Verificación de la información en el seguimiento del ciclo de vida de dispositivos TIC

Javier Cano Esteban

Sistemas de blockchain

Máster en Ciberseguridad y Privacidad

Nombre del tutor/a de TF:
Leandro Navarro Moldes
Alberto Ballesteros Rodríguez

Nombre del/de la PRA:
Victor Garcia Font

9 de enero de 2024

Universitat Oberta de Catalunya

Ficha Del Trabajo Final

Título del trabajo:	Verificación de la información en el seguimiento del ciclo de vida de dispositivos TIC
Nombre del autor/a:	Javier Cano Esteban
Nombre del tutor/a de TF:	Leandro Navarro Moldes Alberto Ballesteros Rodríguez
Nombre del/de la PRA:	Victor Garcia Font
Fecha de entrega:	9 de enero de 2024
Titulación o programa:	Máster en Ciberseguridad y Privacidad
Área del trabajo final:	Sistemas de blockchain
Idioma del trabajo:	Castellano
Palabras clave:	blockchain, pasaporte digital, credenciales verificables

Resumen del trabajo

Este proyecto aborda la implementación de mecanismos de verificación en la trazabilidad del ciclo de vida de dispositivos de Tecnologías de la Información y Comunicación (TIC) utilizando tecnologías de Ledger Distribuido (DLT), específicamente Ethereum, y credenciales verificables. El enfoque principal se centra en la aplicación de estos procesos de verificación a un buscador de Pasaportes Digitales de Producto (DPPs). La veracidad de los eventos relacionados con los dispositivos se valida criptográficamente mediante hashes, asegurando la integridad de la información a lo largo de su ciclo de vida.

Además, se implementa la verificación de la identidad y el rol de los actores del sistema mediante el uso de credenciales verificables. Estas credenciales proporcionan un marco sólido para establecer la autenticidad de los participantes en el ecosistema de trazabilidad. Al aplicar estas tecnologías, se garantiza un registro inmutable y transparente de los eventos, mejorando la confianza y la seguridad en la gestión de dispositivos TIC.

En conclusión, este proyecto logra sus objetivos al incorporar de manera efectiva mecanismos de verificación en la trazabilidad de dispositivos TIC, brindando una solución integral que asegura la autenticidad de los eventos y la identidad de los actores involucrados en el ciclo de vida de estos dispositivos.

Abstract

This project addresses the implementation of verification mechanisms in the traceability of the life cycle of Information and Communication Technology (ICT) devices using Distributed Ledger Technologies (DLT), specifically Ethereum, and verifiable credentials. The primary focus is on applying these verification processes to a Digital Product Passport (DPP) search engine. The accuracy of events related to devices is cryptographically validated through hashes, ensuring the integrity of information throughout its life cycle.

Furthermore, identity and role verification of system actors are implemented using verifiable credentials. These credentials provide a robust framework for establishing the authenticity of participants in the traceability ecosystem. By applying these technologies, an immutable and transparent record of events is guaranteed, enhancing trust and security in the management of ICT devices.

In conclusion, this project successfully achieves its objectives by effectively incorporating verification mechanisms into the traceability of ICT devices, providing a comprehensive solution that ensures the authenticity of events and the identity of actors involved in the life cycle of these devices.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En la era de la información y la conectividad, los dispositivos de Tecnologías de la Información y Comunicación (TIC) desempeñan un papel fundamental en nuestra vida cotidiana y en el funcionamiento de organizaciones en todo el mundo. Estos dispositivos, que abarcan desde ordenadores y teléfonos inteligentes hasta servidores y sensores IoT, son esenciales para la gestión de datos, la comunicación y la automatización de procesos en diversos sectores. Sin embargo, su adopción masiva y su proliferación han generado un creciente desafío social y medioambiental.

