como Bitcoin o Ethereum, el potencial de la tecnología va mucho más allá y tiene grandes aplicabilidades en contextos comerciales, empresariales y públicos, en los campos de seguridad y trazabilidad de productos, entre otros.

Algunas de las aplicaciones de las tecnologías de ledger distribuido son [6]:

- Mediledger Network: red blockchain utilizada en el sector farmacéutico para compartir información relacionada a la seguridad de los medicamentos, sin compartir información sensible. Su uso permite rastrear un producto a través de la línea de suministros para saber si es auténtico.
- Alibaba Food Traceability: Permite rastrear un producto en la cadena de suministros para asegurar que se trata de un producto genuino. Utilizado por Alibaba en Australia para el transporte de alimentos a China.
- Dun & Bradstreet Business Identity: permite comprobar la identidad de un agente contrastando múltiples fuentes de información para evitar posibles riesgos y fraudes en transacciones financieras.
- Wien Energy: blockchain utilizada para gestionar las transacciones entre proveedores de energía en el contexto del crecimiento de las energías renovables.

2.1.2.1 Componentes de una red blockchain

- Nodos: también conocidos como pares (peers). Forman y propagan información a través de la red, y validan transacciones dentro de la misma. Cada uno de ellos ejecuta un cliente que suele estar almacenado en un contenedor como Docker.
- Ledger: almacenamiento de información de la red. Se asemeja a una base de datos compuesta por una lista enlazada de bloques. Cada bloque contiene un número de transacciones y un hash del bloque que lo precede. Mediante dicho hash se puede acceder a todas las transacciones realizadas con anterioridad en una manera ordenada, dando lugar a la analogía de la cadena, donde todos los bloques están fuertemente ligados entre sí.
  Cada uno de los nodos gestiona una copia idéntica del ledger. Se compone de dos elementos principales:
    - El world state es una base de datos que contiene los valores más recientes de los elementos almacenados.
    - El registro contiene todos los movimientos que se han hecho históricamente en la red, lo que permite reconstruir el estado actual y conocer todas las modificaciones realizadas sobre los mismos, pudiendo obtener todos los estados de un elemento hasta su estado inicial.

- Protocolo de Consenso: es un algoritmo utilizado para validar y ordenar los bloques que contienen las transacciones. Es esencial para mantener la seguridad de la red. Todas las transacciones deben pasar a través de él, y solo aquellas que lo cumplan pueden ser almacenadas.
- Contratos Inteligentes: contienen la lógica de negocio que controla las transacciones de la red. Se almacenan en la red al igual que las transacciones. Están compuestos por código e información. Su ejecución debe tener resultados deterministas, ya que son ejecutados por todos los nodos de la red y todos ellos deben tener el mismo resultado para que se produzca consenso.

2.1.2.2 Tipos de redes

Las redes pueden dividirse en distintos tipos dependiendo de la privacidad (redes públicas y privadas) y los permisos necesarios en la misma (redes permisionadas y no permisionadas).

Las redes pueden ser públicas, como es el caso de bitcoin, donde todos los nodos son anónimos. En estos casos, los contratos y los datos deben ser públicos, pero los usuarios suelen permanecer anónimos. No existe confianza entre nodos, pero todo el mundo puede participar en la red siguiendo una serie de normas previamente establecidas. Para garantizar su correcto funcionamiento, se introduce el concepto de divisa nativa, utilizada para recompensar a los usuarios que las mantienen y que participan en los algoritmos de consenso como la prueba de trabajo o la prueba de participación (stake). Dichos algoritmos suelen ser costosos computacionalmente, pero son necesarios cuando se trabaja con usuarios anónimos sobre los que no se puede depositar confianza, y tienen el problema de hacer las redes difíciles de escalar, ya que tienden a ralentizarse cuando aumentan en tamaño.

En las redes privadas, todos los usuarios suelen ser conocidos para el resto de usuarios. A menudo estas redes necesitan permisos para operar en ellas, y no cualquiera puede acceder. Al ser usuarios conocidos,y hasta cierto punto confiables, los contratos y los datos pueden gozar de más privacidad. Es preferible en entornos empresariales y comerciales. Debido a que todos los participantes gozan de cierta confianza e interés común en mantener el correcto funcionamiento de la red, no necesitan de la utilización de divisas nativas, ya que no es necesario establecer protocolos de consenso tan costosos y no se necesita crear un incentivo adicional para su mantenimiento. Esto tiene el añadido de que permite crear redes más escalables y veloces.

Las redes no permisionadas permiten que todos los nodos participantes de la red puedan ejecutar todos los procesos de la red, y que tengan la posibilidad de solicitar transacciones y ejecutar contratos inteligentes. Este tipo de red es el más común dentro de aquellas que son públicas, ya que todos los nodos son mantenidos por clientes anónimos y existen algoritmos de consenso costosos que garantizan que ningún cliente pueda realizar acciones maliciosas.

Las redes permisionadas, en cambio, reservan la capacidad de realizar determinadas acciones a un grupo de nodos con los permisos correspondientes. Por ejemplo, la ejecución de contratos inteligentes o la solicitud de creación de transacciones podría estar reservada a ciertos nodos, mientras que el resto solo quedaría limitado al mantenimiento de los ledgers. Resultan

especialmente interesantes en el caso de las redes privadas, ya que al conocerse la identidad de los nodos pueden asignársele permisos específicos. Sin embargo, también pueden darse redes públicas y permisionadas.

Cuando una red es gestionada de manera privada y permisionada por un grupo de entidades con un objetivo común, manteniendo cierta independencia entre las entidades, podemos hablar de una red de tipo consorcio. Las redes consorcio son útiles en el contexto de sectores económicos o industriales específicos, ya que permiten a entidades con un objetivo común beneficiarse de las propiedades de las redes blockchain aunque sean competidoras entre ellas, a la par que excluyen el acceso público a la misma.

2.1.3 Criptografía

La criptografía es un conjunto de técnicas que buscan proteger la información para evitar que sea consumida por terceras entidades. Normalmente estas técnicas utilizan propiedades matemáticas para transformar la información haciéndola inutil, hasta que es recuperada o desencriptada.

Para este proyecto resulta especialmente relevante la criptografía asimétrica. Es un tipo de criptografía donde cada participante tiene un par de claves. Una de las claves es privada, mientras que la otra es pública. Un participante puede utilizar la clave pública de otro participante para cifrar información de manera que solo pueda descifrarla aquel que tenga la clave privada asociada, o un participante puede firmar un mensaje utilizando su clave privada para garantizar a aquellos que tienen su clave pública que es el autor del mensaje .

Este tipo de criptografía tiene muchas aplicaciones. Por ejemplo, se utiliza en mensajería instantánea para proteger el contenido de los mensajes. El emisor cifra el mensaje utilizando la clave pública del receptor. El mensaje solo puede ser descifrado utilizando la clave privada del receptor, por lo que si el mensaje es interceptado, no puede ser leído.
La seguridad de una red blockchain se basa en este tipo de criptografía, que es utilizada para la verificación de la identidad de sus participantes.

Este tipo de cifrado tiene el problema de que muchas veces no se tiene la certeza de que una clave pública pertenece realmente al destinatario deseado. Para solucionarlo, se introdujo la idea de firmar los documentos mediante una firma o certificado digital, lo que acabó dando lugar a la infraestructura de clave pública, o Public Key Infrastructure (PKI) [16] El refinamiento de esta infraestructura ha ido dando lugar a varios estándares, siendo X.509 [17] uno de los más importantes.

La PKI se centra en las operaciones de certificación y validación. El proceso de certificación liga una clave pública con una identidad de manera auténtica. El proceso de validación se ocupa de verificar la validez de un certificado.

Una PKI tiene los siguientes elementos:

- Certificate Authority (CA): Autoridad encargada de generar los certificados. Un certificado contiene la clave pública y la información del propietario de la clave privada. Un

certificado puede ser revocado por diversos motivos, como la pérdida de la clave privada, y debe ser validada periódicamente.
- Registration Authority (RA): encarga de autenticar a los solicitantes de certificados y de enviar dichas solicitudes al CA.
- Repositorio de certificados: Repositorio donde se almacenan todos los certificados generados por la CA. Puede ser accedido directamente a través de un sistema de distribución o a través de una Validation Authority (VA) que haga de intermediario para esconderlo del cliente.

2.1.3.1 Algoritmos y tecnicas relevantes

Existen varios algoritmos y técnicas criptográficas, cada uno con sus aplicaciones específicas. Para la compresión de la implementación realizada en este proyecto, los siguientes resultan especialmente relevantes:

Rivest-Shamir-Adleman (RSA)
RSA [30] es uno de los algoritmos de cifrado más populares de la historia. Se presenta por primera vez en 1977, en el contexto de la aparición del correo electrónico. Se trata del algoritmo principal en la criptografía asimétrica, utilizado por muchas entidades en todo tipo de contextos.

El algoritmo se basa en una función trapdoor (es decir, que es difícil de calcular pero fácil de comprobar una vez se tienen los resultados) que utiliza números primos aleatorios para generar un par de claves. Estas claves son utilizadas en una función que utiliza módulos y potencias, lo que hace que a mayor tamaño tengan las claves, más difícil de calcularlas sea.

Una de estas claves se hace pública, mientras que la otra se mantiene privada. La información a compartir pasa por una función que modifica su valor, pero este puede ser restaurado utilizando la otra clave.

Adicionalmente, al mensaje cifrado se le añade un padding. Este padding es una serie de información aleatoria que se añade al mensaje para hacer que identificar su formato interno resulte más complicado, de manera que la clave no pueda ser adivinada infiriendo patrones.

En concreto se utiliza RSA-2048 en la implementación de este proyecto, que es la longitud de clave recomendada a fecha de la realización de este proyecto.

Secure Hash Algorithm (SHA)
Los algoritmos SHA son algoritmos de hashing (también conocido como digesting). Convierten un input en un hash (o digest, ambos términos usados como sinónimos en este documento), una cadena de caracteres que en apariencia no tiene significado útil, y que tiene una longitud predefinida. Estos hashes se obtienen mediante una serie de operaciones matemáticas y son comúnmente únicos para cada input. Se trata de funciones de una sola dirección, es decir, que una vez se ha realizado un hash, este no puede ser convertido de vuelta en el input original.

Son útiles para realizar comprobaciones sin tener que almacenar información sensible. Por ejemplo, se utilizan para realizar comprobaciones de contraseñas en aplicaciones web. En lugar de

almacenar la contraseña de un usuario, se almacena el hash de la misma, de manera que alguien que obtuviese acceso al servidor no pudiese obtener la contraseña en texto plano. Cuando el usuario se loguea, se pasa la función de hashing sobre el input de la contraseña, y este se compara con el hash almacenado.

Se ha decidido utilizar el algoritmo SHA-384 en la implementación del proyecto, debido a que es potencialmente más seguro que SHA-256 y más rápido de calcular en algunos casos. El estándar mínimo actual es SHA-256, pero se ha querido añadir seguridad adicional ya que es el algoritmo que se usa para verificar que las actualizaciones no han sido modificadas, la parte central de este proyecto.

JSON Web Token (JWT)
JWT es descrito en el RFC 7519 [31]. Se trata de un estándar de intercambio de información segura. Utiliza JSON como formato, y es muy utilizado en aplicaciones web y APIs.

Se conforma por un encabezado, un payload y una firma. Un JWT se puede formar de distintas maneras, por lo que el encabezado contiene información sobre el propio token. La firma garantiza la veracidad del token, y es creada por el emisor del mismo.

La principal ventaja que aportan es que son stateless, de manera que el propio token es suficiente para realizar la comprobación de identidad sin la necesidad de crear una sesión de usuario.

El propósito de su uso es el de verificar la identidad de los participantes, ya que la información transmitida no está cifrada.

2.2 Estudio de estado del arte

En este apartado se analiza el problema a resolver y se describen brevemente las soluciones propuestas previamente.

2.2.1 Descripción del problema

Con el reciente auge de la tecnología del IoT, han aparecido preocupaciones con respecto a la seguridad y la privacidad. Los dispositivos utilizados generalmente cuentan con capacidades muy limitadas, lo que dificulta la implementación de medidas de seguridad eficaces.

Uno de los ataques más famosos en los últimos años es el caso de la red Mirai [2]. Mirai escanea internet en busca de direcciones IP de dispositivos IoT que pueda considerar vulnerables. Después utiliza una combinación aleatoria de nombres de usuario y contraseñas comunes para el tipo de dispositivo que está atacando. Cuando consigue acceso a un dispositivo, introduce su dirección y credenciales en un servidor. Utilizando dicha información, se introducen binarios maliciosos en los dispositivos que se camuflan dentro del dispositivo aguardando instrucciones. Estos dispositivos después se utilizan de manera masiva para generar tráfico, convirtiéndolos en parte de un ataque DDoS a gran escala. Dichos ataques han sido utilizados para denegar servicios muy importantes, como los ofrecidos por Dyn, un proveedor de servicios de red americano.

A causa de estas vulnerabilidades, la seguridad en el entorno de los dispositivos IoT se ha convertido en una de las preocupaciones del sector a nivel global, desde tan temprano como 2015 [3]. La heterogeneidad, desconocimiento a nivel de seguridad de los usuarios y falta de un entorno de confianza representan serios problemas, lo que sumado a la capacidad limitadas de los dispositivos se convierte en un gran desafío a solucionar.

Cuando aparece una vulnerabilidad en el firmware que operan los dispositivos, es necesario actualizarlos con una imagen que la solucione. Sin embargo, este proceso es uno de los puntos más problemáticos al asegurar los dispositivos [4].

El proceso es susceptible a las siguientes categorías de ataque, entre otras:

- Ingeniería inversa: Los atacantes obtienen el firmware y lo analizan para encontrar vulnerabilidades e información sensible.
- Modificación de firmware: Se modifica el firmware para añadir vulnerabilidades o permitir ataques y operaciones no autorizadas.
- Obtención de acceso: Un atacante puede obtener acceso a un dispositivo externo que se comunique con el dispositivo para realizar diversos ataques.
- Instalación de imagenes maliciosas: Un atacante puede aprovecharse del proceso para instalar una imagen maliciosa que le otorga el control del dispositivo o le permite realizar ataques adicionales.
- Dispositivo no autorizado: Un dispositivo no autorizado puede hacerse pasar por uno que lo es para descargar una copia del firmware, lo que permite realizar un proceso de ingeniería inversa y puede traer consigo nuevos ataques hacia el fabricante.

El ataque de modificación de firmware, por ejemplo, ha sido utilizado para atacar con éxito infraestructuras de comunicación, sistemas SCADA y PLC, cajeros ATM (causando pérdidas económicas significativas) y se ha utilizado para crear la botnet PsyB0t que afectó a miles de routers, y que fue descubierta en el año 2009.

Por otro lado, existen casos en los que el fabricante de un dispositivo deja de realizar o mantener disponibles actualizaciones para dispositivos menos recientes. También existen casos en los que un fabricante desaparece del mercado. En ambos casos, aquellos usuarios de los dispositivos del fabricante dejan de poder acceder a nuevas actualizaciones para sus dispositivos, lo que pone permanentemente en riesgo la seguridad de los dispositivos adquiridos y que no siempre pueden ser sustituidos.

Es por estos motivos que la creación de un sistema seguro de actualización de firmware para dispositivos IoT resulta imprescindible.

2.2.2 Soluciones previas

En este apartado se exponen algunas de las soluciones propuestas previamente a este TFG.

2.2.2.1 RFC 9019

Para solucionar el problema, se han propuesto varias soluciones. La más relevante viene en el RFC 9019 [1] elaborado por el grupo SUIT del Internet Engineering Task Force (IETF). En este documento se presenta una arquitectura basada en un servidor centralizado que establece una serie de pautas a seguir.

En ella, el autor de los dispositivos o el autor de la imagen crea un manifiesto con los metadatos relevantes para la actualización. Es destacable la inclusión de un hash elaborado con la imagen en cuestión, que permite detectar si la imagen ha sido alterada de alguna manera.

La imagen y su manifiesto después son compartidas con un operador de dispositivos, que la hace disponible en un servidor. Dicho servidor contiene un componente de monitorización que puede comunicarse con los dispositivos a actualizar, por lo que el servidor puede conocer el estado de los dispositivos y enviarles las actualizaciones cuando es necesario. De la misma manera, los dispositivos pueden verificar el estado del servidor y solicitar la actualización cuando es conveniente.

Una vez obtenido el manifiesto, el dispositivo o el gateway al que está conectado puede verificar el manifiesto para decidir si quiere descargar la imagen. Con la imagen descargada, puede realizar una comprobación utilizando el hash presente en el manifiesto. Si la imagen es verificada correctamente, puede entonces proceder a instalarla.

La solución presenta algunos puntos mejorables al utilizar un servidor centralizado. Al haber un único punto de fallo, el servidor se convierte en un punto de ataque valioso, ya que todos los ataques en contra de la red se dirigirán contra él. En caso de que dicho servidor caiga o deje de estar disponible, o se modifiquen los datos de su base de datos, la solución puede verse comprometida. Por otro lado, la desaparición del autor del mercado puede implicar que los dispositivos que aún no habían obtenido la actualización pierdan para siempre la oportunidad de obtenerla, y en caso de que haya nuevas vulnerabilidades los dispositivos no contarán con nuevo firmware que lo solucione.

2.2.2.2 Soluciones descentralizadas

Para intentar solucionar los defectos de esta solución, en los últimos años se han realizado varias propuestas que incluyen la utilización de la tecnología de ledger descentralizado o blockchain. Con una arquitectura descentralizada, se dificulta la posibilidad de denegar los servicios. Además, ofrecen un nivel de persistencia que dificulta la tarea de alterar las imágenes. Es altamente improbable que un autor fraudulento pueda introducir manifiestos maliciosos gracias a los protocolos de consenso, que necesitan que todos los nodos de la red aprueben la transacción.

B. Lee Et Al. [7] son los primeros en proponer el uso de una arquitectura descentralizada. Proponen el uso de una red blockchain cuyos nodos están presentes en los dispositivos IoT. Un nodo de verificación gestionado por el fabricante de los dispositivos es siempre conocedor de la última versión de firmware. Un nodo de petición puede enviar su versión al nodo de verificación, de manera que este compare ambas versiones. En caso de que el nodo de petición contenga una versión más antigua, el nodo de verificación solicita a nodos adyacentes su versión hasta encontrar

uno que tenga la versión más reciente. Entonces, el nodo de petición descarga la actualización desde el nodo que contiene la versión más reciente.

La solución presenta el problema de que los algoritmos de consenso exigen demasiada carga de trabajo a los dispositivos, que cuentan con recursos limitados. Por otro lado, solo soporta actualizaciones de un único autor.

A. Boudguiga Et Al. [8] proponen un modelo en el que el autor de las actualizaciones selecciona los dispositivos a actualizar y sube la imagen a un portal web. Los dispositivos a actualizar entonces reciben una notificación. Las imágenes se suben a la red durante las transacciones, lo que las hace persistentes o se comparten off-chain con un nodo de inocuidad, nodos gestionados por entidades de máxima confianza como agentes nacionales de seguridad o información (se menciona la CIA, por ejemplo) para que estos los verifiquen y hagan disponibles.

Tiene la ventaja de que los dispositivos ya no tienen tanta carga de trabajo, ya que ahora los nodos estarán gestionados externamente por gateway. Por otro lado, tiene el problema de que la inclusión de las imágenes en los bloques de transacción puede incrementar su longitud de manera muy significativa, haciendo que el procesamiento de transacciones se ralentice a lo largo del tiempo.

A.Yohan Et Al. [9] proponen un modelo en el que existen un repositorio de fabricante y un repositorio broker que almacena las imágenes y su información asociada. La red está compuesta por nodos completos y nodos ligeros. Cuando un autor genera una actualización la sube al nodo de fabricante, este utiliza un contrato inteligente para subirlo a la red y hacer que los demás nodos lo verifiquen. Los nodos ligeros, que están conectados a gateways, entonces envían la actualización mediante una URI para que los gateways las descarguen y ejecuten la actualización. Alternativamente, un dispositivo puede acceder a todas las actualizaciones en la red a través del repositorio broker.

Este modelo tiene el problema de generar tráfico excesivo cada vez que se llama un contrato inteligente desde un repositorio que es actualizado.

S.Choi Et Al. [10] proponen un modelo en el que se definen nodos de registro y nodos de recuperación distintos al resto de nodos de la red. Los autores registran las actualizaciones en el nodo de registro, que almacena el manifiesto en la red blockchain y la imagen en un sistema de almacenamiento distribuido IPFS. Los dispositivos comparan su versión con la última disponible en el nodo de recuperación. En caso de ser inferior, el nodo de recuperación le envía el manifiesto y la imagen recuperada del almacenamiento distribuido.

M.Son et Al. [11] Proponen un modelo en el que se diferencia dos tipos de subred. Por un lado, los nodos de autor se conectan con otros nodos de tipo broker, formando la red externa. A su vez los nodos broker se conectan con otros nodos en una red interna, donde están los dispositivos a actualizar o sus gateways. Esta solución también propone el uso de IPFS como sistema de almacenamiento off-chain.

Otras propuestas interesantes son las de S.Dhakal Et Al. [12], que propone la inclusión de un mecanismo de actualización delta que solo descarga las modificaciones en lugar de los binarios

completos si no es necesario; la de A. Yohan Et Al. [13] que propone la utilización de Skipchain, un tipo de blockchain que incluye enlaces de larga distancia en sus bloques para facilitar la navegación de la información almacenada, y la de M. Tsai Et Al. [14] que propone el uso del protocolo MQTT.

2.3 Descripción del centro colaborador

Ceit es un centro de investigación tecnológica sin ánimo de lucro fundado por la Universidad de Navarra en el año 1982. Su tarea es la elaboración de proyectos industriales en el ámbito del I+D en estrecha colaboración con distintas empresas. El centro se enfoca en la mejora de la competitividad del tejido empresarial mediante proyectos de investigación aplicada y en la formación de perfiles de jóvenes investigadores. En el año 2020 contaba con más de 200 empleados y con 15 spin-offs constituidos tras 40 años de experiencia.

Ceit cuenta con varios departamentos y grupos, enfocados en distintos sectores de la industria. Abarcan las áreas de materiales y fabricación, transporte y energía, agua y residuos y tecnologías de la información y comunicación.

Es precisamente en este último sector donde se desarrolla este TFG, dentro del grupo de Data Analysis and Information Management (DAIM). El grupo DAIM trabaja en las áreas de la ciberseguridad y el Big Data y colabora en proyectos de relevancia a nivel nacional como HERMES o CYBERPREST.

El equipo y el tutor del proyecto, Santiago Figueroa, cuentan con experiencia en el campo de las tecnologías descentralizadas e IIoT, Industrial Internet of Things. Cabe destacar la publicación de varios artículos relacionados con las tecnologías a tratar en este TFG [32][33][34].

3 Planificación del proyecto

En este capítulo se definirán el producto a realizar, su alcance y las tareas necesarias para su elaboración. El objetivo es crear un marco de acción que permita alcanzar las necesidades del proyecto y reducir al máximo los riesgos e incidencias, así como el impacto de los mismos. También se definirán las estrategias de almacenamiento de la información y los canales de comunicación a utilizar durante la duración del proyecto.

3.1 Descripción de la solución

El objetivo principal de este TFG es la elaboración del diseño de una solución de software para la actualización segura de firmware de dispositivos IoT. Adicionalmente, se implementarán algunos de los elementos diseñados, pero la implementación completa de la solución queda fuera del alcance del proyecto debido a las restricciones de tiempo asociadas al mismo.

La solución se cimentará sobre un sistema descentralizado para el almacenamiento de actualizaciones de firmware para dispositivos IoT. Se utilizará una red IPFS para almacenar las imágenes y se creará una aplicación sobre una red blockchain para el registro y recuperación de las actualizaciones, donde también se almacenarán los manifiestos en el ledger de la red.

La tecnología sobre la que se diseñará e implementará la solución es Hyperledger Fabric, ya que la red será privada y permisionada. La red permitirá el registro de autores y actualizaciones a través de nodos de registro con los permisos para la creación de registros y transacciones. También permitirá la recuperación de las actualizaciones a través de nodos de recuperación con los permisos para recuperar y distribuir información de la red.

3.2 Gestión del alcance

El proyecto se desarrolla en el contexto del grupo de trabajo de DAIM del centro de investigación tecnológica Ceit. Constituye el inicio de un proyecto de alcance potencialmente mayor, una solución de software que permita la actualización segura de dispositivos IoT.

El TFG, sin embargo, tiene un alcance más modesto debido a las limitaciones asociadas a un trabajo de fin de grado. Por ello, se limita a realizar el diseño inicial de la solución incluyendo únicamente el proceso de registro y recuperación de imágenes en una red blockchain, de manera que se asegura la inmutabilidad de las mismas. Adicionalmente, se implementarán los elementos diseñados en la medida en la que la duración del TFG lo permita.

Se asumirá una relación y comunicación segura entre nodos y dispositivos, y que los autores participantes son agentes no maliciosos.

3.2.1 Objetivos

El TFG cuenta con los siguientes objetivos principales:

- Diseñar una red blockchain con distintos nodos para cada tarea. Deberá haber nodos que permitan el registro de imágenes y nodos que permitan la recuperación.
- Diseñar agentes que se encarguen de los procesos de registro, verificación y recuperación.
- Diseñar contratos inteligentes que permitan la comunicación entre los agentes y los nodos de la red.
- Diseñar un sistema de emulación de dispositivos IoT que emule los dispositivos que participan en el proceso de actualización.
- Implementar los agentes diseñados.
- Implementar los contratos inteligentes diseñados.

Objetivos secundarios: Estos objetivos son interesantes de plantear, pero no se espera que la duración del proyecto permita abordarlos.

- Desplegar la red blockchain con una configuración equivalente a la de un entorno real.
- Implementar el sistema de emulación de dispositivos.
- Diseñar e implementar una aplicación cliente que permita a los autores realizar los procesos de registro desde una interfaz gráfica de usuario (GUI).

Objetivos del alumno:

- Compresión y aprendizaje de tecnologías disruptivas: Blockchain e IoT son dos de las tecnologías recientes más interesantes. Su aprendizaje además de enriquecedor de manera personal puede facilitar la búsqueda de una posición laboral en el futuro. Se intentará obtener un conocimiento lo más profundo posible sobre dichas tecnologías.
- Experiencia laboral: obtener la máxima experiencia de trabajo posible en un entorno real en un centro de investigación con años de trayectoria. Aprender a optimizar reuniones y trabajo en equipo.
- Compresión del ciclo de trabajo: obtener la máxima experiencia posible en el ciclo de diseño e implementación de un proyecto completo, de manera que este luego aporte los máximos beneficios posibles en el mundo real.
- Acercamiento al mundo de la investigación académica: obtener conocimientos sobre cómo se lleva a cabo la investigación en el campo de la informática.

3.2.2 Requisitos

Funcionales

- El autor puede subir imágenes y manifiestos
- El autor se puede registrar
- El autor se debe autenticar
- Un dispositivo puede verificar versiones
- Un dispositivo puede solicitar un manifiesto
- Un dispositivo puede descargar una imagen

No Funcionales

- La solución funciona sobre una blockchain privada basada en Hyperledger Fabric
- La blockchain debe tener distintos tipos de nodos con diferentes roles
- El almacenamiento de imágenes será descentralizado pero off-chain.
- Los agentes incluyen keystores/almacenamientos temporales y métodos de interacción con los chaincodes

- Las actualizaciones deben ser inmutables.

Requisitos del centro

- Los elementos deben poder ser testeados mediante Swagger.
- Los agentes deben ser contenidos utilizando docker.
- Los agentes se desarrollarán con NodeJS.

Exclusiones

- El proceso de creación de imágenes y manifiestos es externo a este proyecto
- El uso e instalación de las imágenes es externo a este proyecto
- Se asume una relación de confianza entre nodo de recuperación y dispositivos
- Se asume que los autores registrados son de confianza

El proceso de decisión sobre la posibilidad de que el autor a registrar sea malicioso y el intercambio de información seguro entre nodo de recuperación y dispositivo, si bien son una parte importante de futuras iteraciones del proyecto, no forman parte de este TFG, que se centrara únicamente el proceso de registro y recuperación segura de imágenes a una red descentralizada.

3.3 Planificación y descomposición de tareas

En este apartado se definirán las tareas a realizar, dividiéndolas en fases y paquetes. Se asignarán unas estimaciones temporales de manera que se reduzcan los riesgos tanto como sea posible, teniendo en cuenta la incertidumbre presente en el proyecto.

3.3.1 Fases del proyecto

Debido al extenso alcance del proyecto, y al desconocimiento inicial de las tecnologías por parte del alumno, el proyecto se realizará siguiendo una aproximación incremental en su planificación. Se trabajará en tres fases, aunque se prevé que quizá no sea posible completarlas todas:

- Fase de diseño: Se diseñará la arquitectura del proyecto basándose en los trabajos anteriores y las propuestas disponibles en diversos artículos académicos. Se diseñarán los componentes de la solución teniendo en cuenta su interconectividad. La finalización de esta fase constituye el requisito mínimo de este proyecto.
- Fase de implementación del registro de actualizaciones: Se implementarán los agentes y contratos inteligentes diseñados para el proceso de registro de actualizaciones.
- Fase de implementación de la recuperación de actualizaciones: Se implementarán los agentes y contratos inteligentes diseñados para el proceso de recuperación de actualizaciones.

Aunque es altamente improbable que la duración del TFG permita elaborarlas, también es interesante plantear las siguientes fases, aunque sea a modo de trabajo futuro:

- Fase de despliegue de la red: Se despliega una red blockchain personalizada para emular el funcionamiento en un entorno real, sustituyendo las redes de prueba que se hayan utilizado en fases anteriores.
- Fase de implementación de emulador de dispositivos: Se implementará un emulador de dispositivos que permita completar el flujo completo del proceso de actualización.
- Fase de implementación de aplicación de autor: Se implementará una aplicación que mediante una interfaz gráfica permita al autor realizar el proceso de registrar una

actualización de manera sencilla. Dicha aplicación también podría gestionar el proceso de la creación de digests y firmas.