Como parte del Grupo de Sistemas Distribuidos (DSG) de la Universitat Politècnica de Catalunya (UPC) este trabajo es parte del desarrollo de un sistema adecuado para acompañar con información precisa al ciclo de vida de estos dispositivos. Este es un proceso continuo y dinámico, idealmente circular aunque habitualmente lineal, que comprende varias etapas, desde la fabricación y distribución hasta la adquisición, uso, mantenimiento, entrada en el mercado de segunda mano y eventual desecho o reciclaje. La gestión clara e informada de este ciclo de vida es crucial para obtener un uso óptimo de los recursos que ayude en la reducción de impactos ambientales negativos. Sin embargo, en un sistema con múltiples actores, asegurar la calidad, integridad y verificabilidad de la información relacionada con estos dispositivos se convierte en un desafío crucial.

En este contexto, las tecnologías blockchain han surgido como una solución prometedora para abordar los problemas de integridad, trazabilidad y verificabilidad de la información. Sin embargo, su aplicación en la gestión del ciclo de vida de los dispositivos ICT plantea desafíos únicos, particularmente en lo que respecta a la verificación de los datos generados.

Se tratará entonces, en este trabajo, de abordar algunos de estos desafíos, como es el de disponer de la capacidad de asegurar la integridad y momento de generación de los datos, o comprobar que sus orígenes son una fuente de información fiable.

1.1. Contexto y justificación del trabajo

Actualmente, no existe una regulación exhaustiva que prohíba a las empresas realizar afirmaciones medioambientales sin la presentación de pruebas concretas y verificables de sus productos. Esta falta de regulación específica ha dado lugar a un entorno en el que las afirmaciones sobre la sostenibilidad y el impacto ambiental de los productos pueden carecer de la transparencia y la rigurosidad necesarias para respaldarlas de manera creíble. Sin embargo, la Comisión Europea (CE) está desarrollando una nueva regulación sobre los llamados "Green claims" [1] y el concepto y regulación del pasaporte digital de productos [2]. Esto forzará a las empresas a tener que hacer que estas afirmaciones sobre sus productos sean contrastables con datos. En este caso, se considera que el sistema desarrollado durante la investigación del DSG puede ser de gran ayuda en el caso de los dispositivos TIC, proporcionando un mecanismo que hace los datos fiables. Particularmente, trazar eventos y documentos acreditativos de acciones sobre los diferentes dispositivos durante su ciclo de vida permite agregar datos sobre productos individuales, como por ejemplo su duración media o porcentaje de productos reciclados correctamente, que podrían ser la base para alguno de estos "Green claims".

Se realiza el trabajo con el soporte del proyecto NGI Search [3], que busca nuevas formas de buscar y descubrir información en internet. Se parte de un prototipo previamente desarrollado por el mismo grupo de investigación, más o menos sencillo, implementado usando Smart Contracts en Ethereum, que registra información sobre el ciclo de vida de un conjunto de dispositivos. El objetivo actual es desarrollar mecanismos para encontrar estos datos y proporcionar procesos fiables de verificación de los mismos.

En estos procesos, se busca mantener la privacidad de los datos, escribiendo en la cadena solo lo necesario para su verificación. Esto tendría poco sentido si además no se asegurara su procedencia. Por lo tanto se quiere poder identificar también de forma segura la cualificación de los actores participantes.

1.2. Estado del arte

Si se piensa en una manera de implementar el sistema de trazabilidad mencionado, lo clásico sería la creación de una aplicación con una base de datos controlada por cierta entidad. Los usuarios estarían autorizados por la entidad para crear o modificar datos y sería una implementación sencilla y a primera vista fiable. A esto se le denominaría un sistema centralizado. Sin embargo, esta clase de sistema presenta ciertas desventajas que se consideran críticas en un sistema de trazabilidad. Por ejemplo, si la entidad responsable desaparece, se podrían llegar a perder todos los datos, si es atacada por un actor malicioso, todos los datos podrían estar comprometidos, o en general, centralizar la autoridad en un punto en un sistema donde intervienen tantos actores, puede no ser la mejor idea.