3.3.2 Diagrama de Estructura de Descomposición del Trabajo

En la figura 3.1 se muestra el EDT elaborado a partir de los paquetes y tareas definidos en el apartado anterior.

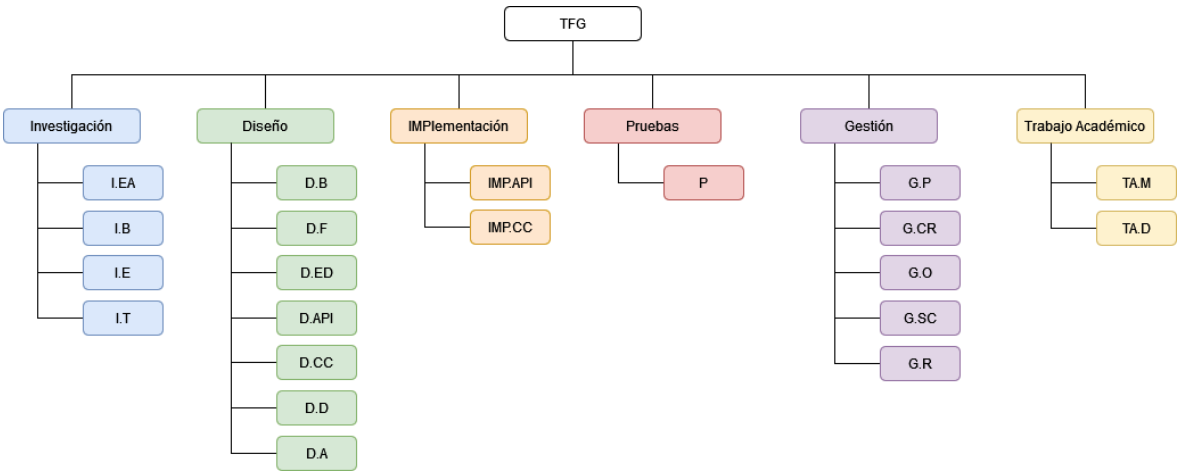

fig 3.1 - Diagrama de EDT

3.3.3 Paquetes de trabajo

El trabajo se dividirá en distintos paquetes, que agruparán las tareas de características similares. Cada paquete está a su vez dividido en subpaquetes. Estos son los paquetes junto a su descripción:

Investigación (I)

En este paquete se llevará a cabo la recolección necesaria para la comprensión del problema a resolver, el estudio de las soluciones propuestas previamente y en análisis y comparación de las distintas tecnologías a utilizar para implementar la solución. El resultado de este paquete de trabajo se ve reflejado en los capítulos de antecedentes y tecnologías a utilizar. Incluye los siguientes subpaquetes:

- Investigación del estado del arte (I.EA): Incluye la investigación del problema junto a las soluciones propuestas previamente.
    - I.EA 1: Análisis del problema a solucionar.
    - I.EA 2: Análisis de las soluciones propuestas con anterioridad.
- Investigación sobre Blockchain (I.B): Investigación sobre el funcionamiento de la tecnología de ledger distribuido y los conceptos criptográficos relevantes.
    - I.B 1: Estudio sobre el funcionamiento de la blockchain.
- Alternativas de emulación (I.E): Investigación sobre las tecnologías y técnicas disponibles para la emulación de dispositivos IoT.
    - I.E 1: Estudio de alternativas de emulación.
    - I.E 2: Elaboración de comparativa de alternativas.

- Investigación sobre Tecnologías a utilizar (I.T): Investigación sobre el funcionamiento y utilización de las tecnologías a utilizar para diseñar e implementar el proyecto.
    - I.T 1: Estudio sobre el funcionamiento de Hyperledger Fabric.
    - I.T 2: Estudio sobre el funcionamiento de IPFS.
    - I.T 3: Estudio del funcionamiento de Docker.
    - I.T 4: Estudio del funcionamiento de Swagger.
    - I.T 5: Estudio del funcionamiento de NodeJS

Diseño (D)

En este paquete se diseñan los componentes del proyecto y se generarán los documentos de diseño, como los diagramas UML. También contiene un componente de investigación, pero al contrario que en el paquete I, está enfocada a producir los diseños en lugar de a obtener conocimiento general.

- Diseño de red Blockchain (D.B): Se diseñarán los distintos tipos de nodo y las conexiones entre sí.
    - D.B 1: Diseño de arquitectura de red blockchain y almacenamiento
- Diseño de Flujos (D.F): Se diseñaran los flujos de información y tareas para los distintos procesos necesarios para el proyecto.
    - D.F 1: Diseño del flujo general
    - D.F 2: Diseño del flujo de registro de autores.
    - D.F 3: Diseño del flujo de registro de actualizaciones.
    - D.F 4: Diseño del flujo de recuperación de actualizaciones.
- Diseño de Estructuras de Datos (D.ED): Se diseñarán las distintas estructuras de datos que se utilizaran durante el proyecto:
    - D.ED 1: Diseño de estructuras.
- Diseño de APIs / agentes (D.API): Se diseñarán las APIs necesarias para el funcionamiento de los agentes. Corresponden al autor, y a los distintos tipos de nodos.
    - D.API 1: Diseño de agente autor
    - D.API 2: Diseño de agente de registro
    - D.API 3: Diseño de agente de recuperación
- Diseño de contratos inteligentes o chain codes (D.CC): Se diseñarán los contratos inteligentes encargados de registrar verificar y recuperar los manifiestos mediante transacciones en la red de blockchain.
    - D.CC 1: Diseño de smart contract de registro.
    - D.CC 2: Diseño de smart contract de recuperación.
- Diseño emulador de dispositivos (D.D): Se diseñará el modelo de emulación de dispositivos, eligiendo tecnología y comportamiento para los mismos. De ser necesario se escogerán las imágenes a utilizar.
    - D.D 1: Diseño de modelo de emulación de dispositivos.
- Diseño aplicación autor (D.A): se diseñará la aplicación que permita al autor subir las imágenes, mediante el uso de la API del agente autor.
    - D.A 1: Diseño de aplicación de autor.

Implementación (IMP)

En este paquete se implementarán los elementos diseñados en el paquete anterior.

- Agentes (IMP.API): Implementación de los agentes según el diseño.
    - IMP.API 1: Implementación del agente autor.
    - IMP.API 2: Implementación del agente de registro.
    - IMP.API 3: Implementación del agente de recuperación.
- Contratos inteligentes (IMP.CC): Implementación de los contratos inteligentes según el diseño.
    - IMP.CC 1: Implementación del contrato de registro.
    - IMP.CC 2: Implementación del contrato de recuperación.
- Documentación de elementos implementados (IMP.DOC): Se generará la documentación necesaria para el mantenimiento y futura extensión del proyecto, así como para su comprensión y utilización.

Pruebas (P)
En este paquete se harán pruebas para asegurar el correcto funcionamiento de los elementos desarrollados en el paquete anterior.

- Pruebas (P): se harán pruebas para asegurar el correcto funcionamiento de todas las tareas programadas.
    - P 1: Prueba de funcionalidades del agente del autor.
    - P 2: Prueba de funcionalidades del agente de registro.
    - P 3: Prueba de funcionalidades del agente de recuperación.
    - P 4: Prueba de funcionalidades del contrato de registro.
    - P 5: Prueba de funcionalidades del contrato de recuperación.

Gestión (G)
En este paquete se llevarán a cabo las tareas relacionadas con la elaboración del propio proyecto, destinadas a asegurar que se realizan correctamente el resto de tareas.

- Planificación (G.P): Planificación de todas las tareas que aparecen en este capítulo y que deben completarse para finalizar el proyecto.
    - G.P 1: Planificación de tareas a realizar según los requisitos de la aplicación.
- Captura requisitos (G.CR): Reunión con el director del proyecto para capturar correctamente todos los requisitos que debe cumplir el proyecto.
    - G.CR 1: Captura de requisitos funcionales.
    - G. CR 2: Captura de requisitos no funcionales.
    - G. CR 3: Captura de requisitos adicionales.
- Establecimiento de objetivos (G.O): Establecimiento de los objetivos que debe satisfacer la solución e implementar, y sobre los cuales se hará el diseño.
    - G.O 1: Establecimiento de objetivos principales.
    - G.O 2: Establecimiento de objetivos secundarios.
- Seguimiento y control (G.SC): Monitorización constante de los objetivos y tareas para evaluar la evolución del proyecto y ver si los objetivos que satisfacen según la planificación. Se realizará de manera continua.
- Reuniones (G.R): Reuniones periódicas para evaluar el progreso y resolver dudas que puedan surgir a lo largo del desarrollo del proyecto. Se planifican reuniones de seguimiento quincenales.

Trabajo Académico (TA)
En este paquete se incluyen las tareas relacionadas a la gestión académica del proyecto.

- Elaboración de la memoria (TA.M): Elaboración de la memoria a presentar después de completar el trabajo.
- Defensa del proyecto (TA.D): preparación y ejecución de la defensa del proyecto ante el tribunal.

3.3.4 Dependencias entre tareas

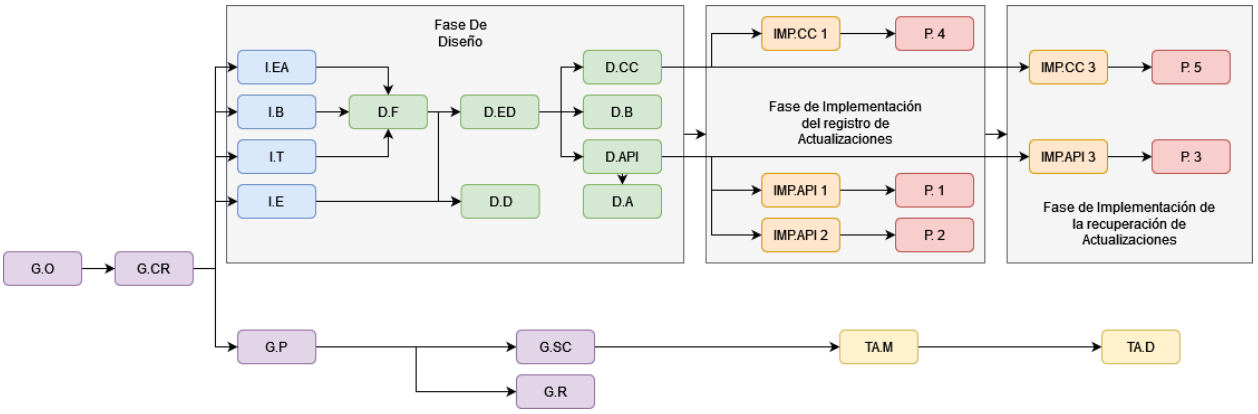

fig 3.2 - Diagrama de dependencias

En la figura 3.2 se reflejan las dependencias de los paquetes de trabajo.
Para poder comenzar el trabajo, es necesario capturar los requisitos y establecer los objetivos. Con los objetivos establecidos, por un lado es posible comenzar con la planificación del proyecto.

Una vez planificado puede hacerse el seguimiento y pueden realizarse las reuniones correspondientes. El seguimiento del proyecto también permite la elaboración de la memoria.

Por otro lado, con los objetivos establecidos, es posible comenzar con la investigación de los distintos elementos que conforman el proyecto. Cuando la investigación de un elemento es completada, se puede iniciar el diseño de los componentes relacionados. Una vez diseñados, es posible comenzar la implementación. Aunque no es estrictamente necesario implementar el proceso de registro antes que el de recuperación, es conveniente realizarlo de esta manera.

3.3.5 Diagrama de Gantt

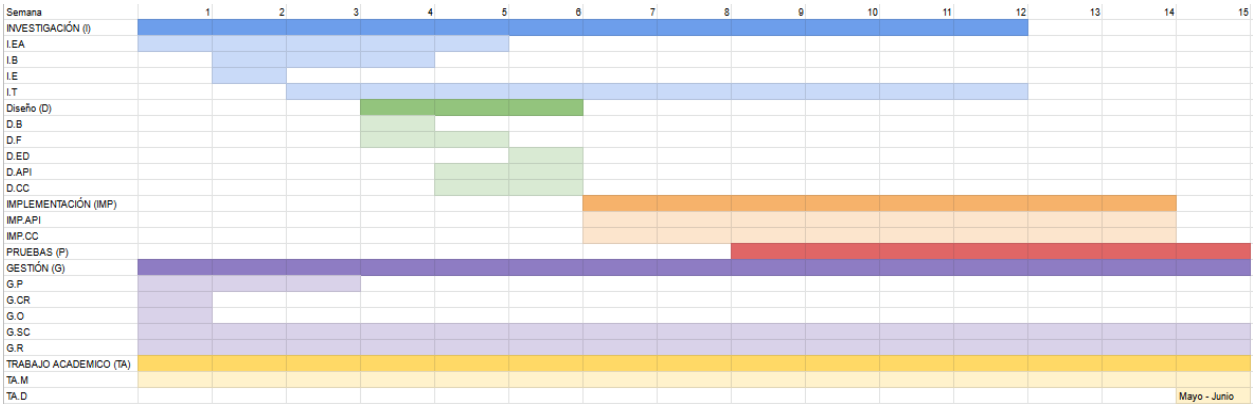

fig 3.3 - Diagrama de Gantt

En la figura 3.3 se puede apreciar el diagrama de Gantt para la planificación. El proyecto tendrá una duración de 15 semanas. El paquete de investigación se prolonga en el tiempo ya que se necesita investigar sobre el uso de las tecnologías con las que se implementa la solución a medida que se realiza la implementación.
El proyecto comienza con la planificación, captura de requisitos e investigación. Una vez realizada la investigación inicial se procede con el diseño, dando una duración de tres semanas a la fase 1. A partir de la semana 6 en adelante, casi todo el tiempo se dedicará a la implementación, dejando una última semana para la elaboración de pruebas.
La gestión y el trabajo académico se hacen de manera continua durante toda la elaboración del TFG.

3.3.6 Tiempo estimado a cada tarea

En este apartado se asigna una estimación de la duración para cada tarea.

Paquete	**Tarea**	**Dedicación (h)**
Investigación (I)	I.EA 1	10
	I.EA 2	10
	I.B 1	10
	I.E 1	8
	I.E 2	2
	I.T 1	10
	I.T 2	5
	I.T 3	2
	I.T 4	2

	I.T 5	3
I Subtotal : 62h		
Diseño (D)	D.B 1	10
	D.F 1	2
	D.F 2	2
	D.F 3	2
	D.F 4	2
	D.ED	6
	D.API 1	5
	D.API 2	5
	D.API 3	5
	D.CC 1	10
	D.CC 2	10
	D.D 1	4
	D.A 1	4
D Subtotal: 67h		
Implementación (IMP)	IMP.API 1	15
	IMP.API 2	15
	IMP.API 3	15
	IMP.CC 1	20
	IMP.CC 2	20
	IMP.DOC	15
IMP Subtotal: 100h		
Pruebas(P)	P 1	3
	P 2	3
	P 3	3
	P 4	4
	P 5	4

P Subtotal 17h		
Gestión (G)	G.P 1	10
	G.CR 1	1
	G.CR 2	1
	G.CR 3	1
	G.O 1	1
	G.O 2	1
	G.SC	10
	G.R	10
G Subtotal 35h		
Trabajo Académico (TA)	TA.M	60
	TA.D	15
TA Subtotal 75h		
Total: 356 Horas		

3.3.7 Gestión de riesgos

El proyecto podría presentar ciertos riesgos, por lo que es importante identificarlos para reducir su impacto en la medida de lo posible.

R1 - Tecnologias recientes y desconocidas para el alumno
En el proyecto se van a utilizar tecnologías de reciente aparición. Conlleva el riesgo de que no hay tanta información disponible y resultan difíciles de aprender inicialmente. Por otro lado, el desconocimiento previo por parte del alumno puede conllevar dificultades.
Prevención: Ejecutar un seguimiento y control estricto para detectar problemas y dificultades tan pronto como sea posible. Planificar el trabajo mediante fases incrementales, asegurando que se van obteniendo hitos que puedan ser defendidos como resultado del TFG
Plan de acción: Solicitar ayuda del tutor de proyecto cuando sea necesario, quien tiene conocimientos en las tecnologías a utilizar. En caso de desvíos significativos (25% del tiempo estimado**)**, realizar nueva planificación para evaluar si es necesario recortar tareas de implementación del alcance.

R2 - Riesgos personales
Es posible que se contraigan enfermedades, lesiones o situaciones personales que impidan dedicar tiempo al trabajo de manera presencial.

Prevención: Establecer métodos de comunicación adecuados y guardar el trabajo en repositorios en la nube para permitir el trabajo no presencial.
Plan de acción: Recuperar tiempo trabajando de manera no presencial en caso de que sea necesario. Si el tiempo a recuperar es significativo, realizar nueva planificación y evaluar si es necesario recortar el alcance del proyecto.

R3 - Planificación incorrecta
Es posible que la planificación realizada al inicio del proyecto no sea correcta, ya bien sea por una captura de requisitos poco acertada, un establecimiento de objetivos errado o algún otro tipo de inconveniente imprevisto.
Prevención: realizar una captura de requisitos lo más detallada posible y repasar los objetivos con ambos directores para detectar las carencias que puedan resultar. Planificar el proyecto con atención e intentar identificar todos los puntos que puedan causar problemas para estimar una serie de tareas y una duración de las mismas lo más acertada posible. Realizar un seguimiento y control continuo. El trabajo en ciclos iterativos también ayuda a mitigar este riesgo.
Plan de Acción: En caso de desvíos significativos (25% del tiempo estimado**)**, realizar nueva planificación para evaluar si es necesario recortar tareas de implementación del alcance.

3.3.8 Gestión de la calidad

Para valorar la evolución y los resultados del proyecto según los requisitos de las partes interesadas, se han establecido unos requisitos de calidad:

- El diseño generado debe quedar reflejado en figuras UML o flowcharts. Las figuras deben quedar almacenadas en el repositorio.
- El diseño debe tener en cuenta posibles mensajes de error.
- La implementación debe gestionar los errores más generales, aunque no es necesario un pulido del código al nivel de una aplicación comercial.
- La eficiencia de la solución no debe decrecer a medida que aumentan el tamaño de los archivos.
- Añadir una red blockchain personalizada. Opcional.
- Añadir una red IPFS privada. Opcional.

3.3.9 Sistemas de almacenamiento

Durante la elaboración del proyecto se irán generando diferentes documentos y ficheros. El almacenamiento seguro de dichos archivos es vital para asegurar que se cumplen los objetivos del trabajo. Por ello es necesario definir un sistema de almacenamiento.

Se definirán dos sistemas de almacenamiento principales:

- Google Drive: se creará una carpeta en la nube donde se contendrán todos los archivos y documentos relacionados a la investigación y elaboración de la memoria. Tendrá las siguientes subcarpetas:
    - Documentación: Contiene todos los documentos generados, como la memoria, comparativas y cualquier otro tipo de redacción.

    - Investigación: Contendrá todo el material utilizado en los componentes de investigación, como artículos, papers o enlaces a material relevante.
    - Diagramas: Contendrá todas las figuras y diagramas realizados.
    - Reuniones: Contendrá todo el material relacionado con las reuniones, como las actas generadas o los documentos previos.
- Repositorio Github: Se almacenará en todo lo relacionado a la implementación y documentación propia del proyecto software

Además, se contará con el almacenamiento local de los dispositivos utilizados en la elaboración del proyecto, que serán principalmente los siguientes:

- Ordenador de la oficina: Computadora asignada en el centro de Ceit para la elaboración del proyecto. Cuenta con acceso a un servidor para las tareas de implementación.
- Ordenador personal: Computadora personal del alumno utilizada para generar documentación.

Todos los archivos generados de manera local deberán ser copiados en la nube, ya sea en la carpeta de google drive o el repositorio github.

3.3.10 Sistemas de comunicación

La comunicación efectiva es un punto clave para el progreso del proyecto y para asegurar que el trabajo realizado avanza hacia el cumplimiento de los objetivos establecidos. Por ello, se establecen los siguientes métodos de comunicación:

- Correo electrónico: El principal método de comunicación será el intercambio de correos electrónicos. Es un método fiable que permite realizar la comunicación de manera asíncrona y permite compartir documentos, lo que resulta de gran ayuda para compatibilizar los diferentes horarios de los participantes. El correo electrónico además permite reenviar los mensajes ya enviados a otros participantes con facilidad.
- Reuniones presenciales: Para tratar temas más complejos y para realizar las tareas de seguimiento y control se harán reuniones presenciales. Serán planificadas de antemano y realizadas de manera periódica alrededor de cada dos semanas.
- Reuniones telemáticas: En caso de ser necesario, pueden organizarse reuniones de manera telemática utilizando herramientas como zoom o teams. Tiene la ventaja de que son más fáciles de organizar al poder realizarse de manera remota.
- Microsoft Teams: Permite la rápida comunicación entre el alumno y el tutor del centro para el intercambio de mensajes y documentos.

Las reuniones realizadas pueden ser de distinta naturaleza:

- Seguimiento y control: Se informará de los progresos realizados y se evaluará la evolución del proyecto y si se están alcanzado los objetivos establecidos. Se valorará si la planificación realizada se adapta a la realidad del proyecto y si es necesario realizar adaptaciones parciales o totales a la planificación en caso de ser necesario.
- Intercambio de información: El objetivo de estas reuniones es realizar un intercambio de información o solicitar ayuda por parte del alumno en caso de que tenga dudas que sea incapaz de resolver por su cuenta.

- Toma de decisiones: Estas reuniones se organizarán cuando surja la necesidad de tomar decisiones con gran impacto en el proyecto.

Todas las reuniones deberán ser recogidas en un acta que deberá ser almacenada de manera segura. Dicho acta deberá contener la fecha de la reunión, los asistentes a la misma y los temas tratados. El correcto seguimiento de las reuniones permitirá evaluar el progreso del proyecto y supone una manera eficaz de almacenar los temas tratados para su posterior consulta.

3.3.11 Stakeholders (Interesados)

En este apartado se recogen los participantes del proyecto, así como sus funciones e intereses dentro del mismo.

- Alumno : Jesús Rugarcía. Se encargará de la elaboración del proyecto. Tiene como objetivo la obtención de experiencia laboral, el aprendizaje de nuevas tecnologías y la obtención del título de grado.
- Tutor del centro: Santiago Figueroa. Marcará los objetivos a alcanzar dentro del proyecto así como las necesidades del mismo, y orientará al alumno en la resolución de las tareas y el aprendizaje de las tecnologías.
- Tutora de la universidad: Maider Azanza. Orientará al alumno el la elaboración de las tareas académicas para asegurar que cumple los requisitos solicitados por la universidad.

4 Tecnologías utilizadas

En este capítulo se enumeran las tecnologías utilizadas y se da una breve descripción de las mismas.

4.1 Hyperledger Fabric

Hyperledger Fabric [15] es la tecnología principal con la que se trabajará en el proyecto, y será utilizada para implementar la arquitectura diseñada.

Se trata de una plataforma de tecnología DLT de código abierto diseñada para uso empresarial. Tiene una estructura modular y configurable, y permite escribir contratos inteligentes en lenguajes de uso general, como pueden ser Go, Java y JavaScript. Fue desarrollada inicialmente por la fundación Linux como un framework para blockchain privadas de uso empresarial y es de código abierto.

Tiene una estructura permisionada, por lo que los nodos de la red son conocidos para el resto. Sus protocolos de consenso son personalizables y no requieren de una criptodivisa para su funcionamiento. Los miembros son enlistados a través de un Membership Service Provider (MSP).

Utiliza la arquitectura execute-order-validate, en la que las transacciones se ejecutan primero en un nodo, luego se ordenan al protocolo de consenso y finalmente se valida contra una política de la aplicación antes de realizar el commit al ledger.

También permite proteger la confidencialidad de los datos y contratos realizados en la red, así como la creación de subredes dentro de la red general.

4.1.1 Infraestructura

Hyperledger Fabric está concebida como una blockchain para uso empresarial, por lo que está diseñada con redes privadas o de consorcio en mente.
Una red blockchain construida en Hyperledger Fabric se compone de los siguientes elementos:

- Peers: Los nodos peers son aquellos que contienen una copia del ledger.
- Ledger: Base de datos que toma la forma de una cadena de bloques. Un bloque es una colección de transacciones enlazada por el bloque anterior mediante hashes.
- Endorsers: Los nodos endorsers son aquellos que ejecutan los contratos inteligentes. Todos los resultados deben coincidir para que el servicio de endorsement siga adelante.
- Ordereres: Los nodos orderers son aquellos que ejecutan los algoritmos de consenso y que generan los bloques de la red.
- Membership Service Provider (MSP): Un directorio utilizado en la configuración de la red que permite verificar las identidades de los participantes, tanto de manera interna (administradores y roles dentro de una entidad) como externa (confirmación de que un participante tiene autorización para ejecutar una tarea).
- Certificate Authority (CA): Autoridad encargada de otorgar identidades a los participantes de la red.

Una red contiene un Ordering Service, que puede estar compuesto por uno o más nodos orderers. Este servicio es el encargado de administrar la red. Contiene la configuración de los canales de la red, incluyendo sus políticas y miembros.

Un canal permite el intercambio de información mediante transacciones entre los miembros participantes del canal y las aplicaciones de la red. Para que un miembro participe en un canal debe autenticarse utilizando la identidad generada por el CA de la entidad. Los participantes del canal acuerdan las políticas del mismo en su configuración.
Los participantes de un canal pueden añadir peers al mismo. Todos los peers contienen una copia del ledger del canal. Algunos de estos peers pueden ser definidos como endorsers, lo que les permitirá ejecutar contratos inteligentes, también llamados chaincodes, y enviar información a las aplicaciones cliente.

4.1.2 Aplicación

Los contratos inteligentes se agrupan en aplicaciones o paquetes de contratos que reciben el nombre de Chaincode dentro de Hyperledger Fabric. Un chaincode es un programa que se ejecuta en un nodo endorser. Puede tener sus propias estructuras de datos y métodos, y se utilizan para establecer la lógica de negocio de la aplicación. Varios chaincodes distintos pueden ser ejecutados en la red. También existen chaincodes de ordenación, que son ejecutados por los nodos orderer.

La aplicación de Hyperledger Fabric está separada de la infraestructura. Una colección de contratos no depende de una red específica para su desarrollo. Para utilizarlo en una red, un chaincode debe ser instalado y posteriormente instanciado en un nodo. Por ello, es posible desarrollar la aplicación y desplegar la red de manera independiente.

Los chaincodes pueden ser desarrollados en distintos lenguajes como Java, Go o JavaScript (NodeJS), siendo este último el lenguaje a utilizar en este TFG.

La aplicación se conecta con el usuario real a través de una aplicación cliente. Dicha aplicación cliente se desarrolla utilizando el Software Development Kit (SDK) proporcionado por Hyperledger Fabric, y a menudo toma la forma de una API. Dicha API puede luego ser utilizada por una aplicación externa desarrollada para el usuario final o mediante herramientas como Postman o Swagger. En este TFG, los agentes representan la aplicación cliente de la red.

Cuando un usuario solicita una tarea en la red a través de una aplicación cliente, la solicitud se envía a un nodo endorser, que ejecuta la aplicación o chaincode contra su versión del ledger. El resultado es enviado como una solicitud de transacción al Ordering Service. La solicitud pasa entonces por el proceso de consenso. El bloque propuesto es enviado a todos los nodos para que verifiquen la transacción. En caso de que haya consenso, todos los ledgers son actualizados de manera asíncrona, ya que el proceso puede tardar segundos.

4.2 InterPlanetary File System (IPFS)

La capacidad de almacenamiento limitada de la tecnología blockchain en el momento de elaboración de este proyecto dificulta enormemente guardar las actualizaciones completamente en la red. Aunque es posible, la longitud considerable de las imágenes podría ralentizar muy significativamente la velocidad de transacción y consenso de la red. Por ello, se utilizará otra tecnología de almacenamiento para guardar las imágenes.

Debido a que uno de los problemas que se quiere evitar es la inclusión de un punto de fallo único, la utilización de almacenamiento centralizado para las imágenes se interpone con la filosofía de diseño descentralizado de la blockchain. Por ello, se considera conveniente utilizar una tecnología de almacenamiento distribuido P2P.

IPFS [19] es una de las tecnologías de almacenamiento distribuido más destacadas. Se trata de un sistema P2P que utiliza direccionamiento de contenidos para buscar entre los nodos de la red, en lugar de en una dirección establecida. Cuando un contenido es añadido a la red IPFS se le asigna un identificador CID, o Content IDentifier. Cuando un nodo busca un archivo, consulta a sus pares a través del CID. Cuando consigue el contenido, el nodo guarda una copia que comparte con sus pares.

Por otro lado, los contenidos almacenados no son modificables, lo que añade la propiedad de inmutabilidad. Debido a los intereses del proyecto resulta una alternativa de almacenamiento muy interesante por este motivo, ya que las imágenes no podrán ser alteradas. Cuando se añade una nueva versión de un archivo, esta obtiene su propio CID.

Existen dos tipos de nodos. Los nodos bootstrap tienen la función de indicar al resto de nodos que se conectan a él la dirección de otros nodos. Los nodos cliente son aquellos que intercambian información.

Se puede crear una red privada configurando los nodos bootstrap. Para ello es necesario generar una swarm key y añadirla a la configuración de todos los nodos participantes. Los nodos que pertenecen a una red privada solo interactúan con otros nodos que pertenecen a esa red privada.

4.3 Tecnologías para el desarrollo de agentes

Los agentes a desarrollar están compuestos por una serie de APIs que deben interactuar con la red blockchain. Para su desarrollo se utilizarán distintas herramientas.

4.3.1 NodeJS

NodeJS [21] es un entorno de ejecución JavaScript destinado a la creación de aplicaciones de red. Es un entorno asíncrono y basado en eventos, lo que facilita la escalabilidad de las aplicaciones. Actualmente es una de las herramientas más populares para el desarrollo de aplicaciones web.