Hoy en día existen alternativas en forma de sistemas distribuidos. En los sistemas distribuidos, esta responsabilidad se puede descentralizar, y la autoridad puede estar distribuida en una red de actores que forman relaciones de confianza entre ellos. En el núcleo de estos sistemas existen a menudo las DLT (Decentralized Ledger Technology), que actúan como un libro de cuentas compartido por todos los actores. Estas tecnologías, principalmente, aportan implícitamente mecanismos de veracidad y no repudio, actuando de una manera similar a un notario digital. Además, solucionan en gran medida los problemas comentados, haciendo que si uno de los actores cae, el sistema pueda seguir, o que los ataques sean más difíciles (ya que normalmente se debe controlar una mayoría de elementos del sistema para un ataque efectivo).

Una característica interesante que ofrecen estos sistemas, es la capacidad de contar con una identidad descentralizada. Esta identidad es controlada por el usuario y para definir relaciones de confianza entre estas identidades, entran los sistemas de credenciales verificables. En el contexto de trazabilidad, estas credenciales, emitidas de usuario a usuario, se pueden utilizar para identificar la capacidad de cada uno a generar o a acceder a información en el sistema. Esto acabará estando en el centro de este proyecto, al diseñar los procesos de verificación de procedencia de la información.

El esfuerzo de estandarización de estos sistemas de credenciales lo publica el World Wide Web Consortium (W3C) [4] y se considera que su aplicación en un caso de uso como este aporta a darle validez como tecnología emergente.

1.3. Objetivos del trabajo

El objetivo principal del trabajo es probar la viabilidad del desarrollo de los procesos de verificación de la información sobre el ciclo de vida de los productos y la identificación y verificación de los actores que escriben cada dato. Esto quiere decir que se debe asegurar que los datos escritos en un cierto momento no se puedan modificar y sean recuperables, que se pueden relacionar con un actor y que se pueden verificar la identidad y permisos o cualificación de este actor para generar estos datos.

El resultado final pretende contar con un prototipo que permita encontrar la información relacionada con uno o varios dispositivos y que aplique estos procesos para darle fiabilidad.

En concreto, la lista de objetivos será:

- Valorar diferentes mecanismos de verificación y recuperación de la información a partir de resúmenes (hash) registrados en la cadena.
- Valorar diferentes marcos de implementación del estándar de credenciales verificables definido por el World Wide Web Consortium (W3C) [4].
- Implementar los procesos de verificación y el sistema de credenciales verificables.

- Desarrollar y validar un buscador de la información prototipo que incluya esta implementación.

1.4. Impacto en sostenibilidad, ético-social y de diversidad

En la dimensión de sostenibilidad, el trabajo tiene un impacto positivo, concretamente en los ODS 11 (Ciudades y comunidades sostenibles) y 12 (Producción y consumo responsables). La trazabilidad de dispositivos TIC puede proporcionar información sobre el impacto medioambiental de estos productos, a la vez que impulsa la economía circular, dando información esencial sobre productos de segunda mano a un potencial comprador o facilitando el trabajo de un reciclador. Impulsando la economía circular se reduce la fabricación de nuevos productos y se impulsa la reutilización y el reciclaje más eficiente de estos.

Como punto negativo, se puede argumentar que las infraestructuras necesarias para apoyar este sistema pueden llegar a consumir demasiados recursos. Se lleva viendo durante años como redes blockchain basadas en "Proof of Work" consumen grandes cantidades de energía. No obstante, existen redes con el objetivo de lograr un bajo consumo, incluso diseñadas para dispositivos IoT, como la desarrollada por la IOTA Foundation [5].

En cuanto a la dimensión de comportamiento ético y responsabilidad social, se podría argumentar que indirectamente, el trabajo tiene un impacto positivo en el ODS 1 (Fin de la pobreza), ya que el crecimiento de un mercado circular puede hacer más asequibles este tipo de productos que son cada vez más esenciales.