Adicionalmente, NodeJS cuenta con una serie de librerías integradas que facilitan y estandarizan ciertas tareas, como pueden ser la lectura y escritura de archivos en el caso de la librería fs. Es destacable la librería crypto, que gestiona todos los procesos de cifrado y firma más utilizados.
También cuenta con un gestor de repositorios llamado node package manager (npm). Gracias a npm es posible descargar e instalar fácilmente librerías creadas por terceros, y cuenta con un sistema de auditoría que detecta vulnerabilidades en las mismas. Algunas de estas librerías suponen el estándar para trabajar con NodeJS y gestionan procesos muy relevantes, como el caso de Express para la creación sencilla de APIs o Axios para la gestión de peticiones HTTP desde la aplicación.

4.3.2 Swagger

Swagger [20] es una colección de herramientas destinadas a facilitar y guiar el desarrollo de REST APIs en multitud de lenguajes.
Entre ellas se encuentra el editor, que permite escribir documentación y se adhiere a los principios de diseño de API establecidos por el estándar OpenApi, cuyas versiones anteriores tomaban el nombre de Swagger precisamente. OpenApi permite definir el comportamiento de una (REST) API en un lenguaje comprensible para humanos y máquinas, utilizando los formatos yaml o json. Se establecen las rutas y las llamadas, incluyendo los cuerpos de llamada y respuesta, y los esquemas de datos asociados.
También incluye Swagger UI, que permite generar documentación digital para la API diseñada, y diferentes herramientas como ReadyAPI para realizar pruebas sobre las APIs una vez están desplegadas en un contenedor Docker de manera automática y continua.
Para el proyecto se ha utilizado la librería Swagger destinada a ser utilizada en conjunto con Express.

4.4 Docker

Docker [18] es una plataforma que permite automatizar el proceso de instalación de una aplicación de software mediante el uso de contenedores e imágenes.

El uso de docker comienza con una imagen. Una imagen es un archivo de solo lectura que contiene las instrucciones para crear un contenedor. Generalmente, las imágenes se construyen sobre otras imágenes ya creadas, añadiendo capas por encima. Por ejemplo, Docker provee de una imagen Ubuntu que contiene las instrucciones para crear un contenedor que ejecute el sistema operativo. Una nueva imagen basada en esta podría instalar adicionalmente un host de servidores, librerías o una aplicación.

A partir de la imagen se construye un contenedor, que es un entorno de ejecución ligero y una instancia de la imagen. Se puede conectar a redes y se le puede añadir almacenamiento, permitiendo ejecutar aplicaciones en entornos de prueba y producción.

Docker resulta útil en contextos de integración y despliegue continuo (CI/CD por sus siglas en inglés).

Compose

Se trata de una herramienta para definir y ejecutar aplicaciones docker que utilizan más de un contenedor. Con el uso de la herramienta se pueden ejecutar distintas imágenes simultáneamente, y se pueden terminar los contenedores de la misma manera.

5 Diseño

En este capítulo se describirán los diseños realizados y las decisiones tomadas para los distintos elementos que componen el proyecto, y se incluyen los diagramas y figuras generados.
Los diseños aquí presentados componen la idea inicial del proyecto y están cimentados sobre los conocimientos adquiridos durante la fase de investigación. En primer lugar se describen las amenazas que el proyecto intenta afrontar, seguidas por la arquitectura de la solución. Después se describen los distintos flujos que forman parte de la solución. Finalmente se describe la aplicación que se construye sobre la arquitectura, que tiene tres partes: la aplicación cliente permite a los usuarios interactuar con la solución, la aplicación blockchain es el conjunto de métodos que se ejecutan dentro de la red blockchain, y el emulador de dispositivo es una herramienta auxiliar para los testeos de la aplicación. Para cada una de estas partes de la aplicación se describe su participación en los flujos definidos.

5.1 Diseño de la arquitectura

El elemento principal a diseñar es la arquitectura de la solución. Basándose en otras soluciones previamente propuestas en el ámbito, la arquitectura estará construida alrededor de una red blockchain descentralizada que sustituye al servidor central propuesto en el RFC 9019.

5.1.1 Amenazas

Antes de comenzar con el diseño, es imprescindible conocer las amenazas y vulnerabilidades a las que se enfrentan los dispositivos IoT en su proceso de actualización. Por ello, se ha recopilado una lista con algunos de los más relevantes, con el fín de tener una visión clara del problema a solucionar.
El abanico de ataques descubiertos que pueden ser utilizados en contra de los dispositivos IoT es muy amplio, por lo que resulta imposible crear una solución que permita protegerse ante todos en el alcance de este TFG. Por ello, nos centraremos en crear una solución que intente solventar solo algunos de ellos. Concretamente, nos centramos en los ataques relacionados con el proceso de actualización en sí mismo, valiéndonos de las propiedades de inmutabilidad que ofrecen las redes blockchain. Algunos de los ataques y problemas que se propone enfrentar son los siguientes [22]:

- **Instalación de imagen antigua:** Un atacante envía una actualización antigua pero válida al dispositivo. El dispositivo la identifica como válida y la instala. El atacante conoce las vulnerabilidades de la versión enviada, por lo que puede realizar otros ataques. Aliviaremos este ataque con la inclusión de campos en el manifiesto que permitan verificar que la imagen a instalar es más reciente que la instalada actualmente. En última instancia, sin embargo, será la lógica del dispositivo la que tome la decisión de instalar una actualización.
- **Dispositivo no conectado:** Un atacante envía una imagen más reciente que la instalada a un dispositivo no conectado a internet. El dispositivo no puede determinar que no es la versión más reciente disponible y la instala. El atacante conoce las vulnerabilidades de la versión, que pueden haber sido corregidas en versiones más recientes, por lo que puede realizar otros ataques. Abordaremos este problema haciendo que los dispositivos solo instalen imágenes que ellos mismos han solicitado a la red descentralizada, que solo puede

enviar la última versión disponible. Nuevamente es la lógica del dispositivo la que tiene la decisión final.

- **Clase de dispositivo incorrecta:** El atacante envía una imagen válida y más reciente que la actual a un dispositivo, pero el firmware no es válido para esa clase de dispositivo. Dependiendo de la similitud del dispositivo con la clase para la que está diseñado el firmware, la actualización podría crear nuevas vulnerabilidades o hacer que sea incapaz de funcionar en su totalidad. Enfrentaremos este riesgo añadiendo la necesidad de que el dispositivo conozca su clase y añadiendo en el manifiesto un campo que identifique para qué clases de dispositivo es válida la actualización. Esta clase será necesaria para solicitar actualizaciones a la red.
- **Clase de payload incorrecta:** El dispositivo intenta realizar la actualización pero procesa la imagen o payload de manera incorrecta. Un atacante podría enviar intencionadamente una imagen o archivo incorrecto para actualizar el dispositivo, lo que puede hacer que deje de funcionar en su totalidad. Para evitar este riesgo, se añaden campos para indicar la codificación de la imagen/payload y si es necesario, los pasos a seguir para instalarla.
- **Directorio incorrecto de actualización:** El dispositivo instala la imagen en un directorio incorrecto, lo que puede hacer que su funcionamiento se vea estropeado. Para evitarlo, se añade un campo al manifiesto para indicar donde debe ser colocada la imagen/payload.
- **Redirección maliciosa:** A menudo los manifiestos contienen una URI para indicar la dirección desde donde se puede obtener el payload de la actualización. Un atacante puede modificar el manifiesto para indicar una dirección maliciosa, o puede atacar al almacenamiento para sustituir el payload por un contenido malicioso. Para evitarlo, se añade una firma para verificar los contenidos del manifiesto, y se implementa una red descentralizada para la recuperación de manifiestos, de manera que los nodos de recuperación son quienes se encargan de distribuir las imágenes. Se añade también una firma para verificar el payload, de manera que los dispositivos pueden verificar ambas partes de la actualización antes de proceder a la instalación.
- **Información adicional en la imagen / modificación de imagen:** Un atacante podría modificar una imagen para que contenga información adicional, como una pieza de código que explote o cree vulnerabilidades. Para evitarlo, se añade una firma para el payload y se incluye un campo en el manifiesto con el digest, de manera que el dispositivo pueda verificar que la imagen descargada no ha sido modificada desde que el autor firma el manifiesto. Por otro lado, las imágenes se almacenan en IPFS, que cuenta con la propiedad de que los archivos almacenados son inmodificables, ya que las nuevas versiones de un archivo reciben un nuevo identificador, que ya no quedará reflejado para la recuperación.
- **Modificación de manifiestos:** Un atacante que obtenga una actualización podría modificar su manifiesto para encubrir otros cambios realizados en la imagen o para interferir con la actualización, potencialmente utilizando el dispositivo. Para evitarlo, los manifiestos son firmados e incluyen su digest (que se elimina antes de verificar), de manera que se puede verificar que un manifiesto no ha sido modificado a efectos prácticos desde que su autor lo firma. Adicionalmente, la propiedad de inmutabilidad del ledger de la red blockchain asegura que una vez ingresan en la red no pueden ser alterados.
- **Desaparición del autor:** Cuando un autor desaparece del mercado, a menudo retira o deja de hacer disponibles las actualizaciones para sus dispositivos, de manera que aquellos dispositivos que no han recibido todavía las actualizaciones más recientes pierden permanentemente el acceso a las mismas. Con el uso de una red descentralizada,

garantizamos que las actualizaciones que ya han sido registradas siguen permaneciendo disponibles para los dispositivos. Adicionalmente, el autor puede ceder la responsabilidad a un nuevo autor otorgándole su clave privada o realizando una última actualización en la que se indique a sus dispositivos la nueva clave pública que deben utilizar. En el futuro podría ser necesario especificar un protocolo más concreto para este proceso.

- **Denegación de servicios:** La arquitectura descentralizada propuesta ofrece la ventaja de ofrecer múltiples puntos de fallo, lo que dificulta el trabajo de un atacante a la hora de ejecutar un ataque de denegación de servicios. Aunque al igual que con un servidor centralizado, es posible atacar todos sus puntos a la vez, pero resulta más complicado, ya que el atacante debe conocer todos los endpoints de la red y atacarlos simultáneamente. De la misma manera, es más sencillo para la red atender múltiples peticiones honestas, ya que puede distribuir la carga entre más endpoints.
- **Pérdida de información:** al contrario que con una red centralizada, una arquitectura distribuida cuenta con múltiples puntos de fallo, lo que supone una capa de protección adicional contra la pérdida de información. Los manifiestos quedan almacenados en múltiples copias del ledger, mantenidas por almacenamientos independientes, y cada uno de estos además almacena todos los cambios producidos, por lo que es imposible (teóricamente) eliminar información de manera intencionada. De manera similar, las imágenes son almacenadas en IPFS, por lo que existen múltiples copias idénticas en almacenamientos independientes y accesibles para la aplicación cliente.

5.1.2 Arquitectura general

Una vez identificados los problemas a resolver y las capacidades de las herramientas a utilizar, así como las propuestas realizadas con anterioridad (descritas en los capítulos 2 y 4), podemos plantear una visión general del funcionamiento de la red blockchain de la solución.
Se ha identificado la necesidad de introducir los siguientes participantes dentro de la solución:

- Autor: creador de las imágenes y manifiestos que componen la actualización.
- Nodo de registro: nodo que almacena las actualizaciones.
- Nodo de general: nodo que ejecuta tareas de mantenimiento de la red y/o contiene el ledger de la red.
- Nodo de recuperación: nodo que recupera las actualizaciones y las envía a los dispositivos.
- Dispositivos y gateways: destinatarios de la actualización.

Para evitar un tráfico excesivo en la red, se designaran nodos de registro con la capacidad de registrar manifiestos enviados por los autores en el ledger de la red. De la misma manera, se designaran nodos con la capacidad de recuperar dichos manifiestos y enviarlos a los dispositivos o a los gateways a los que están conectados. El mantenimiento de la red quedará a cargo de los nodos generales, que no podrán registrar ni recuperar los manifiestos, pero que llevarán a cabo otras tareas de mantenimiento.
Esta separación de tareas también minimiza riesgos de seguridad, ya que una única parte comprometida no tiene acceso a todo el proceso de actualización.

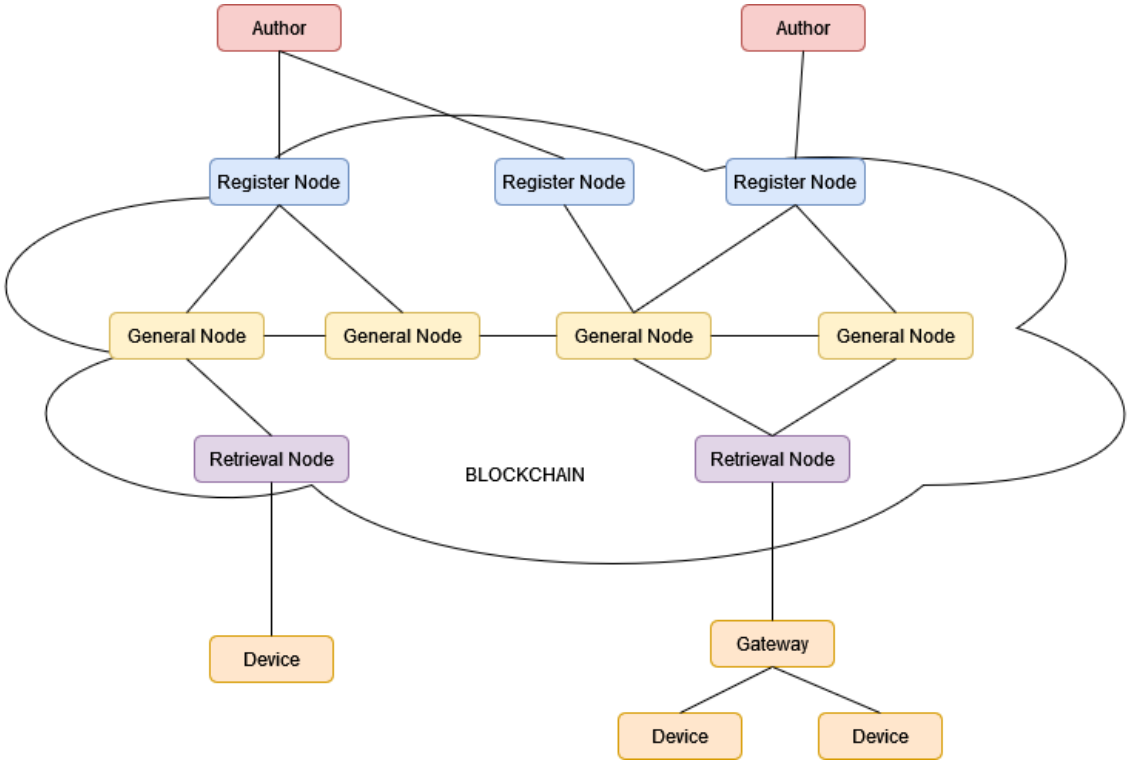

fig 5.1 - Arquitectura general de la solución

Las imágenes de firmware de las actualizaciones, sin embargo, no serán almacenadas en la red debido al retraso que pueden causar al aumentar de manera significativa el tamaño de los bloques. Se almacenarán en un sistema off-chain preferentemente distribuido, y se guardará la dirección de recuperación que será utilizada por los nodos de recuperación.

Una representación visual de esta arquitectura puede verse en la figura 5.1. Como puede apreciarse, los autores tienen acceso a la red a través de los nodos de registro. Un autor puede interactuar con los nodos de registro que desee, y un nodo de registro puede atender a cuantos autores realicen peticiones a través de él. Los nodos de registro se conectan con los nodos generales, que mantienen la red. Por otro lado, los dispositivos también pueden acceder a cualquier nodo de recuperación, que obtienen la información de las actualizaciones que se ha propagado por la blockchain. Los nodos generales sólo pueden interactuar con otros nodos y son inaccesibles desde fuera de la red.

5.1.3 Arquitectura de la solución

La arquitectura general debe ser adaptada a las tecnologías a utilizar para poder ser implementada. En esta solución se utilizarán Hyperledger Fabric como tecnología blockchain e IPFS como sistema de almacenamiento. La adaptación realizada puede observarse en la figura 5.2.

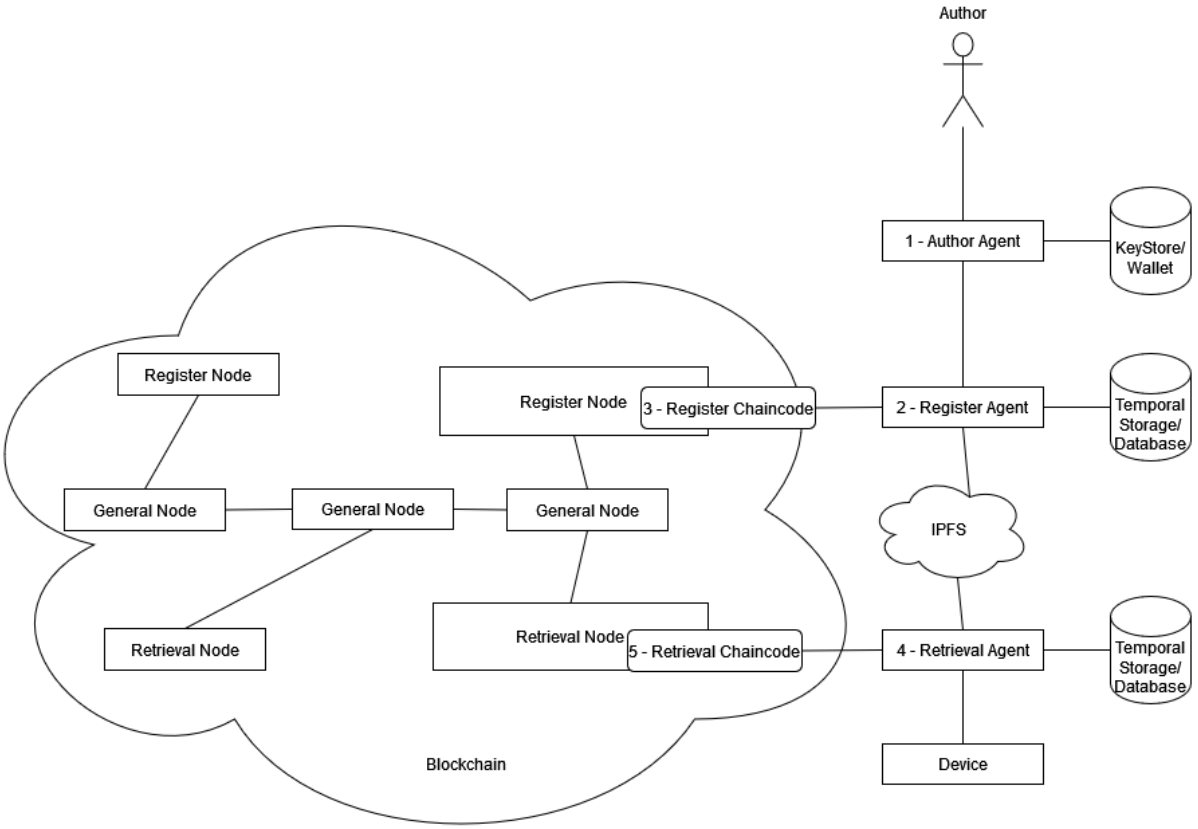

fig 5.2 - Arquitectura de la solución

En la figura se puede ver la red blockchain, que está compuesta por nodos de registro, encargados de registrar los autores y las actualizaciones subidas por estos, los nodos generales, que mantienen la red, y los nodos de recuperación, que recuperan las actualizaciones y las envían a los agentes. Los nodos generales son aquellos que mantienen una copia del ledger de la red, los nodos endorser y los nodos de ordenación, pero que no tienen los permisos para almacenar ni distribuir transacciones. Se utilizará un solo canal en el que participarán todos los nodos de la red. Por este motivo, todos tendrán una copia del mismo ledger.

Ahora, sin embargo, tenemos nuevos elementos en la arquitectura que corresponden con las aplicaciones de Hyperledger Fabric. La parte externa a la red, que es la que utilizan los usuarios, se llama aplicación cliente. Esta aplicación cliente se comunica con la red, y obtiene y almacena información de la red. Sin embargo, no puede hacerlo directamente. Para facilitar esta comunicación existe la aplicación de la red, que se ejecuta en los nodos que la componen. Está compuesta por contratos inteligentes, agrupados en un chaincode en Hyperledger Fabric.

Podemos destacar los siguientes elementos:

1. El agente de Autor está compuesto por una API que se comunica con un almacenamiento de claves. Forma parte de la aplicación cliente. Este agente permite al autor de las actualizaciones de software comunicarse con la red para registrarse en ella y para registrar actualizaciones. Guarda las claves que necesita el autor para verificar su identidad.

2. El agente de Registro está compuesto por una API. Forma parte de la aplicación cliente y es externo a la red blockchain. Su función es la de recibir las actualizaciones enviadas por el autor, verificarlas junto a la identidad de autor, y subirlas a la red. Los manifiestos serán almacenados en la red blockchain, mientras que las imágenes en la red IPFS, con la que el agente está conectado. También tiene la función de actualizar los manifiestos para indicar el CID de la imagen. Las actualizaciones son almacenadas en la base de datos hasta que son subidas a la red.
3. El chaincode de Registro es ejecutado por los nodos de registro y forma parte de la aplicación. Su función es atender a las peticiones del agente de registro para almacenar manifiestos. Solicita a la red almacenar los manifiestos en una transacción.
4. El agente de Recuperación está compuesto por una API. Forma parte de la aplicación cliente. Su función es la de comunicar la última versión disponible en la red para los dispositivos, y la de recuperar las actualizaciones y enviarlas a los dispositivos. Estas actualizaciones son almacenadas en la base de datos hasta que son enviadas a los dispositivos.
5. El chaincode de Recuperación se ejecuta en los nodos de recuperación y forma parte de la aplicación. Es el encargado de atender las peticiones del agente de recuperación para obtener información de los manifiestos en la red mediante queries al ledger.

El uso de Hyperledger Fabric como tecnología blockchain nos permite la implementación de una red privada y permisionada que no necesita de divisas nativas para su funcionamiento. En lugar de algoritmos de consenso costosos como la prueba de trabajo, utiliza algoritmos de consenso más simples, y solo ejecuta las comprobaciones en nodos determinados llamados endorsers.

Esto permite crear una red veloz y segura a la que solo tendrán acceso aquellos participantes que acuerden su implementación, como podría ser un grupo de fabricantes de dispositivos y las empresas que contratan sus servicios e instalan sus dispositivos, en por ejemplo, sistemas de vigilancia para naves industriales.

5.2 Flujos de la aplicación

En este apartado se detalla el funcionamiento de los flujos de la solución. Estos flujos describen el comportamiento de los componentes descritos en el apartado anterior y muestran el intercambio de información entre ellos, dando una imagen del comportamiento de la solución en su conjunto a la hora de llevar a cabo el proceso de actualización. En este apartado se describen los flujos de manera general, mientras que en las siguientes secciones se detalla cómo participa cada elemento de la solución dentro de los mismos.

5.2.1 Flujo General

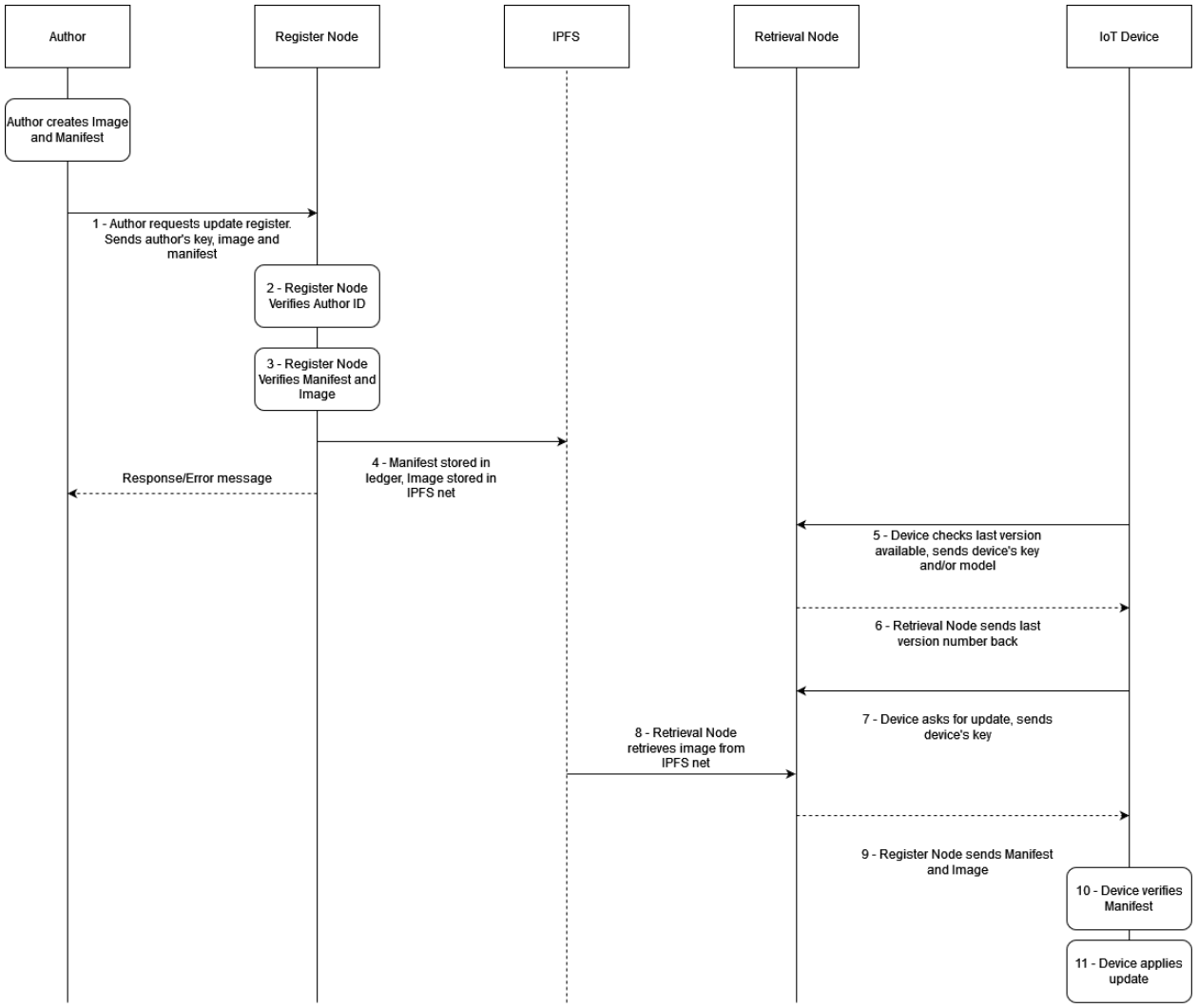

fig 5.3 - Proceso de actualización

En primer lugar, se describe una visión general del flujo de la aplicación en su conjunto. La figura 5.3 muestra gráficamente el intercambio de información entre los participantes de la solución en los procesos de registro y recuperación de la actualización, descritos de manera unificada. El proceso ocurre de la siguiente manera:

1. El proceso de actualización comienza cuando el autor solicita subir una actualización a la red. Para ello, debe enviar al nodo de registro su clave y firmas junto con la imagen y manifiesto de la actualización.
2. El nodo de registro verifica la identidad del autor.
3. El nodo de registro verifica la imagen y el manifiesto
4. El nodo de registro sube la imagen al almacenamiento off-chain y el manifiesto a la red, junto con el identificador de contenidos generado por la red IPFS.
5. El dispositivo solicita la versión más reciente al nodo de recuperación.
6. Nodo de recuperación envía la versión (numérica, no la imagen).
7. Si el dispositivo determina que su versión es inferior, solicita el manifiesto.

8. El nodo de recuperación recupera el manifiesto de la red y la imagen del almacenamiento distribuido y los verifica.
9. El nodo de recuperación envía la imagen y el manifiesto.
10. El dispositivo verifica la imagen y el manifiesto.
11. El dispositivo ejecuta la actualización.