Finalmente, en la dimensión de diversidad, género y derechos humanos, al igual que en el anterior párrafo, se puede argumentar que indirectamente influye de manera positiva en el ODS 10 (Reducción de las desigualdades), dando acceso a más gente a dispositivos TIC y a los servicios y beneficios que proporcionan.

1.5. Enfoque y método seguido

Tal como se ha comentado en el apartado 1.2, el objetivo es utilizar un sistema ya desarrollado y añadirle componentes que lo mejoran. Al final, después de un estudio sobre los mecanismos que permiten lograr añadir estos componentes, se pone como objetivo el desarrollo de un nuevo prototipo que los implemente.

Todo el desarrollo se hace bajo el marco del proyecto NGI Search, para el que se desarrolla un buscador de información sobre el ciclo de vida de dispositivos TIC.

Durante el desarrollo del trabajo, se introducirán los conceptos ya desarrollados anteriormente, se presentará el punto de partida del desarrollo, se estudiarán las nuevas técnicas a aplicar y se hará el desarrollo y validación del prototipo enfocándose en los procesos de verificación.

1.6. Planificación del trabajo

Dado que el trabajo se realiza principalmente por solo una persona, se planificará con una metodología ágil en varios sprints, coincidiendo con las fechas de entrega de las PEC y los objetivos propuestos.

- Los primeros dos sprints, tratarán de valorar y explicar los mecanismos de verificación y credenciales verificables. Coinciden con los dos primeros objetivos, entre el 11 de octubre y el 7 de noviembre.
- El tercer sprint tratará de implementar la verificación de información sobre eventos del ciclo de vida de los dispositivos en el prototipo final. Coincide con el tercer objetivo, entre el 8 de noviembre y el 5 de diciembre.
- El último sprint tratará de implementar la verificación de actores para poner cierre al prototipo final. Coincide con el cuarto objetivo, entre el 6 de diciembre y el 9 de enero.

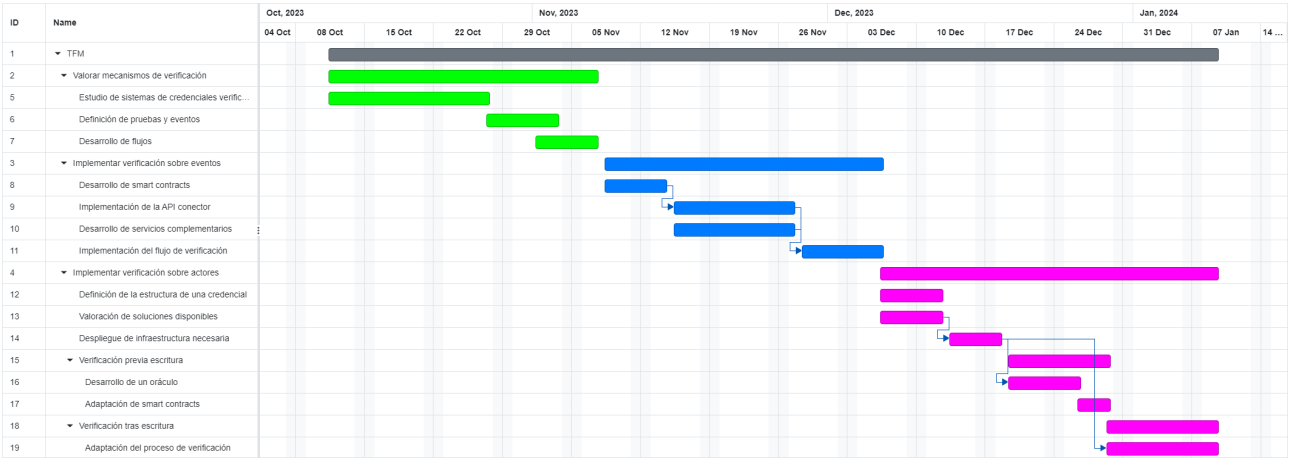

Figura 1.1: Diagrama de Gantt.

Para dar soporte a estas etapas, se producirá un backlog en una herramienta de tablero Kanban.