5.2.2 Flujo del proceso de registro de autor

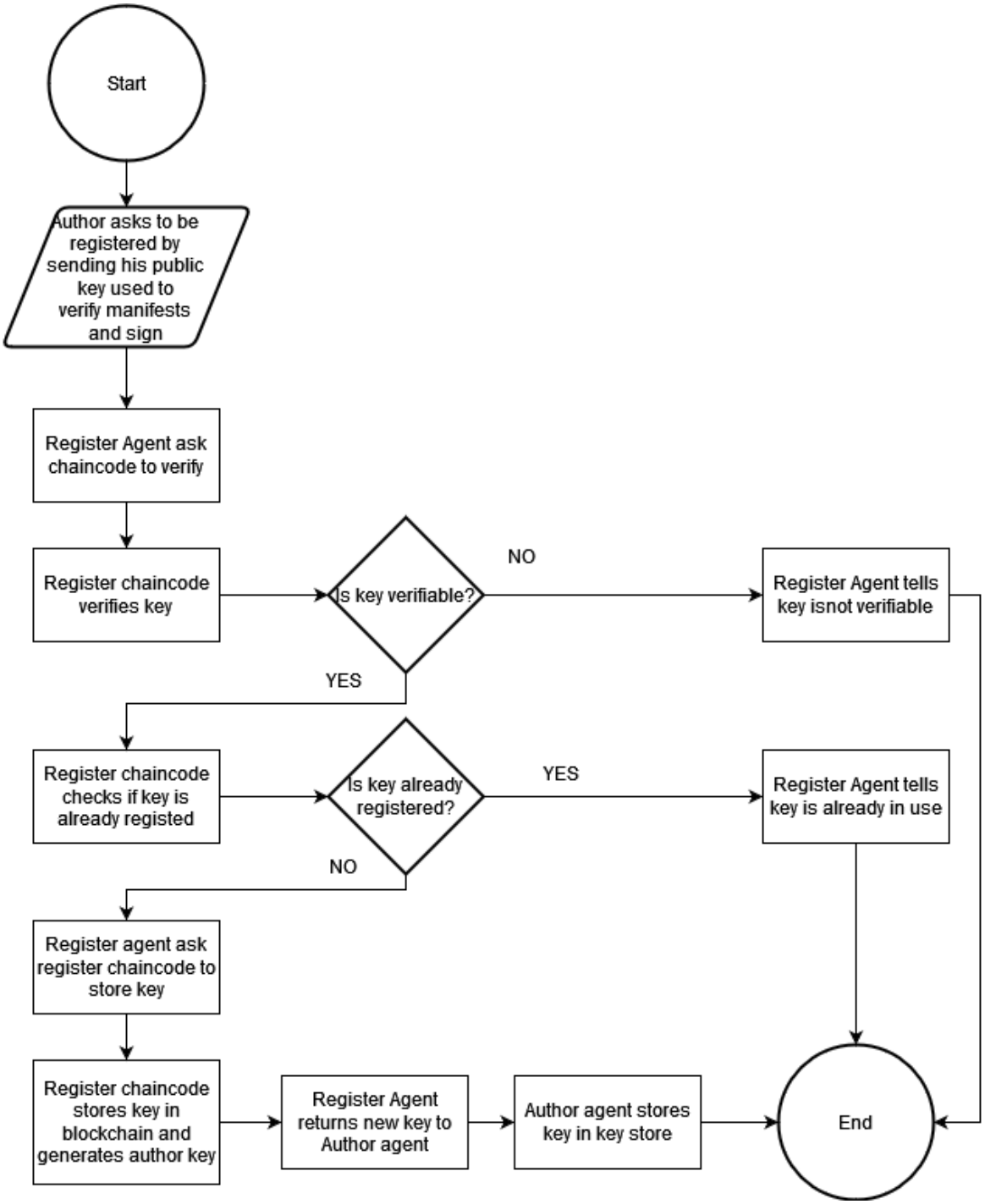

fig 5.4 - Flujo del registro de autores

La figura 5.4 describe el flujo de registro de autores. Este proceso es necesario para que la solución pueda verificar la identidad de un autor que solicita registrar actualizaciones a la red blockchain. A través del mismo autores previamente no identificados demuestran su identidad y dejan su clave pública almacenada en la red, convirtiéndolos en participantes. El proceso ocurre de la siguiente manera: el autor solicita registrarse enviando al agente de registro la clave pública que utiliza para firmar los manifiestos, y un mensaje firmado con su clave privada y sin firmar. El agente de registro la envía al chaincode de registro para que este la almacene en la red. El chaincode de registro utiliza la clave pública para verificar que la firma contiene el mensaje enviado. Si lo contiene, se puede verificar la identidad del autor. Si la clave ya está registrada el agente de registro lo notificará al autor.

Si la clave es válida y se produce el registro, se genera una nueva clave de registro, que el autor debe usar en sus futuras peticiones de registro de actualizaciones. Esta clave es devuelta por el agente de registro al agente autor para que la almacene en su almacenamiento de claves.

En versiones futuras del proyecto, sería interesante el uso de identificadores distribuidos [24].

5.2.3 Flujo del proceso de registro de actualización

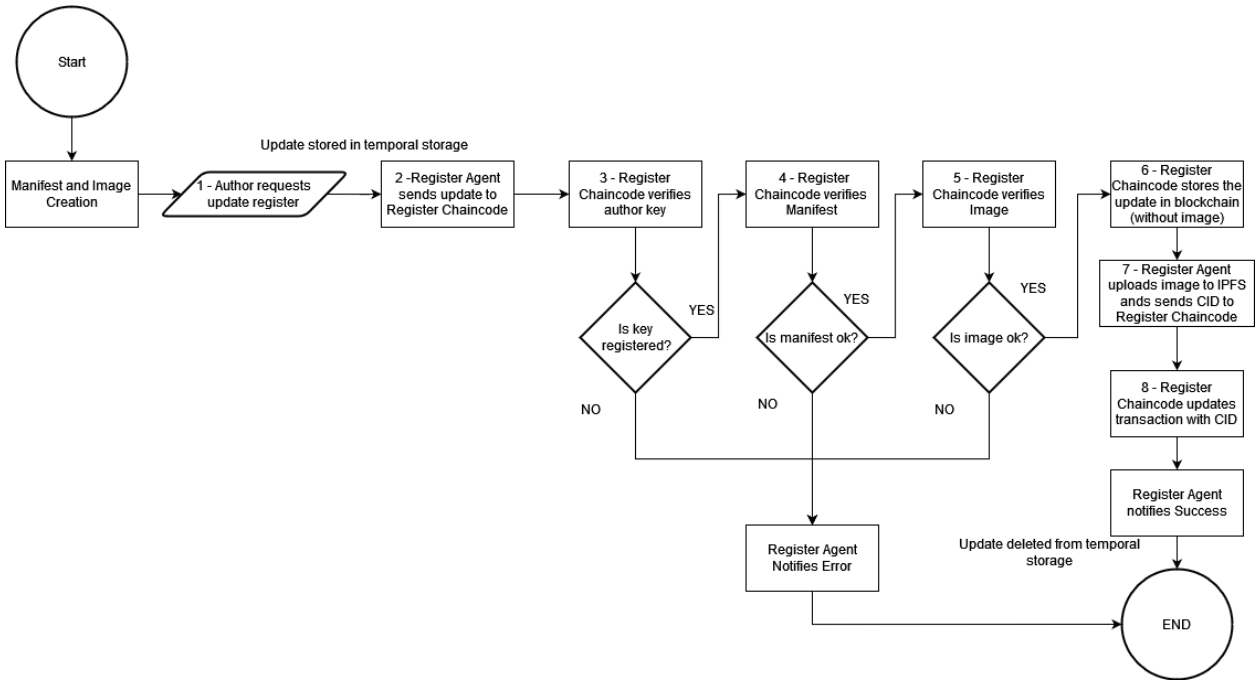

fig 5.5 - Flujo de registro de actualizaciones

Una vez registrado, el autor puede iniciar el proceso de registro de actualizaciones. Para ello, el autor crea la actualización y después comienza el proceso, como se ilustra en la figura 5.5. Se siguen los siguientes pasos:

1. El autor solicita iniciar el proceso de registro de actualizaciones enviando su clave, sus firmas, el manifiesto y la imagen. La actualización queda almacenada en el almacenamiento temporal del agente de registro.
2. El agente de registro envía la actualización al chaincode de registro para que haga la verificación On-chain.

3. El chaincode de registro verifica que la clave del autor está registrada en la blockchain. Si no lo está lo notifica y termina el proceso.
4. Si el autor está registrado, el chaincode de registro verifica el manifiesto. Para ello utiliza la firma y la clave asociada al autor que realiza la petición para obtener el digest del manifiesto, que también obtiene al "digerir" el propio manifiesto con la clave del autor. Si ambos digest son idénticos e iguales al presente en el manifiesto, se puede saber que el manifiesto no ha sido alterado desde la firma del autor. También verifica que contenga los campos necesarios. Si el manifiesto no es correcto o no es verificable, notifica el error y termina el proceso.
5. Si el manifiesto es correcto, el chaincode de registro verifica la imagen con la clave del autor siguiendo el mismo proceso que con el manifiesto, utilizando el digest de la imagen presente en el manifiesto. Si no es verificable, notifica el error y termina el proceso.
6. El chaincode de registro guarda la actualización en la blockchain y notifica al agente de registro.
7. El agente de registro la almacena en la red IPFS, lo que genera un CID. Este CID es enviado al chaincode de registro.
8. El chaincode de registro actualiza la transacción y añade el CID de la imagen. Se envía la notificación de que el proceso ha sido exitoso. La actualización es eliminada del almacenamiento temporal.

5.2.4 Flujo del proceso de recuperación de actualización

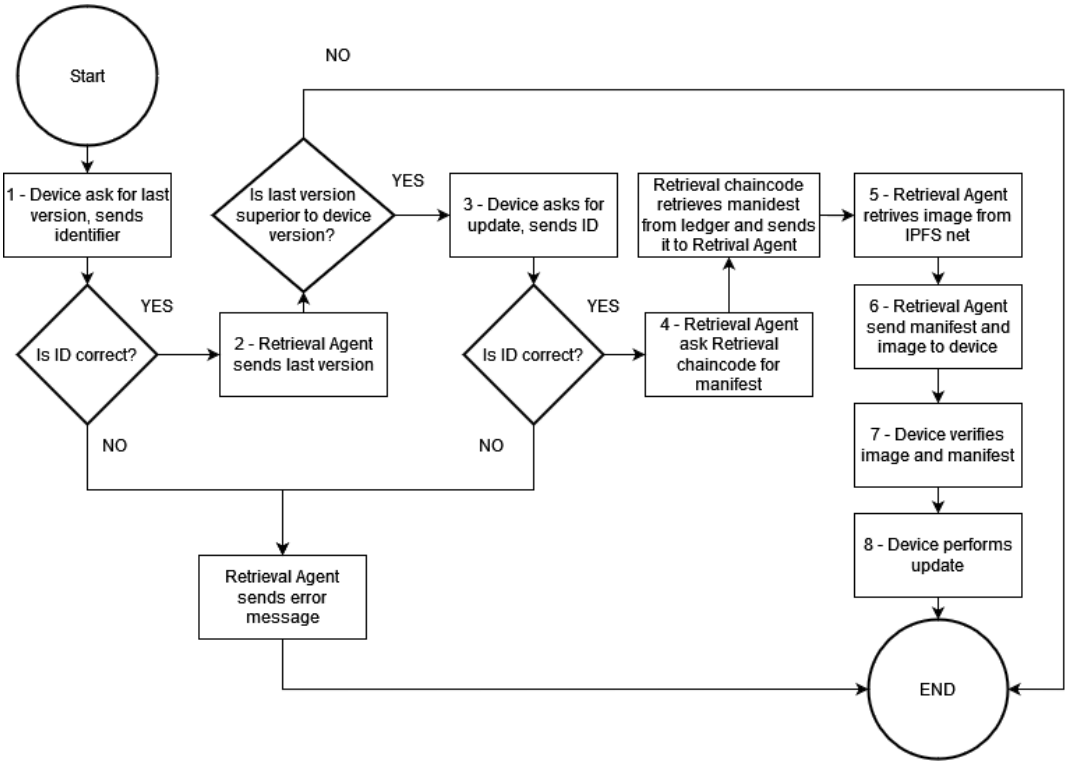

fig 5.6 - Flujo de recuperación de actualizaciones

El flujo de recuperación de actualizaciones, que está reflejado en la figura 5.6, es llevado a cabo cuando un dispositivo quiere comprobar si existe una nueva actualización para la cual es apto. Este proceso sólo tiene sentido una vez se han registrado actualizaciones en la solución. Tiene los siguientes pasos:

1. El dispositivo o su gateway solicitan al agente de Recuperación que le indique la última versión disponible. Para ello envía su identificador, compuesto por la clave pública de su autor y su ClassID, que es el identificador dado por el fabricante para el tipo de actualizaciones que puede aplicar. Si es incorrecto, el agente de recuperación notifica el error y termina el proceso.
2. El agente de recuperación envía la última versión. El dispositivo evalúa si es más reciente que su versión actual. Si no lo es, termina el proceso.
3. El dispositivo o su gateway solicita la actualización. Envía su Identificador.
4. Si el agente de recuperación determina que el ID es correcto, solicita al chaincode de recuperación que obtenga el manifiesto. Este chaincode realiza la query correspondiente a la red y devuelve el manifiesto al agente.
5. El agente de recuperación utiliza el CID para recuperar la imagen de la red IPFS. La verifica utilizando la firma del autor y el digest de la imagen.
6. El agente de recuperación envía la imagen junto a su manifiesto al dispositivo o su gateway.
7. El dispositivo verifica el manifiesto y la imagen.
8. El dispositivo ejecuta la actualización y termina el proceso.

5.3 Diseño de estructuras de datos

Una vez descritos los flujos y la arquitectura de la aplicación, en este apartado se describen las estructuras de datos que los distintos componentes manipulan e intercambian a lo largo de la ejecución de los distintos procesos que forman parte de la actualización.
La estructura más importante a describir es el manifiesto de la actualización. Sin embargo, también se describen las estructuras de las actualizaciones y de los mensajes que los autores y los dispositivos deben enviar para iniciar los procesos de registro y recuperación.

5.3.1 Manifiestos

Los manifiestos son un componente clave del proceso de actualización segura. Sin ellos, los dispositivos que descargan una imagen o payload son incapaces de determinar si la actualización descargada debe ser instalada. El manifiesto permite al dispositivo saber si una actualización es más reciente que la tiene actualmente instalada, si está destinada a su tipo de dispositivo o si se trata de un ataque, entre otras cosas. El formato de esta estructura es importante porque debe garantizar que sus campos otorgan suficiente información al dispositivo para tomar una decisión que no comprometa su seguridad.
El RFC 9124 [22] propone unas pautas para la creación de un formato de manifiestos de firmware para dispositivos IoT. Contiene los siguientes campos:

- ID de versión del manifiesto
- Secuencia de números monotónica: Se trata de una timestamp UTC utilizada para evitar que se reviertan dispositivos a actualizaciones más antiguas (y por lo tanto probablemente menos seguras). Cada actualización debe tener una secuencia más reciente.

- ID del fabricante (Opcional): ID para diferenciar dispositivos con nombres idénticos pero de distintos fabricantes. Se recomienda que siga las pautas de Universally Unique IDentifier (UUID) descritas en el RFC 4122 [23]
- ID de clase: ID utilizada para indicar que clase de dispositivos pueden aplicar la actualización. Una clase puede contener más de un modelo de dispositivo. Se recomienda que siga las pautas de UUID.
- Formato de Payload: Indica en qué formato se encuentra el payload firmado por el autor. El payload es la información que se descarga con el manifiesto, que no necesariamente es la imagen lista para utilizar.
- Procesamiento de Payload (Opcional): Describe los pasos y/o algoritmos necesarios para decodificar el payload y obtener la imagen.
- Localización de almacenamiento: Indica dónde debe localizarse el payload en un componente dado.
- Indicador de Payload (Opcional): Indica dónde obtener el payload en caso de que el componente no sepa intrínsecamente dónde obtenerlo. Puede ser una URI, por ejemplo.
- Digest del Payload: El digest es el resultado de pasar el payload por un algoritmo criptográfico. Puede usarse en conjunto con la clave del autor para verificar el payload.
- Tamaño: tamaño del payload en bytes.
- Firma: La firma del autor no debe ir en el manifiesto sino en su envelope. Junto al digest, permite verificar la integridad del payload.
- Instrucciones Adicionales (Opcional): Instrucciones adicionales necesarias para la instalación de la actualización.
- Dependencias: Lista sin ambigüedad de los otros manifiestos necesarios para aplicar la actualización. Pueden ser digests, por ejemplo.
- Encryption wrapper: Contiene las instrucciones para obtener la clave necesaria para descifrar el payload.
- Payload (opcional): Utilizado para enviar pequeños payload junto al manifiesto, como claves o datos de configuración.

A este manifiesto se le añadirá un nuevo campo adicional que guardará el digest para el manifiesto. Con este campo añadimos la posibilidad de verificar que el manifiesto no ha sido alterado por agentes maliciosos. Este digest es generado antes de añadir este campo, y para obtener su contenido debe eliminarse del manifiesto y utilizar un algoritmo de hash.

5.3.2 Actualizaciones

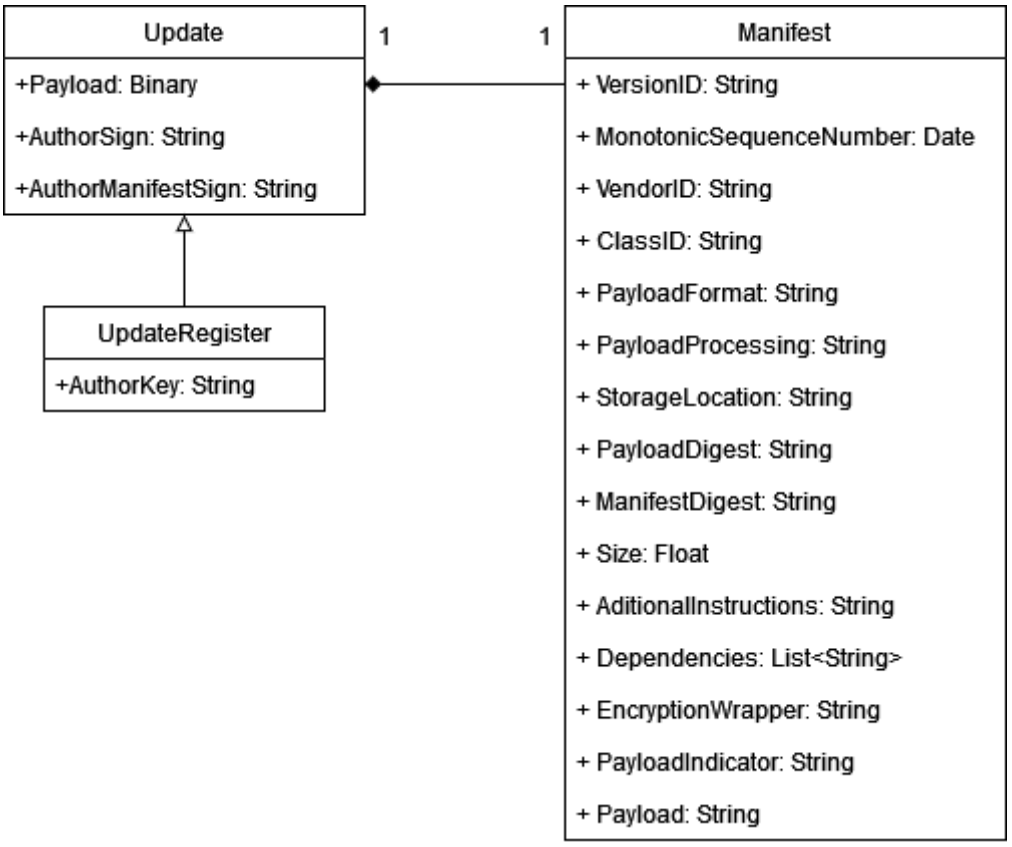

fig 5.7 - Actualizaciones fuera de la red

La estructura de actualización, como se aprecia en la figura 5.7, contiene todos los datos relativos a una actualización concreta. Incluye tanto la propia actualización, compuesta por un manifiesto y un payload, como por las firmas que deben ser utilizadas para comprobar la autenticidad de las actualizaciones.

Cuando se envía inicialmente por el autor, la actualización debe ir acompañada por un token o clave de registro, que es aquella que la solución general para el autor en el flujo de registro de autor.

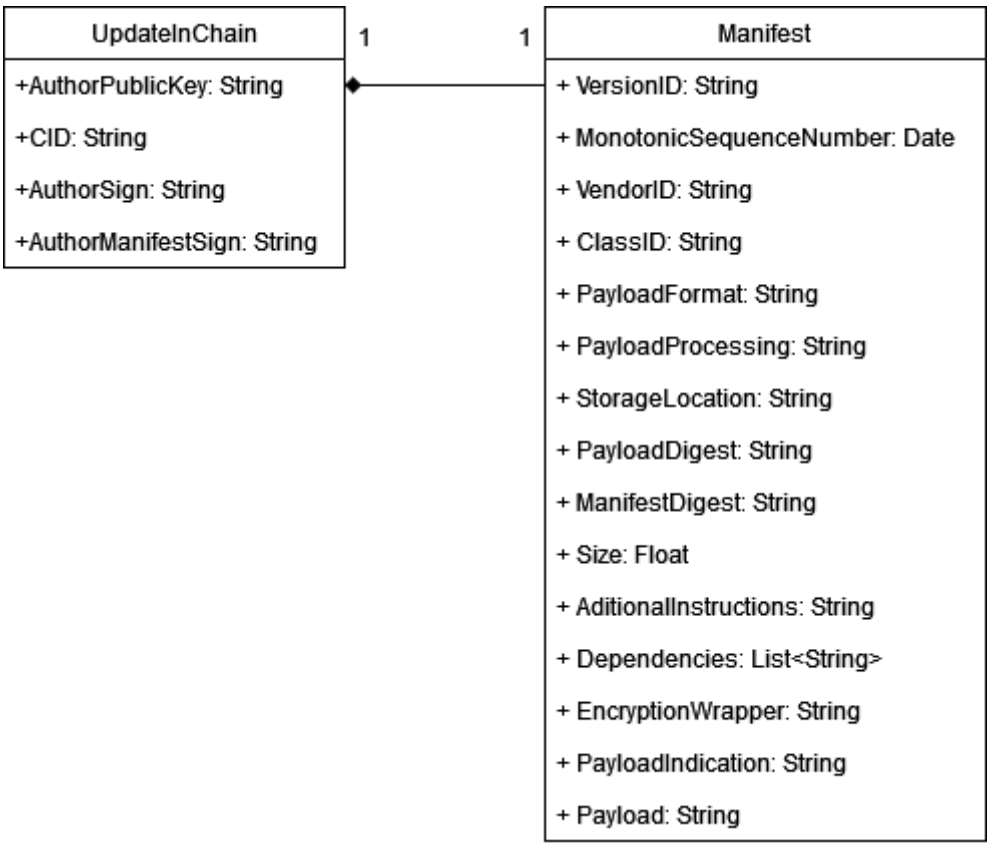

fig 5.8 - Actualizaciones dentro de la red

Cuando son almacenadas en la red, las actualizaciones toman una estructura distinta, la cual queda reflejada en la figura 5.8. En lugar de contener el payload, contienen el CID del mismo, que queda almacenado en IPFS. La clave o token de registro se sustituye por la clave pública del autor.

5.3.3 Identidades de los dispositivos

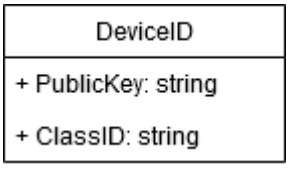

fig 5.9 - Identidad de los dispositivos

Cuando un dispositivo desea obtener una actualización de la solución, debe indicar de alguna manera a la aplicación que actualización desea obtener. Para ello se crea la estructura de identidad del dispositivo, reflejada en la figura 5.9, que está compuesta por la clave pública de su autor y el ClassID del dispositivo, que ha sido asignada por el fabricante y que determina qué tipos de actualizaciones puede recibir. Es necesario que un dispositivo envíe esta estructura cuando realiza peticiones al agente de recuperación.

5.3.4 RegisterPetition

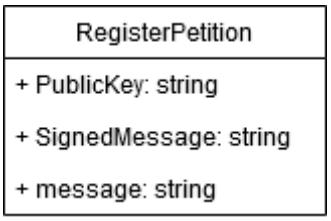

fig 5.10 - Petición de registro

Cuando el autor se quiere registrar en la red, envía su clave y un mensaje. Este mensaje se envía tanto en texto plano como firmado, para que el agente de registro pueda verificar la identidad del autor. La estructura resultante puede verse en la figura 5.10.

5.3.5 KeyPair

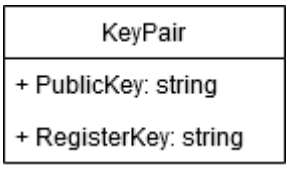

fig 5.11 - Par de claves

Esta estructura relaciona la clave del autor con aquella generada por la blockchain para el registro de actualizaciones. Cuando se almacena por el agente autor, PublicKey es la clave, mientras que en la red la clave será RegisterKey. La estructura resultante puede observarse en la figura 5.11.

5.3.6 UpdatePetition

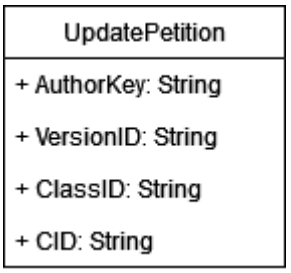

fig 5.12 - Petición de actualización de CID

Esta estructura, apreciable en la figura 5.12, se utiliza cuando el agente de registro quiere enviar el CID al chaincode de registro para actualizar una entrada de actualización. Contiene la clave del autor, la que usa para el registro y no la clave pública, la versionID del manifiesto y la classID del manifiesto, que el chaincode de registro puede utilizar para buscar la actualización a la que debe añadir el CID.

5.4 Diseño de la aplicación cliente

Una vez descritos los flujos y las estructuras necesarias, en este apartado se describe la aplicación cliente, detallando para cada componente como participa en los flujos previamente descritos. Esta

es la parte de la solución que es externa a la red blockchain, pero que permite a los usuarios, ya sean autores o dispositivos, conectarse con la solución e interactuar con la misma.
La aplicación cliente está compuesta por distintas APIs, denominadas agentes, que cumplen distintos propósitos. Se encuentran en un nivel intermedio entre el usuario final de la solución y la aplicación interna de Hyperledger Fabric, que está compuesta por los chaincodes.

5.4.1 Diseño del Agente Autor

El agente autor es el encargado de permitir al usuario autor comunicarse con la red para realizar los procesos de registro, tanto el suyo como el de sus actualizaciones. Este agente debe contar con un almacenamiento donde se guardarán las claves del autor.

El uso de este agente no debe ser público, sino que cada instancia del agente autor debe estar directamente gestionada por un autor. Esto se debe a que en su almacenamiento se guardará la clave de registro del autor, por lo que si la API estuviera accesible para el público, alguien podría utilizarla para realizar peticiones en nombre del autor. La securización de este elemento queda como trabajo futuro, y no forma parte de este TFG.

El agente autor inicia los flujos de registro de autor y actualizaciones a partir de una petición realizada por un autor. Su comportamiento se describe en las siguientes subsecciones.

5.4.1.1 Registro de autor

Esta funcionalidad debe permitir al autor registrarse.

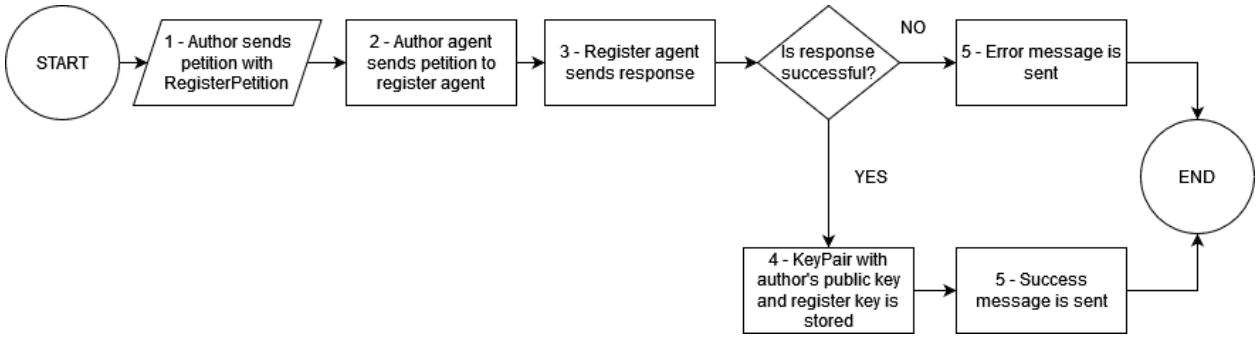

fig 5.13 - Funcionalidad de registro de autor para el agente autor

El agente autor participa en el flujo de registro de autor. Dicha participación queda reflejada en la figura 5.13 y tiene los siguientes pasos:

1. Se recibe la llamada. Contiene una instancia de la estructura de RegisterPetition.
2. El agente envía una llamada POST al agente de registro con la estructura facilitada por el autor.
3. El agente de registro envía una de las siguientes respuestas:
    a. Si se ha completado el registro, envía la clave.
    b. Envía un mensaje con un código de error.

4. Si se ha recibido la clave, se almacena una estructura KeyPair con la clave pública del cuerpo de petición y la clave recibida como respuesta.
5. Se envía un mensaje con el resultado de la operación, que puede contener los siguientes códigos:
    a. 201: Operación realizada con éxito.
    b. 403: Contendrá el mensaje de error enviado por el agente de registro.
    c. 405: Se ha enviado un input no valido.

5.4.1.2 Registro de actualización

Esta funcionalidad debe permitir al autor registrar una actualización.

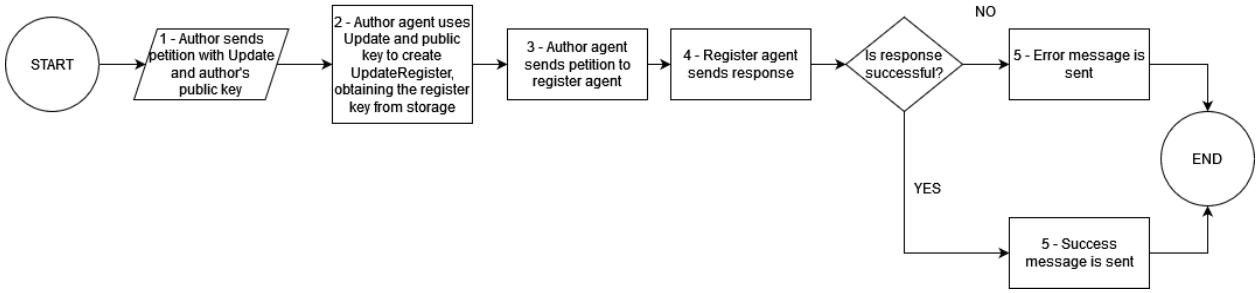

fig 5.14 - Funcionalidad de registro de actualizaciones para el agente autor

El agent autor también participa en el flujo de registro de actualizaciones, como se observa en la figura 5.14. Su participación tiene los siguientes pasos:

1. Se recibe la llamada. Contiene una instancia de la estructura Update y la clave pública del autor.
2. El agente verifica su almacenamiento de claves y crea una estructura UpdateRegister utilizando la clave de registro relacionada a la clave pública del autor.
3. El agente envía una llamada POST al agente de registro con la estructura creada.
4. El agente de registro envía una de las siguientes respuestas:
    a. Envía un mensaje con una notificación de éxito.
    b. Envía un mensaje con un código de error.
5. Se envia una de las siguientes respuestas:
    a. 201: Operación realizada con éxito.
    b. 403: Contendrá el mensaje de error enviado por el agente de registro.
    c. 405: El input enviado no es correcto.