1.7. Breve sumario de productos obtenidos

Diseño y desarrollo de los procesos de verificación de la información sobre el ciclo de vida de productos TIC y la identificación y verificación de los actores que escriben cada dato.

La verificación de la información sobre el ciclo de vida de productos TIC se realizará mediante comparación de hashes escritos previamente en un registro verificable. El resultado final será un botón en el buscador prototipo que inicie el proceso de verificación.

La verificación de actores se llevará a cabo comprobando que tienen una credencial verificable que los autoriza con cierto rol. Para esto hará falta una infraestructura que acomode un sistema de identidad y credenciales verificables. El resultado será también un botón que inicie el proceso de verificación.

1.8. Breve descripción de los otros capítulos de la memoria

Esta memoria se organiza en tres grandes capítulos:

- Materiales - Se exponen los conceptos base necesarios para entender el proyecto y se detalla de forma concisa el punto del que se parte, en relación al trabajo realizado antes de este proyecto.
- Diseño de métodos - Se explican posibles diseños que se pueden utilizar para lograr los objetivos del proyecto. Se trata también de elegir cuál de ellos es más adecuado.
- Arquitectura e implementación - Se detalla la implementación del diseño elegido en el sistema. En ocasiones se argumenta si el diseño ha de sufrir algún cambio respecto al inicio.

Adicionalmente, se presenta un capítulo final de resultados y conclusiones, donde se valora si se han cumplido los objetivos y se proponen varias vías de trabajo futuro.

Capítulo 2

Materiales

2.1. Conceptos base

Se utiliza como base el sistema previamente desarrollado por el DSG. Este sistema, como ya se ha mencionado anteriormente, registra información sobre el ciclo de vida de productos TIC. Para entenderlo, se han de introducir diversos conceptos.

2.1.1. Identificadores de dispositivos

Chassis ID "CHID"

Cada dispositivo tiene un identificador único. Este identificador se obtiene a partir de la exploración de un conjunto mínimo, y que no cambia durante su vida útil, de sus componentes ejecutando un software específico de análisis de hardware.

```
1 fef01bdcdff0672d9a9c3ad820f24e935c371d4bfbbe1817bbe12c73deea405a
```

Figura 2.1: Ejemplo de CHID.

Product Hardware ID "PHID"

Al Chassis ID (a partir de ahora "CHID"), se le concatena otro, que identifica la composición actual de componentes principales que no forman parte del mínimo (para producir el CHID).

```
1 a8d997ff2517d48573d08555028c5dc8825f39759af8a69f6de9f55a73d282e2
```

Figura 2.2: Ejemplo de PHID.

2.1.2. Digital Product Passport "DPP"

La concatenación de CHID y el Product Hardware ID (a partir de ahora "PHID"), forma el identificador de un DPP (Digital Product Passport).

```
1 fef01bdcdff0672d9a9c3ad820f24e935c371d4bfbbe1817bbe12c73deea405a:a8d997ff2517d48573d08555028c5
    dc8825f39759af8a69f6de9f55a73d282e2
```

Figura 2.3: Ejemplo del identificador de un DPP.

Los DPP, son conjuntos de datos que aportan información sobre un producto. La Comisión Europea (CE) los ve como un requisito para hacer que los productos sean más sostenibles, reparables y circulares [6].

En el contexto de este proyecto, un DPP contiene información sobre el dispositivo. Esta información, que puede abarcar desde materiales y sus componentes hasta enlaces a manuales de reparación, contiene eventos que han ocurrido en la vida del dispositivo (mientras tenía una configuración concreta). Cada uno de estos eventos ha de ir acompañado de una prueba que pueda verificar la veracidad de esa información y el evento de incorporación asociado.

Cada dispositivo acabará teniendo uno o más de estos DPP representando la configuración de su hardware a lo largo de su vida.