5.4.2 Diseño del Agente Registro

El agente de registro es un intermediario entre los autores y la red blockchain. Envía las solicitudes de registro realizadas por el autor a la aplicación. También es el encargado de subir las imágenes de firmware a la red IPFS. El agente debe ser accesible públicamente.
El agente de registro participa en los flujos de registro de actualizaciones y de autores, continuando a partir de las acciones realizadas por el agente autor. También interactúa con los contratos inteligentes. Su comportamiento se describe en las siguientes subsecciones.

5.4.2.1 Registro de autor

Esta funcionalidad recibe una petición de registro de autor y la redirige al chaincode de registro.

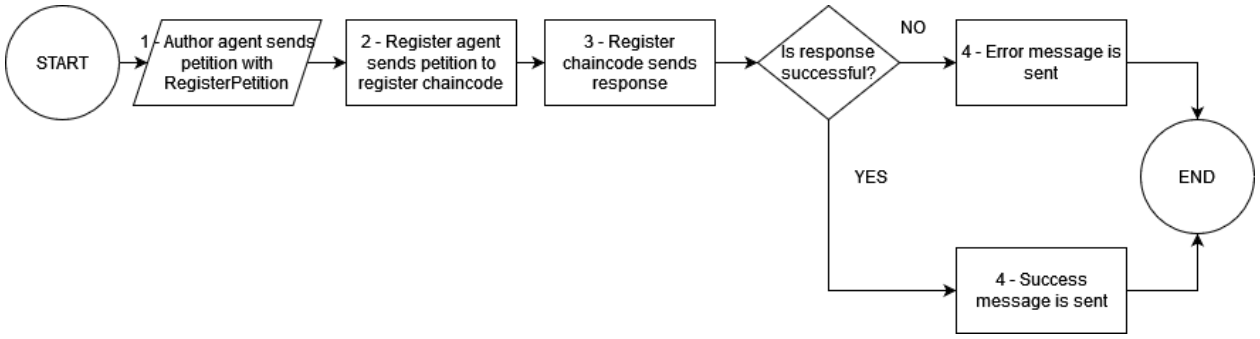

fig 5.15 - Funcionalidad de registro de autor para el agente de registro

El agente de registro continúa con el flujo de registro de autor que inicia el agente autor. Su participación queda reflejada en la figura 5.15 y tiene los siguientes pasos:

1. Se recibe la llamada. Contiene una instancia de la estructura de RegisterPetition.
2. El agente envía la solicitud al chaincode de registro con la estructura facilitada por el agente autor.
3. El chaincode de registro envía una de las siguientes respuestas:
    a. Si se ha completado el registro, envía la clave.
    b. Envía un mensaje con un código de error.
4. Se envía un mensaje con el resultado de la operación, que puede contener los siguientes códigos:
    a. 201: Operación realizada con éxito. Contiene la clave.
    b. 403: Contendrá el mensaje de error enviado por el chaincode de registro.
    c. 405: Se ha enviado un input no valido.

5.4.2.2 Registro de actualización

Esta funcionalidad recibe una petición de registro de actualización. Redirige la petición para que sea comprobada por el chaincode de registro. Si la respuesta del chaincode es exitosa, sube la imagen recibida a IPFS y el CID al chaincode.

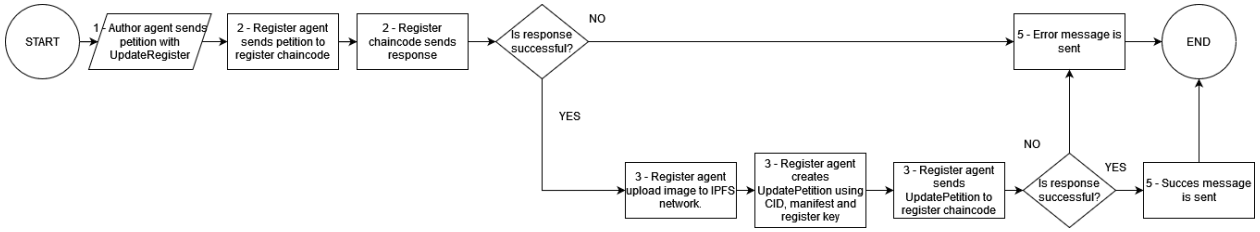

fig 5.16 - Funcionalidad de registro de actualizaciones para el agente de registro

El agente de registro continúa con el flujo de registro de actualizaciones que comienza el agente autor. Su participación se puede observar en la figura 5.16. Tiene los siguientes pasos:

1. Se recibe la llamada. Contiene una instancia de UpdateRegister.
2. Se envía la solicitud con la instancia al chaincode de registro. Puede tener varias respuestas:
    a. Se ha creado la actualización en la red.

b. Mensaje con código de error.

3. Si la respuesta del chaincode ha sido exitosa, guarda la imagen recibida en IPFS. Envía el CID al chaincode de registro. Para ello, construye una instancia de UpdatePetition utilizando el manifiesto de la actualización.
4. El chaincode de registro envía una respuesta:
    a. Código de éxito
    b. Mensaje con código de error
5. Se envía un mensaje con el resultado de la operación:
    a. 201: Operación realizada con éxito. No se envía información sobre la transacción realizada.
    b. 403: Se envía el mensaje de error devuelto por el chaincode.
    c. 405: Se ha enviado un input no valido.

5.4.3 Diseño del Agente de Recuperación

El agente de recuperación es el encargado de recuperar los manifiestos e imágenes de la red para enviarlos a los dispositivos. Debe ser accesible públicamente.
Participa en el flujo de recuperación de actualizaciones, e interactúa con los contratos inteligentes para llevarlo a cabo. Su comportamiento se describe en las siguientes subsecciones.

5.4.3.1 Consulta de versión

Esta funcionalidad permite a un dispositivo conocer la última versión disponible en la blockchain.

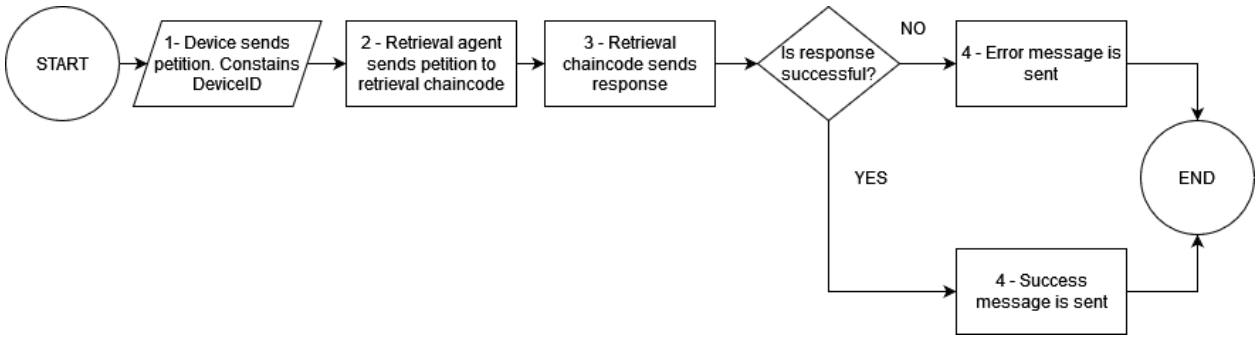

fig 5.17 - Funcionalidad de consulta de versión del agente de recuperación

El agente de registro participa en el flujo de recuperación de actualizaciones. Este comienza con la consulta de versiones, que puede verse en la figura 5.17. Consta de los siguientes pasos:

1. Se recibe la llamada. Contiene una instancia de DeviceID.
2. Se envía la solicitud al chaincode de recuperación con la instancia.
3. El chaincode envía una respuesta:
    a. Envía la última versión
    b. Envia un codigo de error
4. Se envía la respuesta:
    a. 200: Contiene el VersionID de la última versión.
    b. 403: se envía el código de error.
    c. 405: Se ha enviado un input no valido.

5.4.3.2 Recuperación de actualización

Esta funcionalidad permite a un dispositivo obtener la última versión disponible en la blockchain.

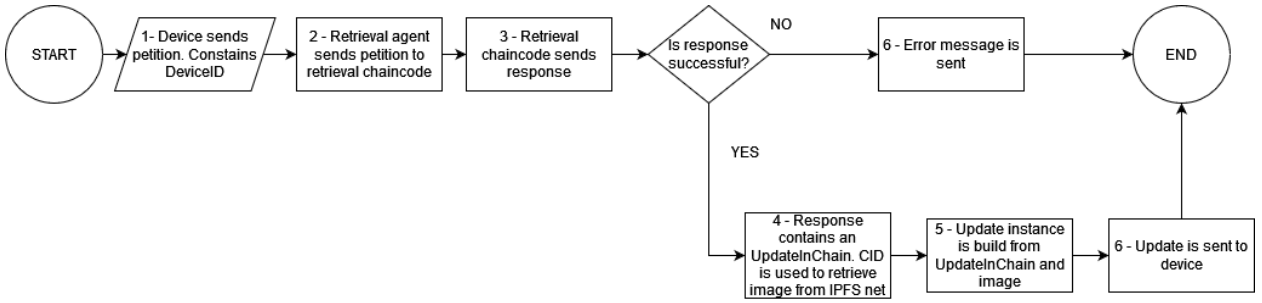

fig 5.18 - Funcionalidad de recuperación de actualizaciones del agente de recuperación

Una vez completado el flujo anterior, el agente de recuperación puede continuar con el flujo de recuperación de actualizaciones, como se muestra en la figura 5.18:

1. Se recibe la llamada. Contiene una instancia de DeviceID.
2. Se envía la solicitud al chaincode de recuperación con la instancia.
3. El chaincode envía una respuesta:
    a. Envía la última versión, una instancia de UpdateInChain.
    b. Envia un codigo de error
4. Si se obtiene una instancia de UpdateInChain, se utiliza el CID para obtener la imagen desde IPFS.
5. Se construye una instancia de Update a partir de la imagen y la instancia de UpdateInChain.
6. Se envía la respuesta:
    a. 200: Se envía la Update.
    b. 403: Se envía el código de error devuelto por la blockchain o se notifica el error al recuperar la imagen.
    c. 405: Se ha enviado un input no valido.

5.5 Diseño de la aplicación blockchain

La aplicación blockchain está compuesta por distintos contratos inteligentes, que se agrupan en una colección llamada chaincode. Se ejecuta en los nodos de la red y sólo es accesible a los usuarios finales a través de la aplicación cliente.
En esta sección se describe como los contratos inteligentes toman parte dentro de los flujos de la aplicación.

5.5.1 Diseño de Contratos de Registro

Los contratos de registro son los encargados de verificar las peticiones de registro de autores y actualizaciones, y de crear las peticiones para que se almacenen en el ledger.

5.5.1.1 Registro de autores

Se verifica la identidad de un autor y se genera una clave de registro para el mismo.

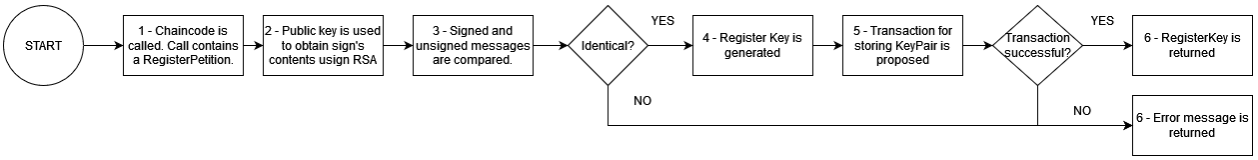

fig 5.19 - Funcionalidad de registro de autor del contrato

El contrato de registro de autores es llamado por el agente de registro durante el flujo de registro de autores. Este contrato se encarga de verificar la identidad de un autor que solicita registrarse, y de generar el token de registro para el mismo. Su comportamiento puede verse en la figura 5.19 y consta de los siguientes pasos:

1. Se recibe la petición del agente de registro. Contiene una instancia de RegisterPetition.
2. Se utiliza la clave pública del autor para ver los contenidos de la firma. Se utilizará el algoritmo RSA.
3. Se verifica que los contenidos de la firma y el mensaje enviado son idénticos. Al solo poder realizarse la firma con la clave privada del autor, confirmamos la identidad del autor que realiza la petición.
4. Se solicita al CA que genere un usuario para el autor / se crea una clave para que sea utilizada por el autor.
5. Se solicita una transacción para crear una instancia de KeyPair con ambas claves.
6. Se envía la respuesta:
    a. 201: Si la transacción se ha completado, se envía la clave de registro.
    b. 403: Se envía un mensaje de error con el código correspondiente.
    c. 405: Se ha enviado un input no valido.

5.5.1.2 Registro de actualización

Se verifican los contenidos de una actualización y se almacena en el ledger.

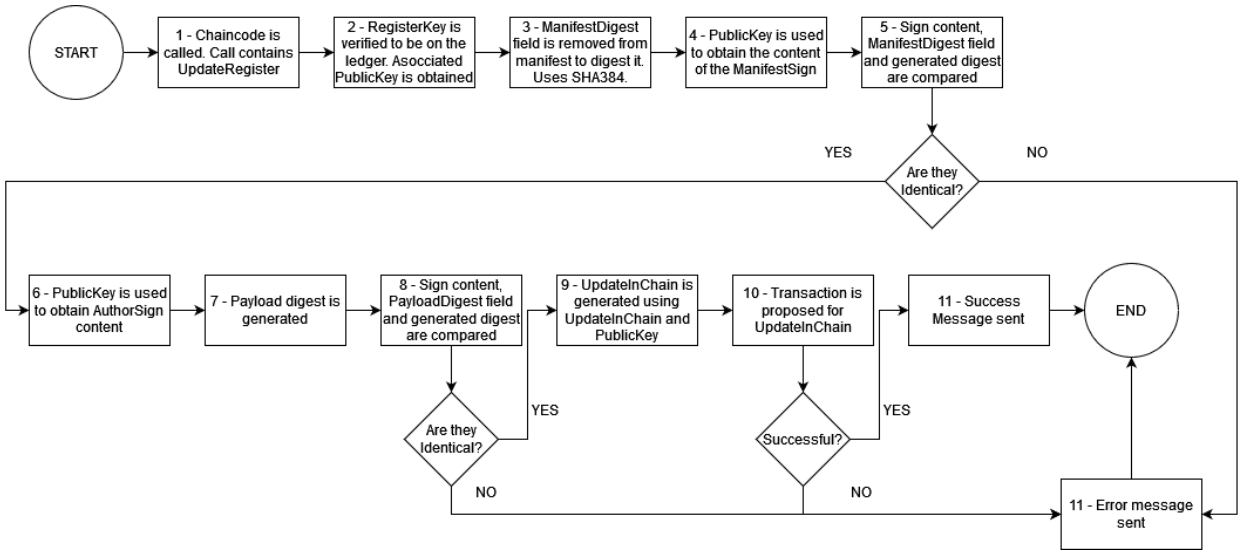

fig 5.20 - Funcionalidad de registro de actualizaciones del contrato

El contrato de registro de actualizaciones forma parte del flujo de registro de actualizaciones y es llamado por el agente de registro. Su función es la de verificar las actualizaciones que intentan ser

registradas en la red, y la de crear las transacciones necesarias para almacenarlas. Su comportamiento puede verse en la figura 5.20. Tienes los siguientes pasos:

1. Se recibe la petición. Contiene una instancia de UpdateRegister.
2. Se verifica que la clave de registro está presente en el ledger, y se obtiene la clave pública asociada.
3. Se retira el campo manifestDigest del manifiesto y se genera el digest, utilizando un algoritmo de hash como SHA 384 (Indicado por el campo PayloadFormat o PayloadProccesing). Se comprueba que el manifiesto tiene todos los campos obligatorios.
4. Se utiliza la clave pública del autor para obtener el contenido de la firma del manifiesto.
5. Se comparan los contenidos de manifestDigest, la firma y el digest generado. Si son idénticos, se verifica que el manifiesto no ha sido alterado desde la firma (Excepto para añadir el campo manifestDigest, que ya hemos tratado).
6. Se utiliza la clave pública del autor para obtener los contenidos de la firma del autor.
7. Se utiliza el algoritmo de hash para generar el digest del payload/imagen.
8. Se comparan el PayloadDigest con el contenido de la firma y el digest generado. Si son idénticos, se verifica la imagen.
9. Se crea una instancia de UpdateInChain a partir de Update.
10. Se solicita la transacción para almacenar UpdateInChain.
11. Se envía la respuesta:
    a. 201: Se ha almacenado la transacción.
    b. 403: Se envía el código de error.
    c. 405: Se ha enviado un input no valido.

5.5.1.3 Actualización de CID

Se actualiza un UpdateInChain para añadir su CID.

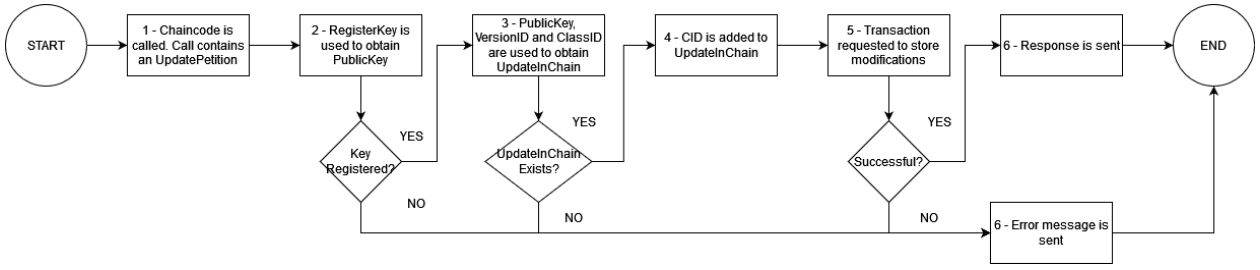

fig 5.21 - Funcionalidad de actualización de CID del contrato

El contrato de actualización de CID forma parte del flujo de registro de actualizaciones, y es llamado por el agente de registro. Su función es la de almacenar el CID generado por IPFS para actualización cuyo manifiesto ya ha sido almacenado en la red blockchain. Su comportamiento queda reflejado en la figura 5.21 y tiene los siguientes pasos:

1. Se recibe la petición. Contiene una instancia de UpdatePetition.
2. Se utiliza la clave de registro para obtener la clave pública a través de una query KeyPair.
3. Se obtiene la actualización del almacenamiento a utilizando la clave pública, la versionID y la ClassID del manifiesto.
4. Se almacena el CID.
5. Se solicita una transacción para guardar la Actualización modificada.
6. Se envía la respuesta:
    a. 201: Se ha actualizado la transacción

b. 403: Se envía el código de error.
c. 404: La actualización no existe en el ledger.
d. 405: Se ha enviado un input no valido.

5.5.2 Diseño de contratos de recuperación

Los contratos de recuperación se encargan de realizar las queries al ledger para recuperar las actualizaciones de la red.

5.5.2.1 Consulta de versión

Se obtiene la versión de la última actualización.

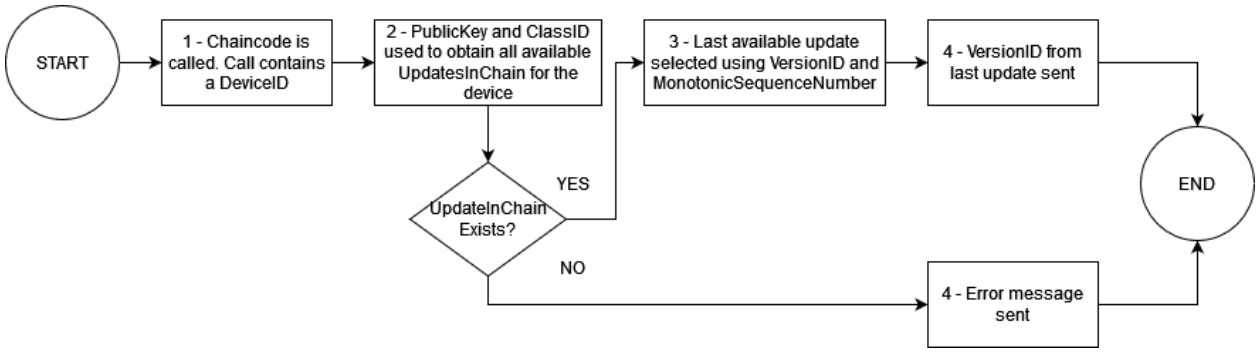

fig 5.22 - Funcionalidad de recuperación de versiones del contrato

El contrato de recuperación de versiones forma parte del flujo de recuperación de actualizaciones y es llamado por el agente de recuperación. Su función es la de recuperar únicamente la versión de la actualización más reciente disponible para un dispositivo. Su comportamiento se puede observar en la figura 5.23 y tiene los siguientes pasos:

1. Se recibe la petición. Contiene un DeviceID.
2. Se utilizan la clave pública y el ClassID para obtener todas las actualizaciones disponibles para el dispositivo.
3. Se utiliza el campo VersionID o MonotonicSequenceNumber (dependiendo del formato de VersionID) para determinar la última versión disponible.
4. Se envía la respuesta:
    a. 200: Contiene la VersionID.
    b. 403: Se envía el error.
    c. 405: Se ha enviado un input no valido.

5.5.2.2 Recuperación de actualización

Se envía la actualización más reciente para un dispositivo.

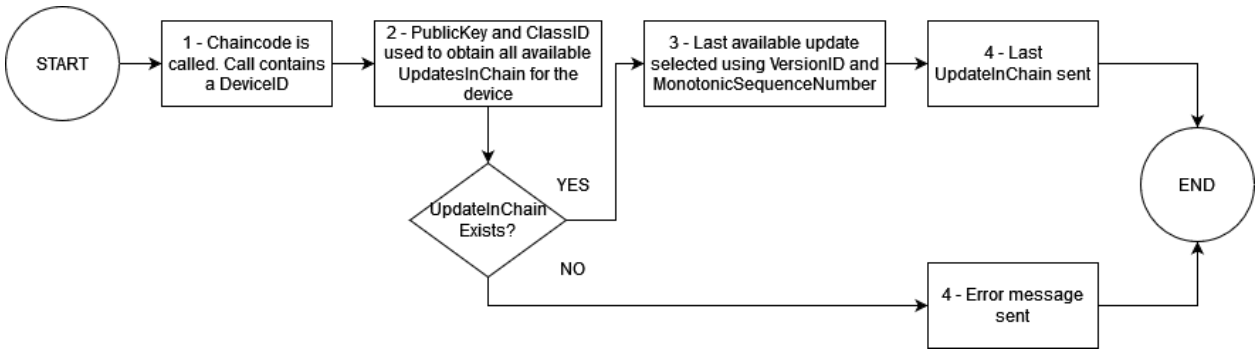

fig 5.23 - Funcionalidad de recuperación de actualizaciones del contrato

Este contrato es llamado por el agente de recuperación como parte del flujo de recuperación de actualizaciones. Queda reflejado en la figura 5.23 y tiene los siguientes pasos:

1. Se recibe la petición. Contiene un DeviceID.
2. Se utilizan la clave pública y el ClassID para obtener todas las actualizaciones disponibles para el dispositivo.
3. Se utiliza el campo VersionID o MonotonicSequenceNumber (dependiendo del formato de VersionID) para determinar la última versión disponible.
4. Se envía la respuesta:
    a. 200: Contiene la UpdateInChain.
    b. 403: Se envía el error.
    c. 405: Se ha enviado un input no valido.

5.6 Diseño del emulador de dispositivos

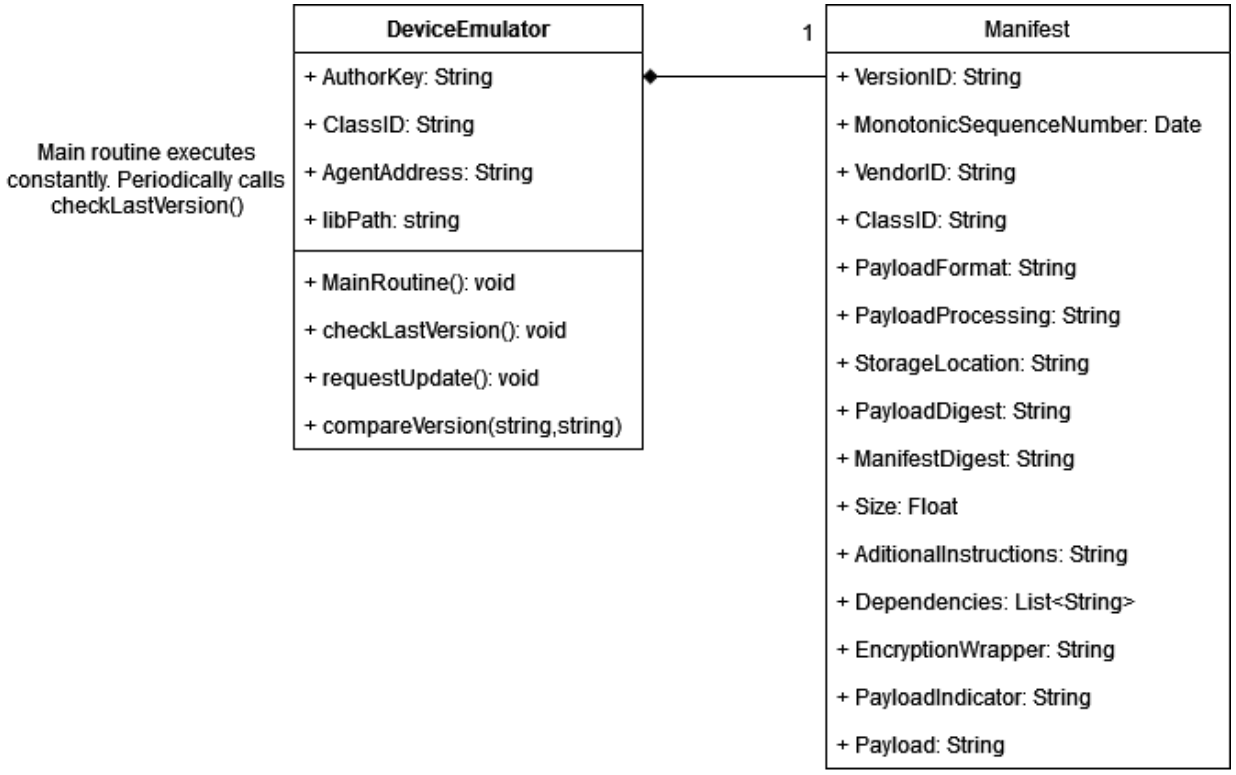

fig 5.24 - Estructura del emulador de dispositivos

El emulador de dispositivos es un componente que debe permitir simular el comportamiento que tendría un dispositivo a la hora de interactuar con el resto de la solución. Este componente es útil para probar el flujo de recuperación de actualizaciones, ya que no existe un dispositivo cuyo firmware soporte el uso de esta solución, y actualizar uno para realizar una prueba con un dispositivo real requeriría tiempo y conocimientos que alargarían el alcance de este TFG por encima de los límites de tiempo disponibles para su elaboración.
Debido a las restricciones del proyecto, se ha decidido que la mejor forma para llevar a cabo el emulador es con un script contenido en una máquina virtual o en una imagen docker. En futuras iteraciones del proyecto puede añadirse la capacidad para ejecutar firmware real, pero queda fuera del alcance de este TFG.

Este dispositivo debe contar con la siguiente información:

- Su ClassID
- La clave pública de su autor
- El manifiesto de la versión actual de su firmware.
- La dirección del agente de recuperación.
- Un método para comparar VersionIDs.
- Un método que solicite la última versión periódicamente.
- Librerías criptográficas para ejecutar la verificación de la actualización.

Dicha información se almacenará en el contenedor o máquina virtual, pudiendo añadir una base de datos para hacer el guardado de información más sencillo.

5.7.1 Solicitud de versión

El dispositivo debe poder solicitar la última versión al agente de recuperación.

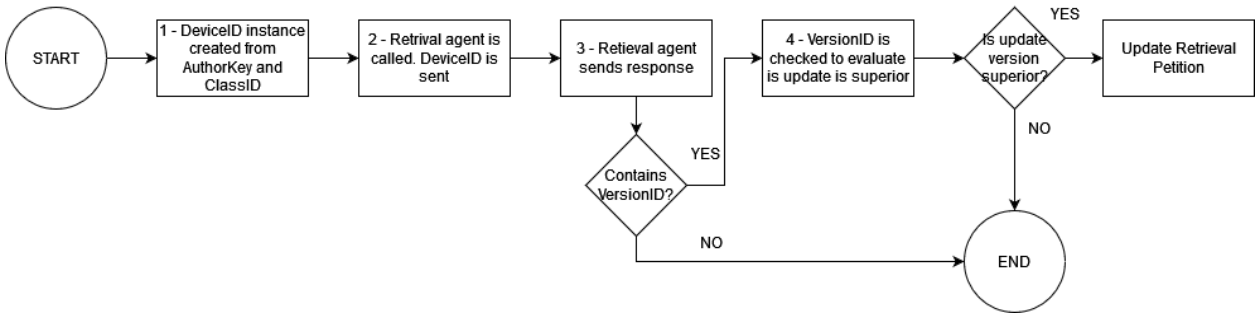

fig 5.25 - Funcionalidad de recuperación de versión del emulador de dispositivos

Este proceso, que se puede observar en la figura 5.25, iniciara el flujo de recuperación de actualizaciones. Consta de los siguientes pasos:

1. Se crea una instancia de DeviceID con la ClassID y clave pública del dispositivo.
2. Se realiza una llamada al agente de recuperación. Se envía el DeviceID.
3. El agente envía la respuesta con la última versión.
4. El dispositivo obtiene la VersionID y compara si es más reciente que su versión. Si lo es, inicia la petición de actualización.

5.7.2 Petición de actualización

El dispositivo ya ha determinado que la última versión presente en la blockchain es más reciente que está ejecutando, por lo que solicita la imagen y la instala.

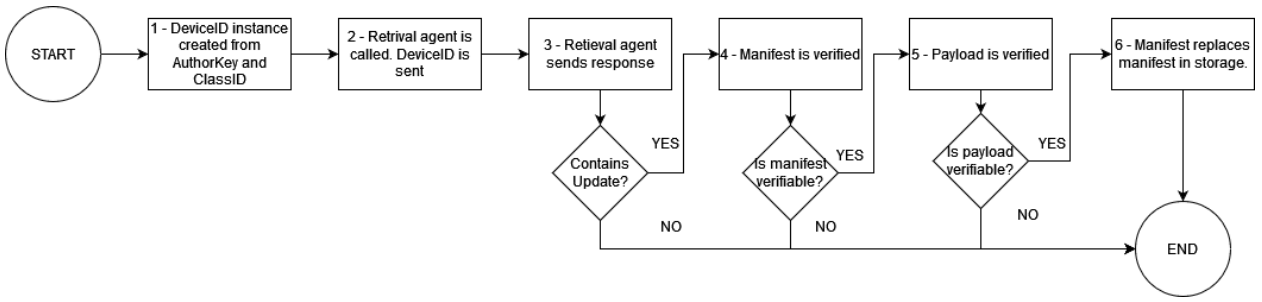

fig 5.26 - Funcionalidad de recuperación de actualización de emulador de dispositivos

Una vez el dispositivo requiere la nueva actualización, podría solicitarla al agente de recuperación, continuando el flujo de recuperación de actualizaciones. Este proceso queda reflejado en la figura 5.26 y tiene los siguientes pasos:

1. Se crea una instancia de DeviceID con la ClassID y clave pública del dispositivo.
2. Se realiza una llamada al agente de recuperación. Se envía el DeviceID.
3. El agente envía la respuesta con la última actualización.
4. El dispositivo utiliza la clave pública del autor para obtener el contenido de la firma y para generar el digest del manifiesto, tras eliminar el campo manifestDigest. Si el resultado de este proceso es idéntico al contenido del campo, se verifica el manifiesto.
5. El dispositivo utiliza la clave pública del autor para obtener el contenido de la firma y para generar el digest del payload/imagen. Si el resultado de este proceso es idéntico al contenido del campo PayloadDigest, se verifica el payload/imagen.
6. Si tanto la imagen como el manifiesto son verificados, sustituye su manifiesto por el obtenido.

En este punto debería reiniciarse el dispositivo y ejecutar la nueva imagen, pero no es necesario para este emulador ya que lo que nos interesa en este TFG es crear un proceso seguro de intercambio de actualizaciones más que la prueba de una imagen firmware concreta.

6 Implementación

En este apartado se describe la implementación del diseño realizado, incluyendo la segunda y la tercera fase de este TFG, además de una pequeña cuarta fase adicional. Debido a que el proyecto contiene una cantidad significativa de elementos, no se va a describir a fondo el código realizado, centrándose en su lugar en dar una explicación sobre los componentes más relevantes y las dificultades superadas durante la implementación.

En primer lugar se describen brevemente la arquitectura y los elementos implementados. Estos elementos son los agentes, los contratos inteligentes, las apps (pequeñas aplicaciones auxiliares) y las redes Hyperledger Fabric e IPFS. Después se describen los flujos implementados. Todos estos elementos se implementan basándose en el diseño presentado en el capítulo anterior. Los cambios realizados respecto al diseño original también quedan reflejados en este capítulo.
El código y su documentación están disponibles en el repositorio del proyecto [0].

6.1 Arquitectura y componentes de la solución

En este apartado se describen los componentes implementados y se muestra la arquitectura final de la solución. También se describe cómo han variado con respecto al diseño realizado en la fase 1 del proyecto.

6.1.1 Arquitectura

Tal y como se describe en el apartado 5.1, se ha implementado una solución basada en Hyperledger Fabric e IPFS. Ambas tecnologías han sido desplegadas a través de contenedores Docker utilizando imágenes distribuidas oficialmente. La arquitectura resultante de esta implementación puede apreciarse en la figura 6.1.

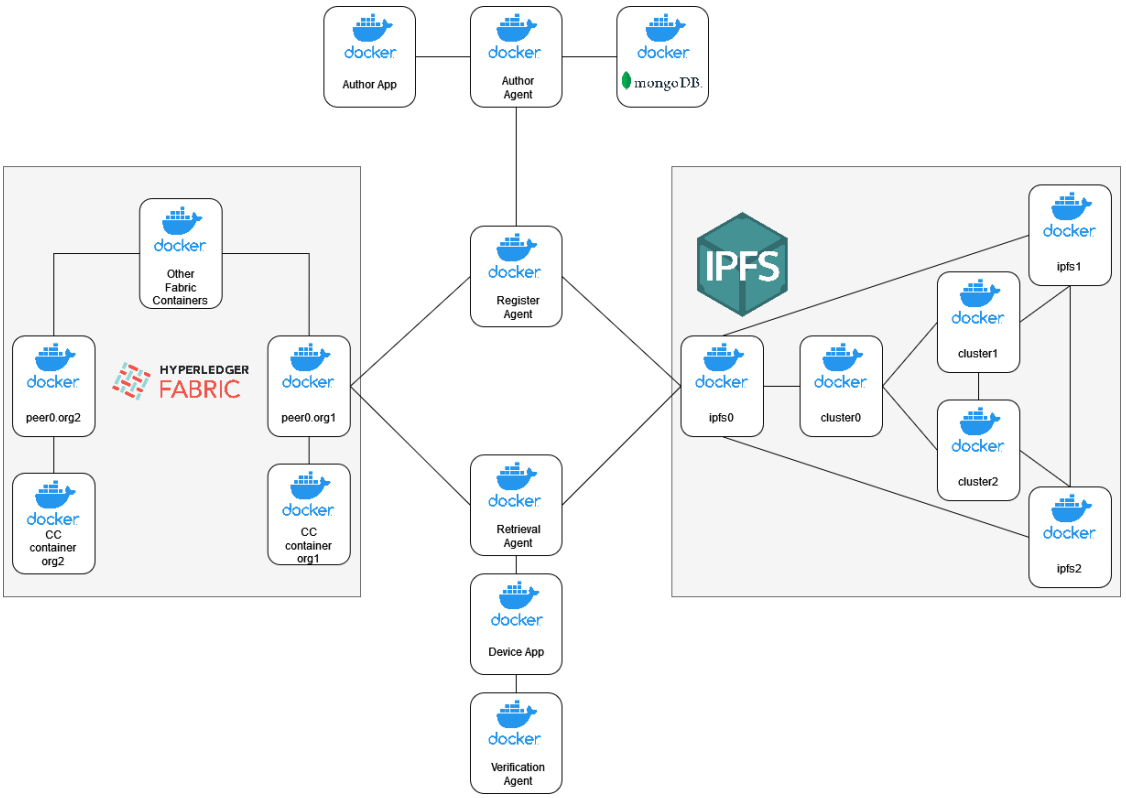

fig 6.1 Arquitectura de la solución implementada

El principal desajuste con respecto al diseño original es la inclusión de las "Apps", aplicaciones frontend que interactúan con los agentes. Su inclusión fue motivada por la necesidad de presentar la usabilidad de la aplicación, ya que las APIs, aunque funcionales, no tienen un comportamiento fácilmente demostrable de manera visual. Adicionalmente, también se ha añadido un agente que apoya a los dispositivos realizando la verificación de las actualizaciones para ellos. Este desajuste ocurre en la cuarta fase del proyecto, que tan solo tuvo una duración de alrededor de 20 horas.

Para que el proceso diseñado pueda ser implementado de manera completa es necesario que exista una conexión entre los agentes de registro y recuperación y una red blockchain, la que a su vez contendrá el contenedor que ejecuta el chaincode de registro. Se ha utilizado la red de prueba de Hyperledger Fabric. Esta red no representa una red de producción y presenta algunas simplificaciones que pueden resultar en fallos de seguridad en un entorno real, pero nos permite comprobar el funcionamiento de los contratos implementados. Aunque no forma parte del alcance del TFG, el proyecto deberá desplegar una red personalizada para poder ser utilizado de manera segura en un entorno de producción.

Por otro lado, se ha utilizado una red privada de IPFS compuesta por tres nodos. Estos tres nodos a su vez forman un cluster para facilitar el intercambio de información entre ellos. Es importante mencionar que el agente de registro fija las imágenes subidas, dado que de lo contrario IPFS funciona como una gran memoria caché y puede eliminar archivos no utilizados comúnmente para liberar espacio.

Aunque se ejecute todo dentro de la misma máquina, cada componente está contenido utilizando Docker, lo que nos permite emular una red en la que los distintos nodos y componentes se ejecutan independientemente. Para conectarlos todos se utiliza la función de red de Docker, incluyendo todos los contenedores una misma red, concretamente aquella creada por la red de pruebas de Fabric (fabric_test).
El proceso de instalación de esta red resulta bastante complejo, por lo que se han escrito algunos scripts con el objetivo de automatizarlo. Sin embargo, hay ciertos pasos que requieren de interacción manual. Una guía detallada del proceso está disponible en el anexo B, así como en la documentación del repositorio.

6.1.2 Agentes

Los agentes, como se describe en el diseño, son aplicaciones web que sirven APIs. Contamos con cuatro agentes en el proyecto: Autor, Registro, Recuperación y Verificación. Los dos primeros atienden al proceso de registro de actualizaciones, mientras que los dos segundos, al de recuperación.

Para servir las APIs y gestionar las rutas de las mismas se ha utilizado la librería Express. Esta es una librería muy popular, ya que permite realizar todas las funcionalidades que ofrece NodeJS de una manera más rápida y eficiente para el programador. Cuenta con una funcionalidad de router, que indica a la aplicación que rutas escuchar y cómo atender a las peticiones realizadas a las mismas.

Cada vez que se recibe una petición, la aplicación ejecuta una serie de middleware, programas intermediarios que tienen acceso a la petición y puede procesar sus contenidos. Es destacable el uso de las librerías body-parser y multer, que procesan el cuerpo de la petición y lo convierten en objetos utilizables. Un ejemplo de la atención a una petición puede verse en la figura 6.2.

```
//Receives a RegisterPetition object. If succesful stores a keyPair with a new register key fro
//the blockchain and the given public key.
router.post("/register/author", async(req, res) => {
  try{
    const response = await registerAuthor(req);
    res.status(parseInt(response.status)).json(response.message);
  }catch(error)  {
    console.log(error);
    if (!error.stat) error.stat = 500;
    res.status(parseInt(error.stat)).json(error.message);
  }
});
```

fig 6.2 - Atención a ruta de registro de autores

Una vez realizado el procesado inicial, la aplicación delega en un controlador. Cada uno de estos controladores funciona como un gestor que a su vez delega el procesamiento de tareas en servicios más específicos, intentando adherirse al principio de responsabilidad única.

Durante estos procesos, el servidor intenta en todo momento controlar la información para devolver el error más adecuado posible para cada situación, por lo que se ha implementado una serie de capturas de y gestión de errores, de manera que la aplicación pueda enviar una respuesta aunque ocurra un error inesperado. De esta manera, la ejecución no se detiene y puede seguir atendiendo peticiones, lo que podría resultar importante en la solución en un futuro, puesto que un entorno de actualización segura podría tener un requerimiento de disponibilidad muy elevado, siendo que tiene que tendría atender a múltiples dispositivos de la manera más veloz posible para evitar que sean atacados con las vulnerabilidades que pueden ser corregidas con una actualización.

Estos agentes son contenidos en contenedores Docker para facilitar su posterior despliegue. Por ello, contienen un archivo Dockerfile y un archivo Docker-Compose. Adicionalmente se ha añadido un script shell para cada agente con el objetivo de automatizar el proceso de inicialización de los contenedores.

Dependiendo de la ruta, los agentes pueden escuchar y atender peticiones con una composición distinta. Todas las rutas atienden llamadas con cuerpos que contienen objetos JSON de manera adecuada. En el caso de que el payload de una actualización sea de tamaño considerable, la petición de registro de actualización también puede realizarse en formato multipart/form-data. Estas peticiones deben contener los campos definidos en las estructuras de datos.

En algunos agentes existen varios servicios que no son utilizados durante la ejecución principal, pero que sirven como apoyo para realizar testeos. Por ejemplo, el agente autor tiene servicios para generar pares de claves RSA y el agente de registro tiene un servicio para ejecutar queries a la blockchain de manera manual.

Cabe destacar en los agentes de registro y verificación la inclusión de un par de métodos ,*enrollAdmin* y *registerUser*, que interactúan con la CA de Hyperledger Fabric. Esta CA es un servicio interno de Fabric que gestiona su PKI y otorga los certificados necesarios a una aplicación para comunicarse con la blockchain. Como resultado, obtienen los certificados necesarios para interactuar con la aplicación blockchain, los cuales deben incluir en cada una de sus llamadas. Los certificados siguen en estándar X.509 y se almacenan localmente en una carpeta llamada wallet.

Estos dos agentes también interactúan con IPFS gracias a la librería cliente HTTP.

Los agentes solo atienden a las peticiones que reciben por un puerto específico. Dentro de un mismo puerto, además, solo atienden a unas direcciones específicas que tienen el nombre de ruta. Una petición a un agente podría dirigirse, por lo tanto, a una dirección similar a ‘http://localhost:3000/ruta’. Cada agente atiende a unas rutas diferentes, que son descritas a continuación.

El agente Autor atiende a tres rutas:

- /sign permite a un usuario de la app cliente firmar contenidos. Requiere del envío de una clave privada. Es recomendable firmar contenidos de manera local, pero ofrece la posibilidad a un autor que no pueda hacerlo de manera local. Puede firmar tanto payloads como manifiestos sin tener que realizar una llamada diferenciada. Devuelve tanto el digest como la firma del contenido enviado.

- /register/author permite realizar el flujo de registro del autor.
- /register permite realizar el flujo de registro de las actualizaciones

El agente de Registro atiende dos rutas:

- /register/author continua con el flujo de registro de autores.
- /register continua con el flujo de registro de actualizaciones.

El agente de recuperación atiende dos rutas:

- /retrieve/version permite realizar el flujo de recuperación de versiones.
- /retrieve permite realizar el flujo de recuperación de actualizaciones.

El agente de verificación atiende tan solo una ruta:

- /verify permite a un dispositivo enviar una Update y su DeviceID para que el servidor ejecute el proceso de verificación.

Finalmente, los agentes atienden a la ruta /api-docs, que permite acceder a la documentación Swagger. Esta documentación además es un pequeño frontend que permite interactuar con los agentes y que resulta útil para testeos manuales. En la figura 6.3 puede observarse un ejemplo de una documentación Swagger.

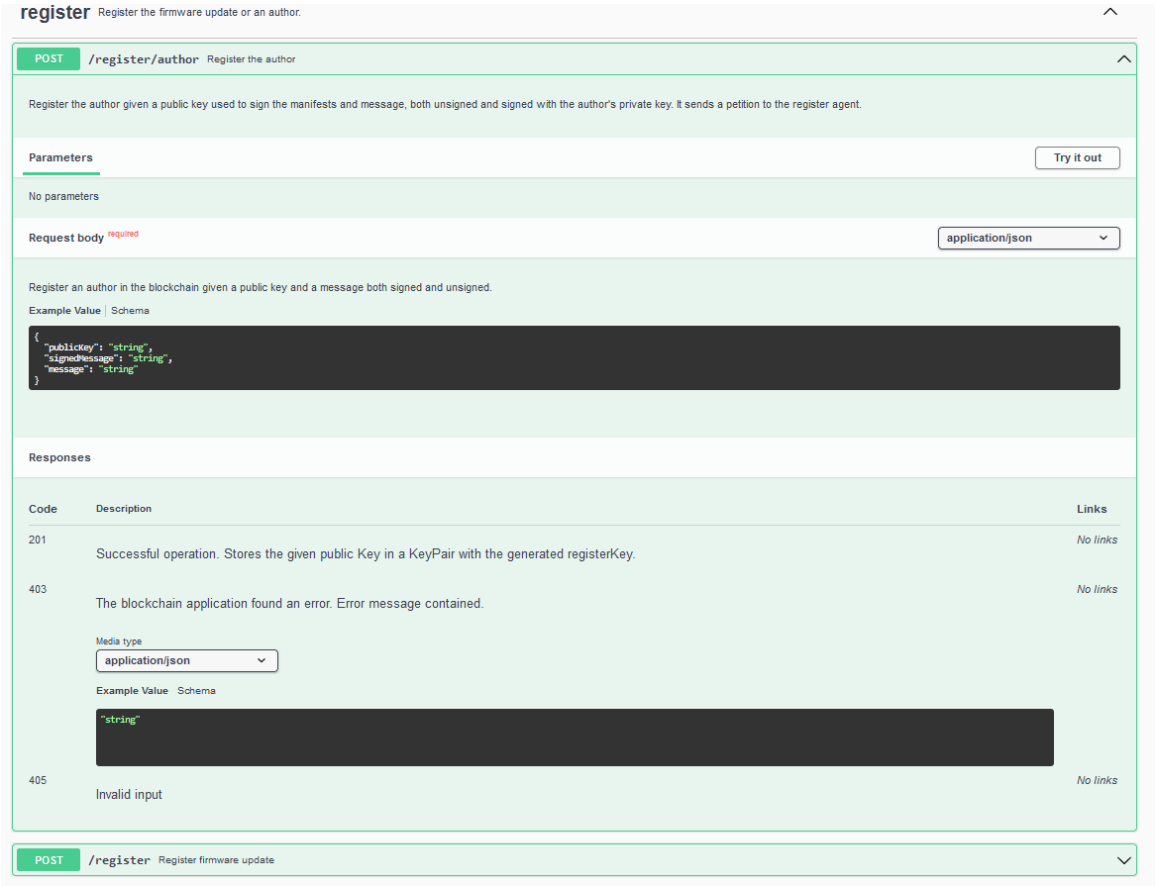

fig 6.3 Documentación Swagger para /register/author

En los agentes de registro y recuperación se tomó la decisión no incluir una base de datos, ya que no se consideró finalmente necesario, al menos en esta fase del proyecto. Esto es debido a que en las ejecuciones de prueba se sigue un comportamiento secuencial, por lo que no se ejecutan contratos inteligentes de manera simultánea. Para que la solución resultase eficaz en un entorno de producción, sin embargo, sería necesario implementar una pila de peticiones con una base de datos como Redis, ya que la ejecución simultánea de dos contratos puede llevar a errores como la escritura fantasma, en la que un fichero almacenado se modifica mientas otra ejecución está realizando una búsqueda sobre el mismo. La implementación de dicha pila requeriría de un tiempo de desarrollo adicional que no resultaba posible obtener durante la ejecución del proyecto.

6.1.3 Chaincode

El Chaincode es una colección de contratos inteligentes. Al contrario que los agentes no sirve una API de manera explícita, sino que es llamado por el agente de registro utilizando la librería cliente de Hyperledger Fabric. Contiene una estructura mucho más sencilla, donde cada uno de los contratos contiene en su interior todas las funciones que interactúan directamente con el ledger. Aquellas funciones auxiliares que no acceden directamente a la información de la blockchain han sido aisladas como servicios. Un ejemplo de contrato inteligente puede verse en la figura 6.4.

```
//Register an author. If the author already exists, overwrites the registerKey.
async createAuthor(ctx, message, signedMessage, publicKey, date) {
    try {
        console.info('============= START : Create Author ===========');
        //Verify public key;
        console.info('============= Verifiying public Key =============');
        const verifiable = verifySign(message,signedMessage,publicKey);
        if (!verifiable){
            throw new Error('ERR_KEY_NOT_VERIFIABLE');
        }
        //create author register key
        console.info('============= Creating Register Key ===========');
        const registerKey = createRegisterKey(publicKey);
        //Store keyPair.
        const author = {
            docType : 'author',
            publicKey : publicKey,
            registerKey: registerKey
        }
        console.info('============= Storying keypair ===========');
        await ctx.stub.putState('Author'+date.toString(), Buffer.from(JSON.stringify(author)));
        console.info('============= END : Create Author ===========');
        return registerKey;
    } catch (err) {
        throw err;
    }
}
```

fig 6.4 - Función de registro del contrato de registro de autores.

El chaincode no es contenido en Docker de manera directa, ya que será la red de Hyperledger Fabric quien se encargará del proceso cuando se instala el chaincode en la misma.

Dispone de tres contratos inteligentes, uno para cada proceso: author, update y retrieve-update. El contrato de autor contiene funciones para registrar claves de registro y para buscar sus claves públicas asociadas. El contrato de update contiene funciones para introducir una nueva actualización, para identificar si una actualización ya existe previamente y para añadir el CID a una actualización ya existente. Finalmente, el contrato de recuperación contiene funciones para obtener la versión más reciente para un aparato dado su DeviceID, y para obtener la actualización relacionada a dicha versión.
Este chaincode también está implementado utilizando NodeJS, utilizando para ello las librerías del SDK. Aunque era la elección más coherente con el resto del proyecto, que también está desarrollado en NodeJS, resultó ser una decisión que conllevó ciertos problemas, como se describe en el apartado 6.3.

6.1.4 Apps

Las apps de la solución son un pequeño añadido realizado al final de la implementación. Su propósito es facilitar a un usuario interactuar con el resto de los componentes implementados de manera visual. Aunque son útiles para el testeo de la aplicación, no suponen un producto final.

Se ha implementado una app que funciona como un frontend para el autor, permitiendo registrar autores y actualizaciones. Permite la opción de cargar archivos, necesario para las claves pública y privada, ya que si se suben como texto (con la herramienta copiar y pegar del sistema operativo, por ejemplo) tienden a corromperse.

También se ha implementado una app que simula el comportamiento esperado de un dispositivo, permitiendo solicitar una actualización y pudiendo verificarla a través del agente de verificación. Aunque no tiene un verdadero comportamiento de dispositivo y no ejecuta realmente el firmware, es suficiente para demostrar el funcionamiento del flujo. Se ha elegido realizar esta app en lugar de simular un dispositivo de manera completa debido a que esta segunda opción excedería con creces el tiempo originalmente planificado para el TFG.

Estas apps han sido desarrolladas en NodeJs con el uso del framework React, que permite crear aplicaciones frontend rápidamente. Al ser un framework popular existe mucha documentación sobre el mismo, lo que sumado a la experiencia adquirida en el uso de NodeJS con los componentes anteriores permitió terminar ambas apps en un periodo de tiempo muy corto, sin añadir una carga de trabajo realmente significativa a la planificada originalmente. Su uso es de carácter demostrativo y opcional. El aspecto de la GUI para la app de autor puede verse en la figura 6.5.

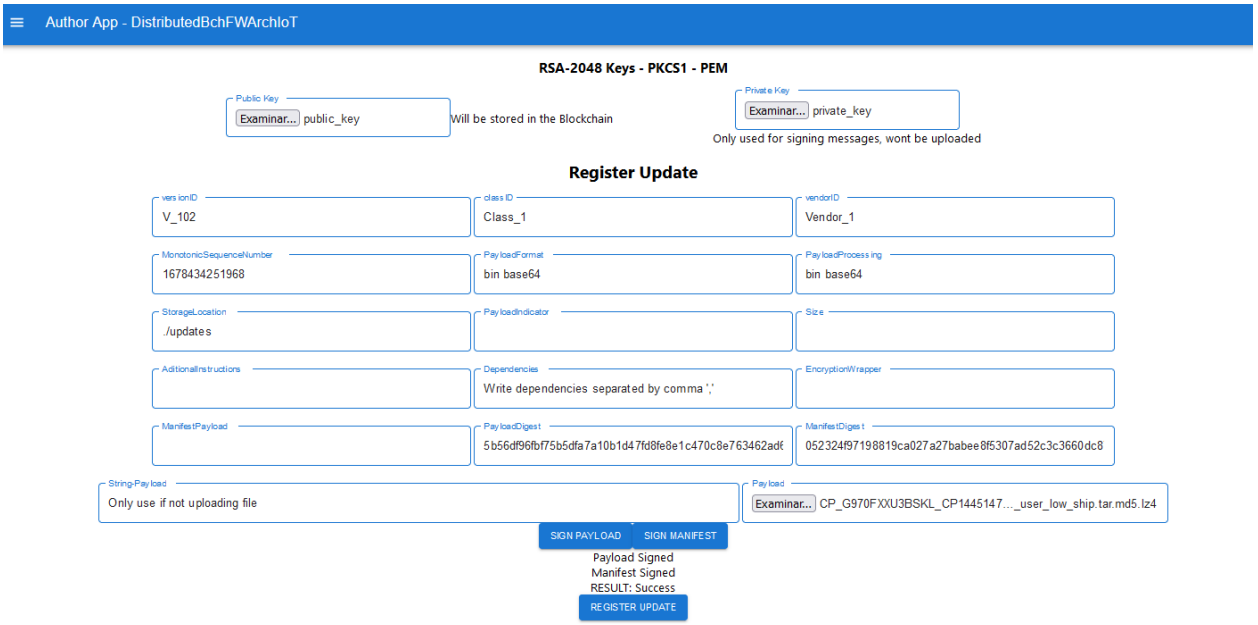

fig 6.5 - Captura de la app de autor

6.2 Flujos de la solución

En este apartado se describe cómo se han implementado los flujos diseñados. Aunque son fundamentalmente muy similares a los establecidos en el capítulo de diseño, se han introducido pequeños cambios y mejoras a la hora de realizar la implementación.

6.2.1 Registro de Autores

Para este flujo tanto el agente de autor como el de registro utilizan la ruta /register/author.

1. El flujo comienza con el agente autor, que verifica que la petición tiene un cuerpo que contiene los elementos requeridos, es decir, un objeto equivalente a la estructura RegisterPetition.
2. El agente autor llama a un servicio que realiza la petición POST al agente de registro.
3. El agente de registro verifica nuevamente los contenidos de la petición. Esta verificación se realiza nuevamente para evitar que peticiones ajenas al agente autor puedan causar daños o peticiones incorrectas que añadan carga innecesaria a la blockchain.
4. El agente de registro llama al servicio que gestiona la petición al chaincode (esta comunicación se realiza mediante grpc, pero es gestionada por la librería de Fabric).
5. El chaincode de registro verifica nuevamente los contenidos de la petición.
6. El chaincode verifica la clave pública enviada comparando los contenidos del mensaje con aquellos de descifrar el mensaje firmado. Se realiza mediante la librería crypto incluida por defecto en NodeJS. Se utiliza el algoritmo RSA.
7. El chaincode genera una clave de registro para asociar a la clave pública. Realizamos este proceso mediante un JSON Web Token (JWT) con expiración. Para ello, utilizamos la librería json-web-token disponible en npm.
8. El chaincode guarda el par de claves como un asset en el ledger.
9. El chaincode devuelve al agente de registro la clave de registro generada.

10. El agente de registro devuelve al agente autor la clave recibida.
11. El agente autor guarda en su base de datos mongodb el par de claves.
12. El agente autor devuelve un mensaje de éxito y muestra la clave de registro recibida, ya que el autor la puede necesitar si utiliza otra instancia del agente autor o quiere realizar una petición directamente al agente de registro.

Para la implementación de este proceso se tomó la decisión de utilizar el algoritmo RSA-2048. Las claves producidas por este algoritmo suponen el estándar de seguridad mínimo en el momento de la realización de este documento. Aunque la generación de estas claves no es responsabilidad de la solución, para poder realizar pruebas es necesario contar con al menos un par de claves, por lo que se ha incluido un servicio encargado de crearlas.
Es importante tener en cuenta que se utiliza el estándar PKCS1, el formato PEM y el tipo de padding RSA_PKCS1_PADDING.

Para generar el mensaje firmado necesario para registrar al autor, se utiliza un string. Este string, que originalmente está en texto plano, conforma el mensaje. El mensaje es pasado por un algoritmo de hashing, SHA-384 en nuestro caso. El hash resultante es luego encriptado utilizando la clave privada del autor. Ambos mensajes son enviados al contrato inteligente de registro de autor, que está presente en el chaincode. Dicho contrato por un lado pasa el mensaje por el mismo algoritmo de hashing, y por el otro desencripta el mensaje firmado utilizando la clave pública. Si ambos mensajes coinciden, el contrato tiene la certeza de que el emisor del mensaje tiene la clave privada del autor.

La principal diferencia con respecto al diseño es que en lugar de comprobar si el autor ya está registrado, se registra la nueva clave generada. Este cambio es debido a una decisión de seguridad. Se ha elegido utilizar JWT para la generación de claves de registro, las cuales caducan cada media hora. Cada vez que un autor desee participar en la red deberá demostrar su identidad registrándose nuevamente. Como el JWT caduca, no tiene sentido verificar si ya existe, por lo que se sobreescribe si ya estaba presente. De esta manera mitigamos los riesgos asociados a que la clave de registro sea filtrada, limitando enormemente el daño potencial.

Las claves utilizadas para la generación de los tokens se almacenan localmente dentro de los contenedores de chaincode y nunca entran en el ledger.

6.2.2 Creación de Update

Se ha añadido un nuevo flujo mediante el cual un usuario puede crear una actualización utilizando la app de autor.

1. El usuario introduce los datos en los distintos campos del manifiesto, y selecciona o escribe un payload.
2. El usuario clica el botón de firmar payload, lo que envía una petición al agente autor.
3. El agente autor firma la petición y devuelve el digest y la firma.
4. El digest es añadido al manifiesto y la firma a la actualización para el payload.
5. El usuario clica el botón de firmar manifiesto, lo que envía una petición al agente autor.
6. El agente autor firma la petición y devuelve el digest y la firma.
7. El digest es añadido al manifiesto y la firma a la actualización para el manifiesto.

8. El usuario clica el botón de registrar actualización, lo que comienza el flujo de registro de actualización.

El uso de este flujo es opcional. Si el autor cuenta con sus propios medios para crear manifiestos y para realizar las firmas correspondientes, puede utilizar la API del agente autor de manera directa.

6.2.3 Registro de Actualizaciones

Para este flujo tanto el agente de autor como el de registro utilizan la ruta /register

1. El agente autor recibe la petición que contiene una clave pública y un objeto JSON o un cuerpo multipart/form-data equivalente a la estructura Update.
2. El agente autor realiza la verificación del contenido del input.
3. El agente autor recupera de su base de datos la clave de registro asociada a la clave pública.
4. El agente autor construye un objeto UpdateRegister.
5. El agente autor envía una petición Post al agente de registro con el objeto creado.
6. El agente de registro verifica los contenidos de la petición.
7. El agente de registro llama al chaincode de registro y le envía el objeto UpdateRegister.
8. El agente de registro comprueba si la clave de registro está asociada a una clave pública registrada, y si el token es aún válido.
9. El chaincode verifica los contenidos de la petición. El servicio encargado delega en distintos métodos la verificación de los campos, el hash del manifiesto y el hash del payload. Para estos hashes se utiliza el algoritmo SHA-384 con la librería crypto de NodeJS.
10. El chaincode genera el objeto UpdateInChain.
11. El chaincode almacena como asset el objeto creado.
12. El chaincode devuelve al agente registro un mensaje de exito.
13. El agente registro almacena el payload en el nodo IPFS mediante la librería ipfs-http-client. Se obtiene el CID.
14. El agente registro llama al chaincode de registro para que almacene el cid. Se utiliza la clave del autor y las ID de versión y clase del manifiesto para encontrar el asset, por lo que se envían como parámetros.
15. El chaincode recupera la actualización y añade el CID.
16. El chaincode devuelve un mensaje de éxito.
17. El agente de registro devuelve un mensaje de éxito.
18. El agente autor devuelve un mensaje de éxito.

Los procesos de verificación de información firmada se realizan de la misma manera que en el flujo anterior, utilizando los algoritmos RSA-2048 y SHA-384.

En este flujo se introduce la conexión con la red IPFS. Afortunadamente, la librería cliente se encarga de gestionar el proceso de manera sencilla. Cabe mencionar que el fijado de los contenidos debe ser indicado explícitamente, ya que por defecto los contenidos subidos a la red son susceptibles de ser eliminados por el proceso de limpieza.

En este caso, la única diferencia con el diseño original es la necesidad de verificar la validez del token.

6.2.4 Recuperación de versiones

Para este flujo, el agente de recuperación utiliza la ruta /retrieve/version

1. El dispositivo (o la app que lo sustituye) solicita la versión a través de la ruta en el agente de recuperación. Para ello envía su DeviceID, que está compuesto por la clave pública de su autor y su clase de dispositivo.
2. El agente de recuperación verifica la petición y llama al chaincode.
3. El contrato de recuperación obtiene la actualización y devuelve el número de la versión.
4. El agente de recuperación envía de vuelta la versión al dispositivo.

6.2.5 Recuperación de actualizaciones

Para este flujo, el agente de recuperación utiliza la ruta /retrieve

1. El dispositivo (o la app que lo sustituye) solicita la actualización a través de la ruta en el agente de recuperación. Para ello envía su DeviceID, que está compuesto por la clave pública de su autor y su clase de dispositivo.
2. El agente de recuperación verifica la petición y llama al chaincode.
3. El contrato de recuperación obtiene la actualización y la devuelve al agente.
4. El agente utiliza el CID para obtener el payload de IPFS
5. El agente utiliza la actualización en cadena y el payload para construir el objeto de actualización, y lo devuelve.
6. El dispositivo realiza la verificación de la actualización. En este caso, la app del dispositivo llama al agente de verificación para que realice la comprobación.

Para los flujos de recuperación se ha seguido el diseño sin introducir ningún cambio significativo.

6.2.6 Verificación de actualizaciones

Se ha añadido este flujo para el agente de verificación. Su objetivo es permitir realizar verificaciones a dispositivos más restringidos que no cuenta con un gateway y a la app de dispositivo. Se accede a través de la ruta /verify.

1. El dispositivo envía la petición conteniendo su DeviceID.
2. El agente utiliza la clave pública para verificar los contenidos de las firmas tanto del payload como del manifiesto.
3. El agente envía true de vuelta si la verificación es correcta, o un error si no lo es.

Los pasos seguidos para realizar esta verificación son los mismos que realiza el contrato de registro de actualizaciones dentro de la aplicación blockchain. Para ello se utilizan los mismos algoritmos RSA-2048 y SHA-384.

7 Pruebas

En este capítulo se describen las pruebas realizadas sobre la implementación del proyecto. Dichas pruebas se dividen en dos tipos: funcionales y de rendimiento. Las pruebas funcionales tenían como objetivo comprobar el correcto funcionamiento de la solución, teniendo en cuenta que no es un producto destinado al público sino una implementación de carácter investigativo. Las pruebas de rendimiento tenían como objetivo medir el tiempo necesario para ejecutar los flujos de la aplicación, para posteriormente añadir las conclusiones al proyecto de investigación del que forma parte este TFG.

7.1 Pruebas de funcionalidad

Las primeras pruebas realizadas estaban orientadas a verificar el correcto funcionamiento de la aplicación. Es aquí, durante la implementación, donde se introducen pruebas unitarias.

Se han añadido pruebas funcionales a los agentes de registro y autor. Estas pruebas están implementadas utilizando las librerías Jest, que permite realizar testeo unitario, y superTest, que facilita las pruebas funcionales contra los endpoints de las APIs. Puede verse un ejemplo del uso de estas librerías en la figura 7.1.
Se han implementado pruebas tanto para probar el funcionamiento de servicios individuales como para los flujos completos contra los endpoints.

Para facilitar estas pruebas se ha añadido una carpeta en cada agente que contiene varios archivos JSON que se utilizarán para simular el contenido de las peticiones. Estos ficheros contienen casos de pruebas con inputs correctos para testear el correcto funcionamiento de la solución, así como casos en los que se espera que sean rechazados en algún punto de la ejecución.

Es importante mencionar que, aunque se han implementado pruebas en los agentes de autor, recuperación y registro, la solución presentada en este TFG no está destinada a ser utilizada por usuarios en un contexto real. En su lugar, forma parte de un proyecto de carácter investigativo, y por lo tanto, no se espera de él una gestión de errores al nivel de un producto comercial.

Las pruebas más importantes las encontramos en el agente autor, concretamente en las pruebas a sus endpoints. A través de estas pruebas se testea todo el flujo del proceso de registro, pasando por el agente de registro y el chaincode. Se han incluido casos correctos y erróneos, de manera que se testee lo mejor posible la solución en su conjunto. Algunas de estas pruebas están destinadas a fallar en otros componentes, como el caso de que el proceso sea correcto pero el manifiesto no sea verificable, error que salta en la comprobación on-chain.

```
describe('/register', () => {
  test('Correct input', (done)=> {
      var json = readJSON("./test-json/update-register.json",'V_1');
      request
      .post("/register")
      .send(json)
      .expect(201)
      .end((err, res) => {
          if (err) return done(err);
          return done();
      });
  },20000);
```

fig 7.1 - Test de caso correcto para registro de actualizaciones.

7.2 Pruebas de rendimiento

Una vez finalizada la implementación, se decidió implementar una serie de pruebas para medir el comportamiento de la solución y su rendimiento.

7.2.1 Pruebas manuales

En primer lugar, se realizaron algunas pruebas manuales (fig 7.2) para obtener una idea general del rendimiento del proceso de registro.

	Payload Size	Initial Date	store Date	IPFS Date	Dif	update CID Date	Dif	End date	Dif	Manifest Store	Full STORE	Final Response	TOTAL Seconds
2	**552 KB**												
3		1678366675586	1678366676155	1678366676517	362	1678366676791	274	1678366678987	2196	**569**	**1205**	**3401**	**3.401**
4		1678367066422	1678367066875	1678367067293	418	1678367067595	302	1678367069772	2177	**453**	**1173**	**3350**	**3.35**
5		1678367233877	1678367234370	1678367234766	396	1678367234991	225	1678367237215	2224	**493**	**1114**	**3338**	**3.338**
6	**26.2 MB**												
7		1678368554326	1678368561443	1678368580347	18904	1678368581137	790	1678368583653	2516	**7117**	**26811**	**29327**	**29.327**
8	UPGRADED MACHINE												
9	**552 KB**												
10		1678371533781	1678371534131	1678371534388	257	1678371534611	223	1678371536707	2096	**350**	**830**	**2926**	**2.926**
11	**9.5MB**												
12		1678434323673	1678434326225	1678434328622	2397	1678434328831	209	1678434330973	2142	**2552**	**5158**	**7300**	**7.3**
13		1678434843853	1678434845866	1678434847542	1676	1678434847766	224	1678434849878	2112	**2013**	**3913**	**6025**	**6.025**
14		1678435033628	1678435035479	1678435037305	1826	1678435037526	221	1678435039711	2185	**1851**	**3898**	**6083**	**6.083**
15	**15.4 MB**												
16		1678373029681	1678373033687	[illegible]36827	3140	1678373037166	339	1678373039313	2147	**4006**	**7485**	**9632**	**9.632**
17	**26.2MB**												
18		1678370792091	1678370797788	1678370803475	5687	1678370803977	502	1678370806186	2209	**5697**	**11886**	**14095**	**14.095**
19		1678371164506	1678371170928	1678371175567	4639	1678371175809	242	1678371178079	2270	**6422**	**11303**	**13573**	**13.573**
20		1678371363362	1678371369294	1678371374767	5473	1678371375072	305	1678371377230	2158	**5932**	**11710**	**13868**	**13.868**
21	**36.1 MB**												
22		1678373421525	1678373430066	1678373438286	8220	1678373438739	453	1678373441067	2328	**8541**	**17214**	**19542**	**19.542**

fig 7.2 - Resultados de testeos manuales del proceso de registro de actualizaciones

Durante estas pruebas, se descubrió que la solución estaba limitada a archivos ligeramente inferiores a 40 MB. Dicha limitación puede modificarse editando la configuración de la red Hyperledger Fabric, pero se decidió que no era necesario para la elaboración del TFG. También se descubrió que era necesario aumentar la capacidad de la máquina donde está desplegada la solución, debido a que la falta de capacidad hacía que los contenedores de chaincode fueran incapaces de atender peticiones de 26MB o superiores, obteniendo fallos en el servicio de endorsement de Fabric. La máquina pasó de tener 4 a 10 Gb de memoria RAM, lo que además aceleró considerablemente el proceso. Es posible que aumentar más las capacidades de la máquina mejore los tiempos, pero no se contaba con más recursos.

	Payload Size	Initial Date	Retrieve	Retrieve res	Dif	IPFS retrieve	Dif	End date	Dif	Update retrieve	Full Retrieve	Final Response	TOTAL Seconds
2	**552 KB**												
3		1678706423664	1678706424027	1678706426127	2100	1678706426151	24	1678706426494	343	**363**	**2487**	**2830**	**2.83**
4	**9.5MB**												
5		1678707554417	1678707554801	1678707556890	2089	1678707556912	22	1678707628235	71323	**384**	**2495**	**73818**	**73.818**
6	**15.4 MB**												
7		1678707786915	1678707787262	1678707789323	2061	1678707789388	65	1678707981776	192388	**347**	**2473**	**194861**	**194.861**
8		1678707982070	1678707982584	1678707984683	2099	1678707984723	40	1678708178660	193937	**514**	**2653**	**196590**	**196.59**
9		1678711461626	1678711462160	1678711464295	2186	1678711464346	51	1678711660968	196622	**534**	**2720**	**199342**	**199.342**
10	Retrieval Code Improved												
11	**552 KB**												
12		1678782473673	1678782474302	1678782476399	2097	1678782476419	2117	1678782476518	99	**2726**	**2746**	**2845**	**2.845**
13	**9.5MB**												
14		1678785224903	1678785225517	1678785227637	2120	1678785227679	42	1678785227948	269	**614**	**2776**	**3045**	**3.045**
15	**15.4MB**												
16		1678785539346	1678785539819	1678785541877	2058	1678785541900	23	1678785542351	451	**473**	**2554**	**3005**	**3.005**
17	**36.1 MB**												
18		1678785788697	1678785789041	1678785791091	2050	1678785791100	9	1678785792050	950	**344**	**2403**	**3353**	**3.353**
19		1678786487929	1678786488270	1678786490378	2108	1678786490419	41	1678786491249	830	**341**	**2490**	**3320**	**3.32**

fig 7.3 - Resultados de testeos manuales del proceso de recuperación de actualizaciones

Después se realizaron pruebas manuales para comprobar el comportamiento del proceso de recuperación fig 7.3). Rápidamente se descubrió que el proceso era demasiado lento para ser utilizable, lo que revelaba la existencia de algún error en la implementación.
Efectivamente, se encontraron dos puntos que una vez solucionados aceleraron el proceso de manera muy significativa. Por un lado, el procesamiento de los contenidos de IPFS se estaba realizando mediante concatenaciones de los “chunks” de la respuesta, lo que añadía tiempo innecesario. Se soluciono agrupando los elementos de la respuesta en una pila y concatenando solo una vez al final.

Por otro lado, la planificación inicial contemplaba dar la actualización como una única respuesta. Al no existir una manera predeterminada de responder con objetos múltiples en HTTP (al contrario que al solicitar con multipart/form-data), la respuesta estaba siendo enviada en JSON. El procesamiento adicional y el peso añadido de convertir los payloads a strings estaban añadiendo tiempo extra a la respuesta. Se soluciono separando la ruta de /retrive en retrieve/update y /retrieve/payload, de manera que se pudiese enviar la actualización separada, lo que permite enviar el manifiesto como JSON y el payload cómo binario.

Una vez solucionados ambos problemas, los resultados cambiaron muy positivamente.

7.2.2 Pruebas automáticas

Se creó una herramienta para realizar múltiples peticiones y capturar los tiempos de los resultados. Los tiempos quedan recogidos en distintos ficheros de texto. Estos ficheros después son cargados en un cuaderno jupyter para su procesado.
En el proceso de registro, se realizaron al menos 100 pruebas para imágenes de distintos tamaños, desde 552 KB hasta 36.1 MB. Se recolectó información sobre los tiempos de uso de la solución de manera semi-automática: Cada petición a la ruta deja un rastro de fechas Unix en una serie de logs que pueden ser recolectados desde los contenedores Docker mediante el uso de volúmenes, que enlazan los contenidos de dos carpetas. Estos logs después son procesados mediante el uso de un cuaderno jupyter, para obtener las figuras 7.3 y 7.4.

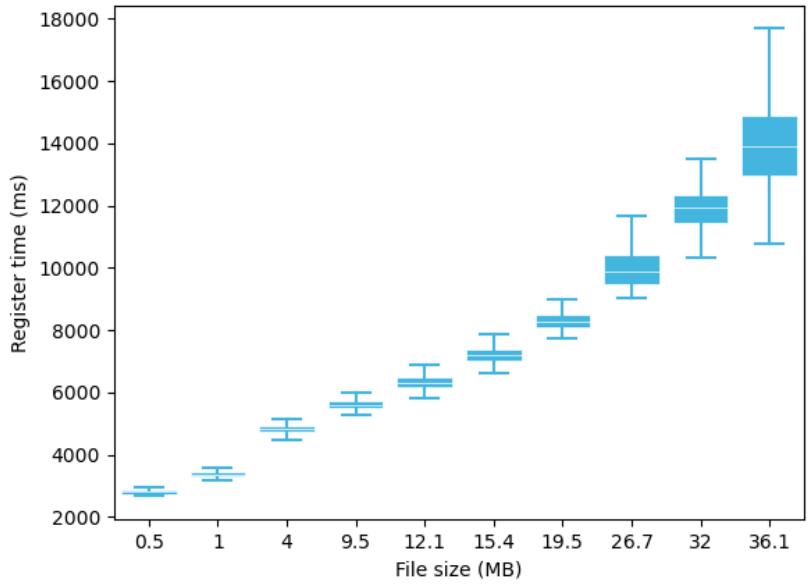

fig 7.4 Tiempos de registro totales

En la figura 7.3 podemos ver la distribución de los tiempos totales de registro. Estos tiempos son capturados desde que el agente de registro recibe la petición hasta que está listo para enviar la respuesta. En la figura 7.4 podemos ver los tiempos medios para las distintas etapas del proceso.

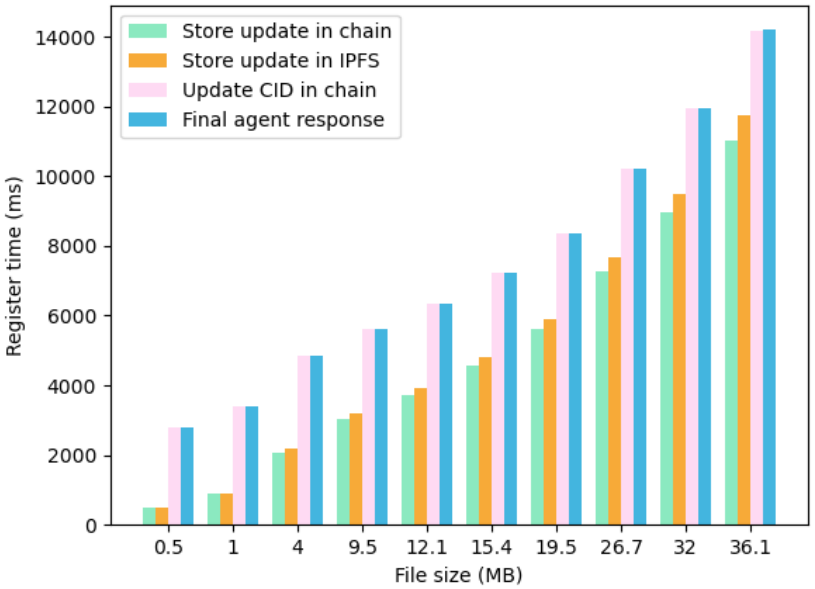

figura 7.5 - Tiempos medios de ejecución del proceso de registro

Podemos observar que el incremento de los tiempos se hace más pronunciado a medida que aumentamos el tamaño de los archivos. Adicionalmente, a mayor tamaño de archivo más varían los tiempos entre distintas peticiones. Dentro del proceso, la tarea que más aumenta en tiempo de ejecución es el guardado del manifiesto en la blockchain. Esto se debe a que la actualización completa, incluyendo el payload, viaja entre los dos agentes y el contenedor de chaincode. Vemos por otro lado que la subida a IPFS no varía tanto al aumentar el tamaño de archivo.

De la misma manera, se capturaron los tiempos para el proceso de recuperación, que quedan reflejados en las figuras 7.5 y 7.6.

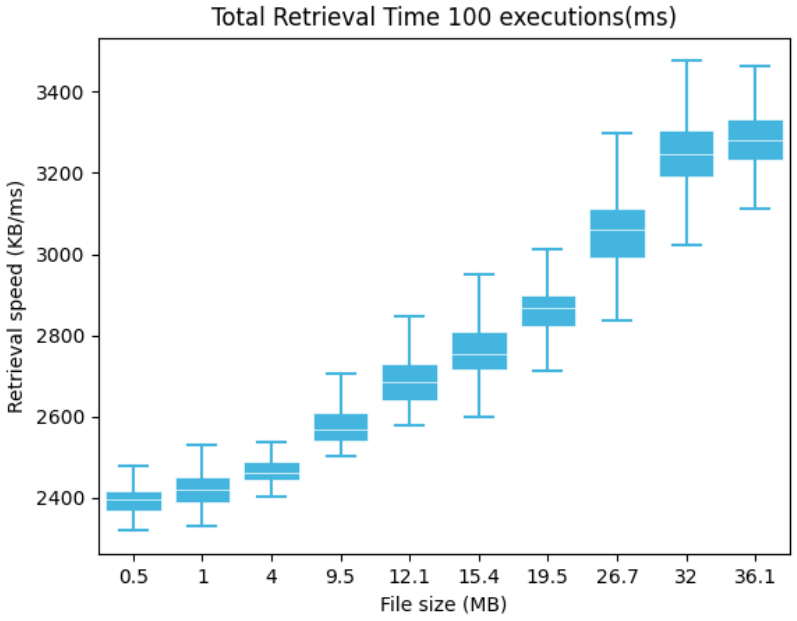

fig 7.6 - Tiempos totales de recuperación

En este caso, la captura de tiempos comienza cuando el agente de registro recibe la petición, y concluye justo antes de que este envíe su respuesta.

Como se puede observar, los tiempos no incrementan mucho conforme se aumenta el tamaño de archivo a recuperar. Cabe destacar que la recuperación del manifiesto tiene aproximadamente la misma duración sin importar el tamaño del payload de la actualización. La recuperación desde IPFS por otro lado aumenta de manera moderada.

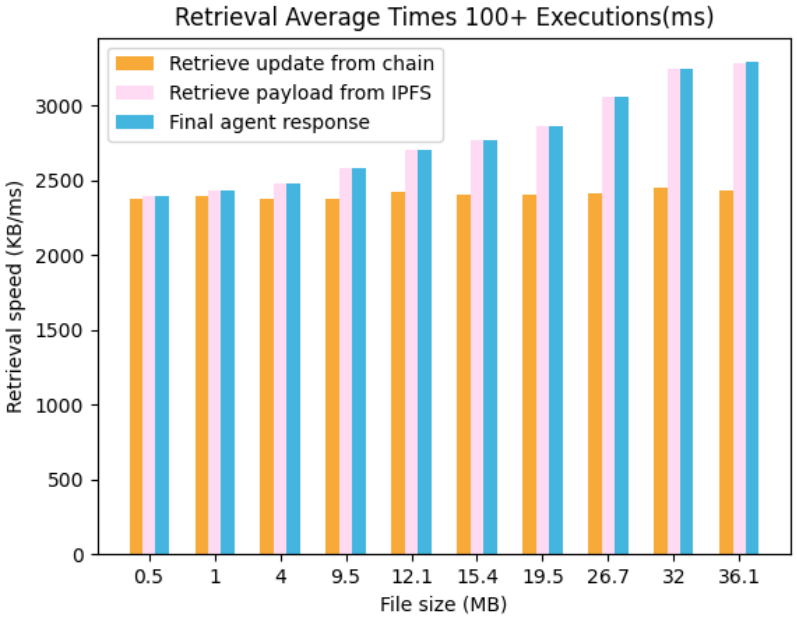

fig 7.7 - Tiempos medios de recuperación

Finalmente, en la figura 7.7 se puede ver una comparación entre la eficiencia de ambos procesos, entendida como la cantidad de KBs que se registran o recuperan por milisegundo. Podemos observar que el proceso de recuperación aumenta en eficiencia mucho más rápido que el de registro, pero ambos procesos aumentan su eficiencia a medida que crece el tamaño del archivo. Estos resultados nos hacen pensar que la solución es lo suficientemente veloz y eficiente para justificar próximos trabajos de investigación.

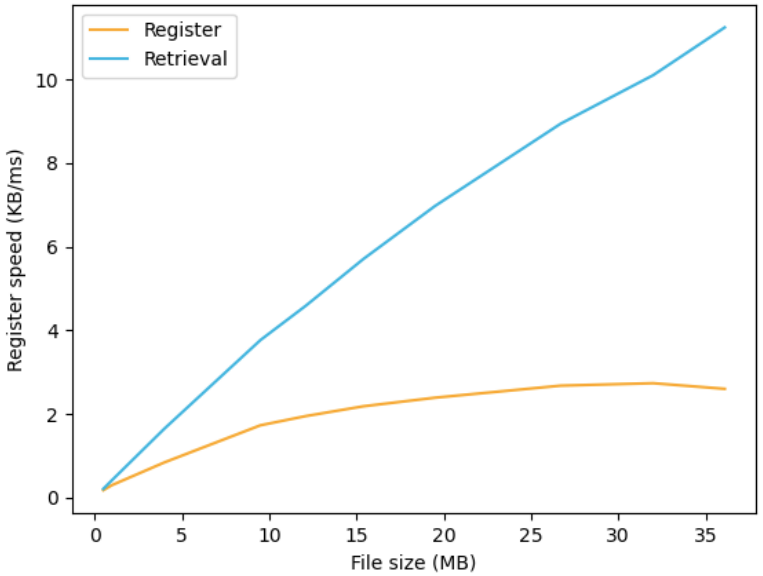

fig 7.8 - eficiencia para ambos procesos

8 Retos del proyecto

En este capítulo se recogen todos los desafíos que el alumno ha debido superar para la correcta elaboración del proyecto. Los desafíos han sido tanto de carácter técnico como conceptual, ya que todas las tecnologías utilizadas eran nuevas para el alumno.

8.1 Nuevas tecnologías

El desarrollo del proyecto comenzó con la fase de investigación. Esta fase resultó clave porque el alumno no conocía las tecnologías de ledger distribuido, ni conocía el funcionamiento del entorno web3 u otras iniciativas descentralizadas. Por otro lado, el concepto del internet de las cosas también resultaba nuevo. También había que familiarizarse con el problema a solucionar y las propuestas previas. Debido a que dichas tecnologías son recientes, los recursos disponibles resultan más escasos, y los que existen están más dirigidos a un público con conocimientos previos, por lo que la etapa inicial resultó complicada. Adicionalmente, aunque el trabajo se comenzó en diciembre, dejando mucho tiempo hasta la fecha de la defensa, el convenio tiene un límite más cercano, en marzo, por lo que esta fase de adquisición de conocimientos tuvo que ser terminada con cierta prisa para poder tiempo de diseñar e implementar la solución.

8.2 NodeJS Como BackEnd

El alumno contaba con una experiencia muy limitada en el uso de JavaScript y no tenía ninguna experiencia en el uso de NodeJS. Su uso presenta varios desafíos que el alumno ha debido superar.

Por un lado, JavaScript no utiliza objetos tipados. Por lo tanto, cada objeto intercambiado entre APIs debe ser tratado con especial atención. Este problema supuso un reto en las comparaciones realizadas en la blockchain. Un caso concreto es la comparación entre el valor del hash presente en el manifiesto (un string dentro de un objeto JSON) y el obtenido tras pasar el propio manifiesto por el algoritmo de hashing (un string primitivo). Aunque en apariencia ambos objetos son iguales, las comparaciones sólo son verdaderas si ambos objetos son la misma instancia.

Este problema fue resuelto reduciendo el valor de los objetos a primitivos y realizando entonces las comparaciones. También se ha utilizado la técnica de convertir en strings todos los valores antes de ser enviados y reconstruyendo los objetos una vez recibidos.

Por otro lado, NodeJS introduce el concepto de asincronía. Cumple una función práctica que se podría comparar con la programación concurrente. Se lleva a cabo mediante la emisión de promesas. Cuando una función asíncrona es llamada, el resultado de dicha función es una promesa no resuelta. La ejecución continua, y si algún otro punto del programa hace uso de la variable en la que se encuentra dicha promesa encontrará un valor “vacío”. Esto puede hacer que un programa deje de funcionar correctamente. Por ello se introduce también el concepto de la espera. La ejecución puede esperar en un punto concreto a que una promesa se resuelva.
Las promesas pueden ser resueltas de manera explícita (con Promise.resolve(valor a devolver)) o implícitamente (con return).

Aunque no se trata de un problema en sí mismo (se trata más bien de una característica muy útil) añadió un tiempo adicional al desarrollo debido a la necesidad de comprender el funcionamiento y la gramática relacionada a la misma y a JavaScript, que pueden resultar difíciles de utilizar correctamente inicialmente, como puede ser el caso de las funciones flecha utilizadas como middleware, por ejemplo.

8.3 Complejidad de la solución

La solución hace uso de muchos elementos que son dependientes entre sí para el correcto funcionamiento de los distintos flujos. Dichos elementos están además contenidos utilizando Docker, lo que añade una capa de dificultad adicional a la interconectividad de los mismos. Esto hace que inicializar y configurar toda la red de elementos suponga una labor que consume grandes cantidades de tiempo.

Para solucionarlo parcialmente se utiliza Docker-Compose, una herramienta que permite ejecutar la inicialización de manera conjunta de los distintos elementos, y se han escrito varios scripts que automatizan parte del proceso. Parte de la configuración sin embargo debe seguir siendo manual.

Por otro lado, la configuración y conexión individual de algunos elementos resultó especialmente complicada, como la conexión entre el agente de registro y la red blockchain. Al utilizarse tecnologías relativamente nuevas y en constante evolución, la documentación disponible no siempre sigue el ritmo al desarrollo.

8.4 Criptografía

El alumno no tenía experiencia en el uso de técnicas criptográficas, que son cruciales para el proyecto. Algunos de los problemas encontrados durante el desarrollo han sido los siguientes:

Los pares de claves deben seguir un formato muy estricto. En el caso de las claves RSA que utiliza la librería crypto de NodeJS, deben contener padding (que indica que empieza y acaba la clave (similar a '—Start— Clave —End—') y deben estar en formato de buffer con base64. También se ha encontrado el problema de que es difícil gestionar estos formatos de manera dinámica. Cuando se utilizan de manera manual o se usan los comandos de copiar y pegar, las claves tienden a corromperse.
La solución implementada es la de tratar con las claves únicamente al sacarlas de un archivo local con la librería fs o de la base de datos, y enviarlas en formato de string. A estos almacenamientos de claves suele dársele el nombre de cartera (wallet).

Otro problema ha sido el determinismo necesario para la generación de hashes. Dos objetos JSON, que son el formato elegido para el intercambio de información entre APIs, aunque puedan parecer idénticos no son necesariamente iguales debido a algunos metadatos (como la fecha de creación de un elemento del mismo). Si enviamos un JSON a través de la API, el JSON tratado por la misma será diferente ya que el método stringify (que utilizamos para procesar un JSON antes de enviarlo)

no es determinista (serializa las claves no numéricas por fecha de creación y no por orden alfabético, por ejemplo). Por lo tanto al realizar el hashing del objeto nos podemos encontrar con que no coincide con el hecho previamente para rellenar el campo manifest digest, aunque ambos JSON contengan las mismas claves y los mismos valores para las mismas.
Para solucionar este problema utilizamos la librería json-stringify-deterministic, que nos asegura que si dos JSON tienen los mismos valores para las mismas claves, el resultado de realizar el método stringify será igual en ambos casos.

Este determinismo es además importante para las operaciones en la blockchain. Las transacciones requieren que los distintos nodos endorsers tengan resultados iguales para las mismas operaciones, lo que se usa para validar los contratos. Estas pequeñas diferencias pueden hacer que un asset a almacenar, aunque a efectos prácticos sean iguales, no lo sean de manera literal, lo que a su vez hará que una transacción que nos interesa que se valide acabe siendo rechazada.

8.5 Archivos Pesados

Al principio el proyecto se elaboró utilizando JSON como formato para el intercambio de información entre componentes. Durante la implementación de los procesos tanto de registro como de recuperación dio buenos resultados y permitía un procesado conveniente de la información. Una vez iniciada una serie de pruebas más elaboradas, sin embargo, se descubrió rápidamente que la solución no soportaba imágenes más pesadas.

Aunque el trabajo realizado hasta el momento seguía siendo útil, ya que para algunas actualizaciones los payloads necesarios son de tamaño muy reducido, fue necesario añadir soporte para otro tipo de formato de intercambio de información para actualizaciones más pesadas. Se modificaron todos los componentes para poder recibir y/o enviar información en formato multipart/form-data, que permite enviar ficheros binarios.

8.6 Hyperledger Fabric SDK para JavaScript

Fabric está desarrollado principalmente en Go, y sus primeras versiones sólo soportaban contratos en este lenguaje. Como consecuencia, la comunidad de desarrolladores se construyó y formó inicialmente en las librerías Go.
Aunque son muy similares y utilizan los mismos nombres de funciones, las librerías para distintos lenguajes tienen algunas diferencias sutiles. El problema surge cuando toda la documentación e interacciones en foros como Stack Overflow está escrita para la librería de Go, mientras que para la librería de NodeJS apenas hay soluciones en internet.
Esto hace que las soluciones deban ser adaptadas para el uso con JavaScript, que a veces varía en la forma en la que implementa algunas operaciones o en la forma en la que trata algunos tipos de datos.
A la hora de instalar los contratos, las instrucciones a seguir también varían, ya que los distintos lenguajes tienen una manera distinta de ser compilados y ejecutados. En este aspecto, trabajar con contratos NodeJS tiene un proceso de instalación más complicado.

Por desgracia, este problema no fue resuelto directamente, sino que se trabajó a pesar de él. Para futuras iteraciones del proyecto, se ha planteado la posibilidad de reescribir los contratos en Go, ya que no resultaba factible hacerlo durante la ejecución del TFG.

8.7 Elaboración de un artículo académico

El proyecto concluyó con la elaboración de un artículo académico, escrito con la intención de ser enviado a una revista académica. La elaboración de este artículo supuso un reto, ya que además de no contar con conocimientos previos al respecto, se tuvo que elaborar en una ventana de tiempo reducida. Este reto, sin embargo, resultó muy instructivo una vez fue solucionado.

9 Seguimiento y control

En este capítulo se muestra el trabajo realizado en la tarea de seguimiento y control (SyC). Se describe cómo se han gestionado el alcance, tiempo, calidad y riesgos del proyecto.

9.1 Gestión del alcance

El alcance del proyecto era uno de los puntos que más se discutió durante la planificación del proyecto, dado que en un principio parecía poco probable que el tiempo disponible permitiese elaborar todos los elementos de la solución.
En la práctica, sin embargo, se completaron todos los objetivos principales establecidos. En cuanto a los secundarios, el despliegue de una red de producción fue el único aspecto que no pudo completarse, aunque se elaboró parcialmente y se añadió una red IPFS personalizada. Cabe destacar que también se elaboró un artículo académico que no se concebía como un objetivo en la etapa de planificación. Dicho artículo se escribió para ser enviado a una revista académica. Esto hizo que el apartado de pruebas se extendiese para incluir un testeo semi automático que midiese el rendimiento de la aplicación para distintos tamaños de archivo, lo que no se tuvo en cuenta en la planificación.
Finalmente, todos los objetivos del alumno se cumplieron, y este obtuvo una experiencia muy enriquecedora.

Por otro lado, en cuanto a los requisitos establecidos, todos fueron satisfechos.

9.2 Gestión del tiempo

La gestión del tiempo ha sido la parte que más se ha desviado respecto a la planificación inicial. La tabla completa puede comprobarse en el anexo C.
De manera resumida, la investigación y el diseño de la solución tomaron menos tiempo del planificado, pero la implementación, las pruebas, la gestión del proyecto y el trabajo académico tomaron más tiempo del previsto, en parte debido a que se decidió finalmente elaborar los objetivos secundarios que no se tuvieron en cuenta durante la planificación inicial. En total se pasó de las 356 horas planificadas a 484 horas de duración del proyecto.

El proceso de investigación resultó significativamente más reducido que el estimado originalmente, ya que el proyecto tardó mucho más tiempo del esperado en obtener la aprobación por parte de la universidad y del centro, lo que permitió que el alumno se familiarizara de antemano con muchos de los conceptos más importantes del proyecto.
El diseño también fue más rápido de lo esperado, ya que muchos de los elementos tenían flujos muy similares.
La implementación fue más complicada de lo previsto en cuanto a horas de trabajo, ya que distintos fallos y mejoras descubiertos en las pruebas obligaron al alumno a realizar ajustes importantes. La documentación del repositorio también llevó mucho más tiempo del esperado, pero se consiguió una documentación muy completa del proyecto. También se implementaron las apps de autor y dispositivos que no entraban en la planificación original.

Las pruebas también llevaron mucho tiempo, sobre todo las pruebas de rendimiento que no estaban previstas al inicio del proyecto. Estas pruebas surgieron de la necesidad de añadir una conclusión al artículo académico. Eso es debido a que tomaban mucho tiempo al ejecutarse secuencialmente. Las pruebas de archivos más grandes llevaban la máquina virtual a su límite, por lo que se pasó mucho tiempo repitiendo pruebas al aumentar las capacidades de la máquina y reinstalando la red cada vez que ocurría un error. El proceso de instalación de la solución puede tomar hasta media hora, y se repitió varias veces. Por otro lado, las pruebas del agente autor acabaron funcionando como pruebas funcionales para el proceso de registro en su integridad, y se acabaron alargando bastante.
Finalmente, la elaboración de documentos también tomó más tiempo del previsto. La memoria se extendió 20h, y se elaboró el artículo, que añadió 50h al proyecto.

El proyecto en su conjunto se alargó en alrededor de 135h, aunque este aumento no tiene en cuenta las horas que el alumno invirtió antes del inicio oficial del mismo. Si tenemos en cuenta la investigación previa realizada, unas 35 horas, se podría hablar de alrededor de unas 170 horas adicionales de trabajo con respecto a la planificación original, aproximando la duración real del proyecto a las **515 horas**.

En cuanto a la dedicación de cada tarea a lo largo del tiempo, el diagrama de Gantt elaborado refleja casi a la perfección la realidad de la elaboración del proyecto. La única diferencia significativa es que la implementación también incluyó una fase adicional, y que en las dos últimas semanas se añadió la tarea de elaboración del artículo. El Gantt final se aprecia en la figura 9.1.

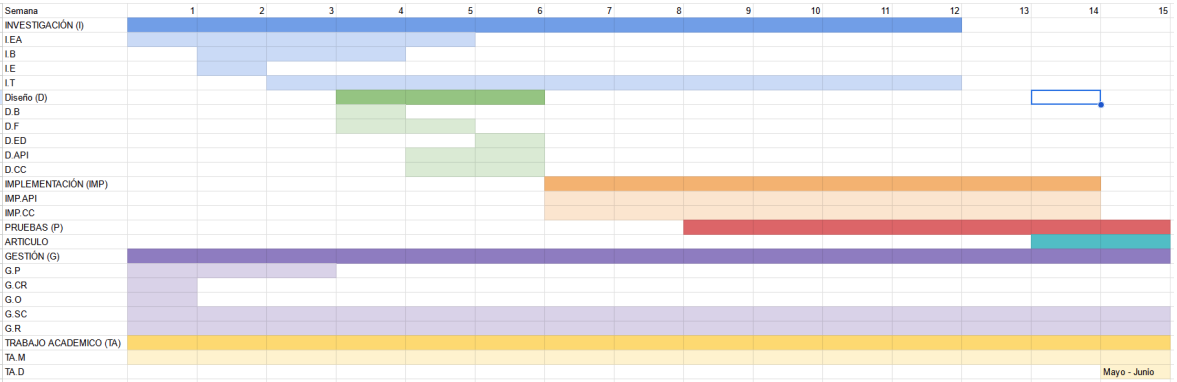

fig 9.1 - Diagrama de Gantt SyC

Para la elaboración de un futuro proyecto similar, las lecciones aprendidas más importantes serían el no subestimar las capacidades de aprendizaje del alumno, así como dedicar más tiempo en la planificación a todas las tareas académicas y de gestión, al igual que dedicar más tiempo para las pruebas.

9.3 Gestión de los riesgos

Los planes de prevención y la planificación del proyecto llevaron a que el proyecto se llevará a cabo sin realizar recorte en el alcance. Por el contrario, el aumento del tiempo dedicado se debe mayoritariamente a la inclusión de nuevas tareas. Por lo tanto, los riesgos no se materializaron y no hubo que poner en práctica ningún plan de acción.

10 Conclusiones

El proyecto se llevó a cabo satisfactoriamente. Se implementaron todos los elementos diseñados y aunque no está al nivel de pulido de un producto comercial, la solución ofrece un rendimiento competente, como se ha mostrado en el capítulo 6.
La solución ofrece un punto de partida muy sólido para continuar con futuras investigaciones, y ha demostrado que es viable utilizar una arquitectura distribuida para la actualización segura de firmware.

Como demostración de la elaboración satisfactoria del proyecto, se ha escrito un artículo académico presentando los resultados obtenidos, el cual ha sido enviado al journal Internet Of Things, y está esperando ser revisado a la fecha de la escritura de este documento.

Durante la elaboración del TFG se han identificado varias rutas de trabajo futuro:

- La identidad de autores y especialmente dispositivos es difícil de verificar de manera rápida e inequívoca. Las distintas propuestas anteriores no explican de manera explícita cómo llevar a cabo este proceso, dejando al implementador la decisión de cómo realizar la verificación.
- El protocolo HTTP utilizado dificulta enormemente la participación de los dispositivos más limitados en la solución. Futuras propuestas deberán tener en cuenta estas limitaciones, ofreciendo un protocolo seguro de comunicación entre dispositivos o gateways o utilizando protocolos seguros más ligeros.
- Aunque la solución permite que las actualizaciones permanezcan disponibles después de la desaparición de su autor, no existe un método para legar la responsabilidad de crear nuevas actualizaciones a un nuevo autor.
- No existe un estudio que compare distintas topologías de redes blockchain para evaluar cómo pueden afectar al rendimiento de la solución.

Por otro lado, el alumno ha obtenido una valiosa experiencia, tanto a nivel laboral como en cuanto a su introducción al mundo de la investigación académica.

Bibliografía

Artículos científicos

[2] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., & Zhou, Y. (2017). Understanding the mirai botnet. In Proceedings of the 26th USENIX Security Symposium (pp. 1093-1110). (Proceedings of the 26th USENIX Security Symposium). USENIX Association.

[3] Sicari, Sabrina & Rizzardi, Alessandra & Grieco, Luigi & Coen-Porisini, Alberto. (2015). Security, privacy and trust in Internet of Things: The road ahead. Computer Networks. 76. 10.1016/j.comnet.2014.11.008.

[4] Bettayeb, Meriem & Nasir, Qassim & Abu Talib, Manar. (2019). Firmware Update Attacks and Security for IoT Devices: Survey. ArabWIC 2019: Proceedings of the ArabWIC 6th Annual International Conference Research Track. 1-6. 10.1145/3333165.3333169.

[7] Lee, B., & Lee, J. H. (2017). Blockchain-based secure firmware update for embedded devices in an Internet of Things environment. Journal of Supercomputing, 73(3), 1152-1167. https://doi.org/10.1007/s11227-016-1870-0

[8] Aymen Boudguiga, Nabil Bouzerna, Louis Granboulan, Alexis Olivereau, Flavien Quesnel, et al.. Towards Better Availability and Accountability for IoT Updates by means of a Blockchain. IEEE Security & Privacy on the Blockchain (IEEE S&B 2017) an IEEE EuroS&P 2017 and Eurocrypt 2017 affiliated workshop, IEEE, Apr 2017, Paris, France. hal-01516350

[9] A. Yohan and N. -W. Lo, "An Over-the-Blockchain Firmware Update Framework for IoT Devices," 2018 IEEE Conference on Dependable and Secure Computing (DSC), 2018, pp. 1-8, doi: 10.1109/DESEC.2018.8625164.

[10] Choi, S., & Lee, J. (2020). Blockchain-Based Distributed Firmware Update Architecture for IoT Devices. IEEE Access, 8, 37518-37525.

[11] M. Son and H. Kim, "Blockchain-based secure firmware management system in IoT environment," 2019 21st International Conference on Advanced Communication Technology (ICACT), 2019, pp. 142-146, doi: 10.23919/ICACT.2019.8701959.

[12] S. Dhakal, F. Jaafar and P. Zavarsky, "Private Blockchain Network for IoT Device Firmware Integrity Verification and Update," 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), 2019, pp. 164-170, doi: 10.1109/HASE.2019.00033.

[13] A. Yohan, N. -W. Lo and L. P. Santoso, "Secure and Lightweight Firmware Update Framework for IoT Environment," 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE), 2019, pp. 684-685, doi: 10.1109/GCCE46687.2019.9015316.

[14] M. -H. Tsai, Y. -C. Hsu and N. -W. Lo, "An Efficient Blockchain-based Firmware Update Framework for IoT Environment," 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), 2020, pp. 121-127, doi: 10.1109/AsiaJCIS50894.2020.00030.

[16] Albarqi, A. , Alzaid, E. , Ghamdi, F. , Asiri, S. and Kar, J. (2015) Public Key Infrastructure: A Survey. Journal of Information Security, **6**, 31-37. doi: 10.4236/jis.2015.61004.

[25] J. L. Hernández-Ramos, G. Baldini, S. N. Matheu and A. Skarmeta, "Updating IoT devices: challenges and potential approaches," 2020 Global Internet of Things Summit (GIoTS), Dublin, Ireland, 2020, pp. 1-5, doi: 10.1109/GIOTS49054.2020.9119514.

[26] N. S. Mtetwa, P. Tarwireyi, A. M. Abu-Mahfouz and M. O. Adigun, "Secure Firmware Updates in the Internet of Things: A survey," 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC), Vanderbijlpark, South Africa, 2019, pp. 1-7, doi: 10.1109/IMITEC45504.2019.9015845.
[27] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, & Jason Yellick (2018). Hyperledger fabric. In Proceedings of the Thirteenth EuroSys Conference. ACM.
[30] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, *21*(2), 120–126. doi:10.1145/359340.359342
[32] Figueroa-Lorenzo, Santiago & Benito, Javier & Arrizabalaga, Saioa. (2021). Modbus Access Control System Based on SSI over Hyperledger Fabric Blockchain. Sensors. 21. 5438. 10.3390/s21165438.
[33] Figueroa, Santiago & Añorga, Javier & Arrizabalaga, Saioa & Irigoyen, Inigo & Monterde, Mario. (2019). An Attribute-Based Access Control using Chaincode in RFID Systems. 1-5. 10.1109/NTMS.2019.8763824.
[34] Lorenzo, Santiago & Benito, Javier & Garcia Cardarelli, Pablo & Garaia, Jon & Arrizabalaga, Saioa. (2019). A Comprehensive Review of RFID and Bluetooth Security: Practical Analysis. Technologies. 7. 15. 10.3390/technologies7010015.

Referencias de interés

[0] Repositorio del proyecto
[1] RFC 9019 - Internet Engineering Task Force - 2021
[5] RFC 7228 - Internet Engineering Task Force - 2014
[6] Distributed Ledger in IIoT - Industrial Internet Consortium - 2020
[17] RFC 3647 - Internet Engineering Task Force - 2003
[22] RFC 9124 - Internet Engineering Task Force - 2022
[23] RFC 4122 - Internet Engineering Task Force - 2005
[24] Decentralized IDentifiers (DIDs) - W3C - 2022
[31] RFC 7519 - Internet Engineering Task Force - 2015

Tecnologías

[15] https://www.hyperledger.org/use/fabric
[18] https://www.docker.com/
[19] https://ipfs.tech/
[20] https://swagger.io/
[21] https://nodejs.org/en/
[28] Instalación de Hyperledger y la red de prueba
[29] Instalación de Chaincode sobre la red de prueba

Anexo A - Códigos de error

Durante la ejecución de los procesos de registro o recuperación pueden ocurrir distintos errores que deben ser notificados al autor o al dispositivo que los solicita.

Códigos de Error Generales:

- ERR_IPFS_NOT_AVAILABLE: La red IPFS o sus servicios no están disponibles.
- ERR_BC_NOT_AVAILABLE: La red blockchain o sus servicios no están disponibles.
- ERR_UPDATE_NOT_EXISTENT: Se intenta actualizar o recuperar una actualización que no existe.

Códigos de Error del Proceso de Registro:

- ERR_KEY_NOT_VERIFIABLE: La clave pública del autor no puede ser utilizada para verificar su identidad en el proceso de registro de autor.
- ERR_KEY_NOT_REGISTRABLE: La clave pública del autor ya está registrada en la red.
- ERR_KEY_NOT_VALID: La clave de registro utilizada no está presente en la blockchain.
- ERR_MANIFEST_NOT_VERIFIABLE: El manifiesto enviado no puede ser verificado con la firma y el digest enviados para la clave pública relacionada a la clave de registro enviada.
- ERR_INCORRECT_MANIFEST_FORMAT: El manifiesto no cuenta con todos los campos obligatorios.
- ERR_PAYLOAD_NOT_VERIFIABLE: La imagen/payload no puede ser verificada utilizando la firma y el digest enviados para la clave pública relacionada a la clave de registro enviada.

Anexo B - Instalación del proyecto

Para realizar la implementación se utiliza un servidor privado en la red interna del centro, al que se accede desde el ordenador de la oficina. Es posible, sin embargo, descargar los contenidos del repositorio para crear un entorno de ejecución en otro dispositivo. Es recomendable realizar dicha instalación en un dispositivo linux ya que facilita el proceso de instalación.

Para la utilización del repositorio de manera local deben cumplirse los siguientes requisitos:

- Instalar Docker y Docker Compose.
- Para realizar modificaciones sobre el código de la aplicación es necesario instalar NodeJS y npm.
- Los agentes y chaincodes necesitan una red de Hyperledger Fabric para poder ser ejecutados. Dado que el despliegue de una red es un proceso largo y complicado y no forma parte de los requisitos establecidos al inicio del TFG, se utiliza la red de prueba obtenible desde el repositorio oficial de Hyperledger Fabric [28].
- Es recomendable el uso de un IDE que soporte JavaScript. Para la implementación se ha utilizado Visual Studio Code, que ofrece extensiones que permite gestionar todos los elementos utilizados.
- Es necesario tener al menos un nodo IPFS en funcionamiento.

B.1 Inicialización de la Red de Pruebas

Como se ha indicado en el apartado anterior, se utiliza la red de pruebas de Hyperledger Fabric. El proceso de inicialización se explica en la documentación oficial, pero resulta laborioso ejecutarlo cada vez que se necesita reiniciar la red o instalar un chaincode. Por ello, se han escrito varios scripts que semi-automatizan el proceso, disponibles en el repositorio.

Para comenzar, nos situaremos en el directorio de instalación de la red. Si se ha instalado la red de prueba como en [28], el directorio será similar a "<carpeta de instalación>/fabric-samples/test-network". Aquí podemos utilizar el script network.sh, como se describe en [29], y seguir el resto de pasos para instalar nuestros chaincodes. De manera alternativa, los scripts startHyperledger, instalCCOne, instalCCTwo, updateOne y updateTwo pueden ser copiados al directorio y utilizados para agilizar significativamente el proceso.

El script startHyperledger.sh se encarga de limpiar posibles restos de una red iniciada con anterioridad y de iniciar la red. Además se encarga de iniciar el proceso de instalación de los chaincodes. Como parámetros toma primero la dirección relativa a la ubicación del chaincode (../../../chaincodePath, por ejemplo) y luego el nombre con el que se desea instalar.

Después se debe ejecutar instalCCOne.sh, que toma como parámetro el nombre de instalación, que debe ser el mismo que ha sido utilizado en paso anterior. Por consola se mostraran los IDs de todos los chaincodes disponibles.

InstalCCTwo.sh toma como parámetro el ID del chaincode que queremos instalar, que habrá sido mostrado en consola tras el paso anterior y el nombre de instalación, que debe ser el mismo que el

utilizado en los dos pasos anteriores. Si el proceso se ejecuta correctamente, la consola mostrará un mensaje con los chaincodes instalados.

La ejecución de estos scripts tendría una forma similar a la siguiente:
‘sudo sh <test-network-folder>/startHyperledger.sh <path to chaincode> <name for chaincode instalation ,ex: register>
sudo sh <test-network-folder>/installCCOne.sh <name>
sudo sh <test-network-folder>/installCCTwo.sh <ID given in terminal> <name>’

Cabe mencionar que la red de pruebas se inicia como un grupo de contenedores Docker enlazados en la red fabric_test. Cada nodo tiene su contenedor, y también tiene uno por cada chaincode instalado. Podemos utilizar sus IDs para ver los logs de ejecución de la red.

Cuando se implementan cambios en los chaincodes, éstos deben volver a ser instalados en la red. Una opción es volver a ejecutar el proceso descrito anteriormente, pero como veremos en el siguiente apartado, conlleva algo de trabajo de configuración manual para que los agentes vuelvan a poder conectarse con la red. Por ello, se han escrito scripts similares para actualizar los chaincodes.

updateOne.sh toma como parámetro la ubicación relativa del chaincode, el nombre de instalación, que debe ser el mismo que el proceso anterior para cada chaincode, y la versión. Es importante que la versión incremente en uno cada vez que se ejecuta el proceso de actualización, incluso si hay algún error, ya que la red es estricta con los números de secuencia de los intentos de actualización y no permitirá que se ejecute una actualización con un número equivocado. Como resultado se mostrará por consola los IDs de los chaincodes instalados.

UpdateTwo.sh toma como parámetros la ID del chaincode a actualizar, su nombre de instalación y su versión. Es importante que el ID sea el correspondiente a la nueva versión, y que tanto nombre como versión coincidan con los del paso anterior. Como resultado muestra un mensaje de validez.

Un ejemplo de su utilización en consola es:
‘sudo sh <test-network-folder>/updateOne.sh <path to chaincode> <name> <version>
sudo sh <test-network-folder>/updateTwo.sh <ID given in terminal> <name> <version>’

B.2 Inicialización del nodo / red IPFS

Una red IPFS es necesaria para almacenar el payload de una actualización. A efectos prácticos un nodo estándar y público es suficiente para llevar a cabo el proceso de registro de actualización para este TFG. No obstante, el entorno de producción debe estar en una red privada por diseño, ya que un CID filtrado permitiría a un atacante obtener la actualización.

Si no tenemos una red privada, podemos inicializar un nodo si tenemos docker instalado con el siguiente comando:

‘docker run -d --name ipfs_host --network fabric_test -e IPFS_PROFILE=server -v $ipfs_staging:/export -v $ipfs_data:/data/ipfs -p 4001:4001 -p 4001:4001/udp -p 127.0.0.1:8080:8080 -p 127.0.0.1:5001:5001 ipfs/kubo:latest’

También se ha incluido en el repositorio del proyecto una carpeta IPFS con los archivos necesarios para ejecutar una red privada construida a partir de contenedores Docker. Dicha red se conecta con la red fabric_test que agrupa todos los elementos del proyecto.

La red está compuesta por tres nodos que forman un clúster. Un clúster es una estructura que permite compartir información de manera más eficiente y propagar la fijación de contenidos. Está compuesta por seis contenedores Docker. Aunque en el proyecto no se entra en mucho detalle sobre el uso y funcionamiento de la red, es conveniente contar con la red para posibles futuras fases posteriores al TFG.

Para iniciarla, después de iniciar la red Hyperledger Fabric, basta con ejecutar el script build.sh.

B.3 Inicialización del Agente de Registro

Con la red iniciada, podemos inicializar el agente de registro. Podemos hacerlo de manera local o mediante el uso de una imagen Docker. En ambos casos, sin embargo, es necesario realizar algunas configuraciones.

Primero debemos copiar los archivos de conexión de la red blockchain a la carpeta src/config del agente de registro. Una vez desplegada la red de prueba se encuentran en la carpeta test-network/organizations/peerOrganizations/org1.example.com/connection-org1.json.

Una vez copiada, debemos obtener la dirección gateway de la red blockchain. Para ello se ejecuta docker ps para ver las imagenes en ejecución y se busca la correspondiente a peer0.or1.example.com. Con su ID ejecutamos docker inspect <ID>, lo que nos revela la dirección. Sustituimos localhost por la dirección obtenida en el archivo de configuración, que aparece en más de una ocasión.

Con la conexión configurada, es necesario crear los certificados de conexión. Si la aplicación se ha ejecutado con una red que ya ha sido eliminada, es necesario borrar los contenidos de src/wallet, que contendrán los certificados para una red previa y que no serán válidos para la inicializada actualmente.

Debemos crear ahora los certificados nuevos. Si se utiliza Docker, este paso puede omitirse ya que se ejecutará automáticamente al crear la imagen. En la terminal nos situamos en la carpeta src/services, y ejecutamos enrollAdmin.js y después registerUser.js. Si el proceso de inicialización se ha ejecutado correctamente, en la carpeta src/wallet se habran creado dos los archivos admin.id y RegisterAgentUser.id, que contienen las claves necesarias para que la aplicación se conected con la red blockchain.

En el archivo config/config.json debemos indicar la dirección del nodo IPFS a utilizar.

Finalmente, podemos ejecutar la aplicación. En el caso local, con el comando npm start. Si se utiliza docker, se puede utilizar build.sh, presente en src/scripts. Este script crea la imagen y la ejecuta de manera que el contenedor se conecte con la red blockchain.

Una vez iniciada la ejecución, podemos realizar pruebas funcionales a través de Swagger accediendo a 127.0.0.1:3001 en un navegador.

B.4 Inicialización del Agente Autor

Una vez inicializado el agente de registro, podemos ejecutar el agente autor.

Antes de ejecutar la aplicación, debemos añadir la dirección del agente de registro al archivo de configuración en src/config en el campo requestHost, así como su puerto, que por defecto es el 3001.

Si utilizamos la aplicación de manera local, hay que asegurarse de que hay un proceso de mongodb accesible mediante el puerto 27017, ya que la conexión se hace mediante ‘mongodb://mongo:27017‘.

Para iniciar la aplicación de manera local, ejecutamos npm start. Para ejecutarla mediante Docker, ejecutamos el script src/scripts/build.sh. Este script construye la imagen y la conecta con una imagen de mongodb utilizado un archivo dockerfile.

B.5 Inicialización del agente de recuperación

Después de ejecutar los pasos siguientes podemos inicializar el agente de recuperación. También es posible hacerlo antes que los agentes de registro y de autor, pero no habría nada que recuperar de la red en ese caso.

El proceso es similar a los casos anteriores. El agente atiende al puerto 3002 y se puede inicializar con el comando build.sh disponible en la misma carpeta.

Al igual que con el agente de registro, hay que configurar la conexión a la red blockchain e IPFS. pueden seguirse los mismos pasos, ya que la conexión se efectua exactamente de la misma manera en ambos agentes.

B.5 Inicialización de las apps

Las apps también pueden ser inicializadas con el comando build.sh, disponible en src/scripts como en los casos anteriores. Necesitan que los agentes hayan sido inicializados con anterioridad, ya que dependen de ellos para ejecutar sus funcionalidades.

Anexo C - Tabla de desviaciones

Paquete	Tarea	Dedicación Estimada(h)	Tiempo Real (h)
Investigación (I)	Análisis del problema a solucionar	10	5
	Análisis de las soluciones propuestas con anterioridad	10	5
	Investigación sobre Blockchain	10	6
	Estudio de alternativas de emulación.	8	2
	Elaboración de comparativa de alternativas.	2	0
	Estudio sobre el funcionamiento de Hyperledger Fabric	10	8
	Estudio sobre el funcionamiento de IPFS	5	5
	Estudio del funcionamiento de Docker	2	1
	Estudio del funcionamiento de Swagger	2	2
	Estudio del funcionamiento de NodeJS	3	4
I Subtotal : 62h			38h
Diseño (D)	Diseño de arquitectura de red blockchain y almacenamiento	10	8
	Diseño del flujo general	2	2
	Diseño del flujo de registro de autores	2	3
	Diseño del flujo de registro de actualizaciones	2	3
	Diseño del flujo de recuperación de actualizaciones	2	3
	Diseño de estructuras	6	4
	Diseño de agente autor	5	3
	Diseño de agente de registro	5	5

	Diseño de agente de recuperación	5	3
	Diseño de smart contract de registro	10	4
	Diseño de smart contract de recuperación	10	4
	Diseño de modelo de emulación de dispositivos	4	3
	Diseño de aplicación de autor	4	2
D Subtotal: 67h			47h
Implementación (IMP)	Implementación del agente autor	15	20
	Implementación del agente de registro	15	22
	Implementación del agente de recuperación	15	16
	Implementación del contrato de registro	20	18
	Implementación del contrato de recuperación	20	16
	Documentación de elementos implementados	15	25
	Implementación de apps *	-	20
IMP Subtotal: 100h			137h
Pruebas(P)	Prueba de funcionalidades del agente del autor	3	15
	Prueba de funcionalidades del agente de registro	3	3
	Prueba de funcionalidades del agente de recuperación	3	4
	Prueba de funcionalidades del contrato de registro	4	6
	Prueba de funcionalidades del contrato de recuperación	4	6
	Pruebas de rendimiento de registro *	-	20

	pruebas de rendimiento de recuperación *	-	12
P Subtotal 17h			66h
Gestión (G)	Planificación de tareas a realizar según los requisitos de la aplicación	10	11
	Captura de requisitos funcionales	1	1
	Captura de requisitos no funcionales	1	1
	Captura de requisitos adicionales	1	1
	Establecimiento de objetivos principales	1	1
	Establecimiento de objetivos secundarios	1	1
	Seguimiento y control	10	6
	Reuniones	10	25
G Subtotal 35h			47h
Trabajo Académico (TA)	Elaboración de la memoria	60	80
	Defensa del proyecto	15	16
TA Subtotal 75h			96h
Elaboración de artículo académico *			50h
Total: 356 Horas			**TOTAL 484h**

*Los elementos marcados con un asterisco no formaban parte de la planificación inicial, sino que fueron añadidos durante la elaboración del proyecto.