En resumen, un dispositivo:

- Se identifica con su CHID durante toda su vida útil.
- Contará con uno o más DPP, dependiendo de si su configuración cambia en el tiempo. El identificador de un DPP será "CHID:PHID".

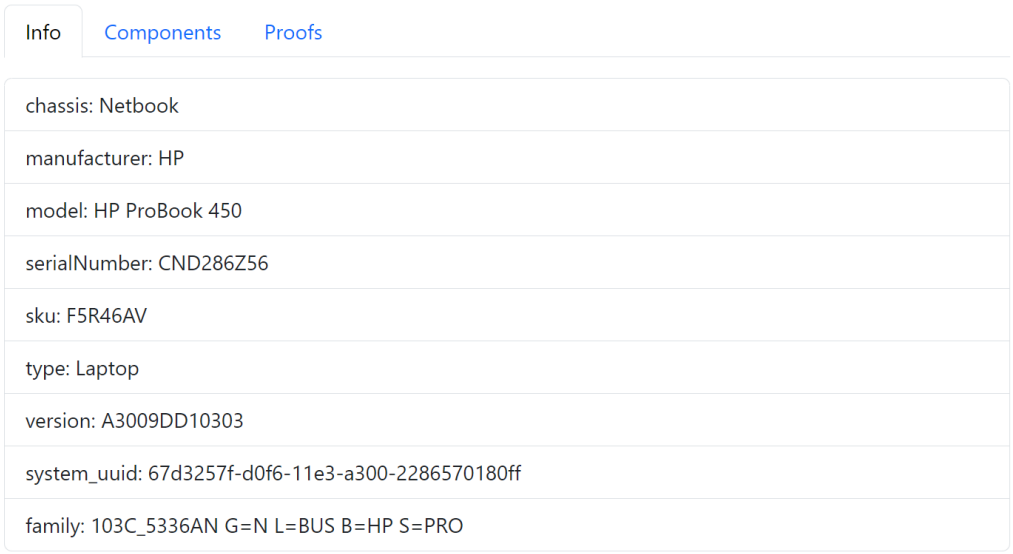

Figura 2.4: Ejemplo de DPP.

2.1.3. Identificador descentralizado "DID"

El W3C define una especificación para los llamados DID [7]. Estos identificadores han de poder ser únicos y generados sin una entidad central que los maneje. Todo DID, además, forma parte de un método, que define cómo se genera su identificador y cómo se interactúa con él, y tiene un documento asignado, que describe algunos metadatos sobre el identificador.

En este proyecto, se pueden convertir los CHID en DID abarcándolos en un método definido por el DSG, llamado "ereuse". Por lo tanto, un DID en este método tiene la estructura:

```
1 did:ereuse:CHID
```

Figura 2.5: Estructura del DID de un dispositivo.

Como ejemplo del CHID de la figura 2.1:

```
1 did:ereuse:fef01bdcdff0672d9a9c3ad820f24e935c371d4bfbbe1817bbe12c73deea405a
```

Figura 2.6: Ejemplo de DID de un dispositivo.

Su documento, obtenible a través de smart contracts, como se detallará más adelante, contiene referencias necesarias para el funcionamiento del sistema.

2.1.4. DLT

Una DLT (Tecnología de Ledger Distribuído) trata de la implementación de una estructura de datos distribuída, repartida en varios nodos, describiendo mediante transacciones ordenadas, generadas y firmadas por los usuarios, sus estados actual y anteriores por los que ha pasado. Un ejemplo son las llamadas blockchains, que agrupan estas transacciones en bloques de datos e incluyen el resumen (hash) del bloque anterior.

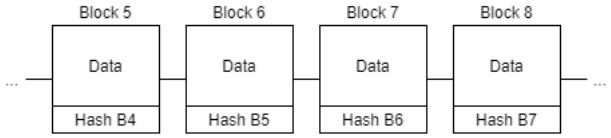

Figura 2.7: Esquema de bloques de una blockchain.

Si se modifica cualquier bloque, es fácil de detectar ya que no concuerda con el hash guardado en el siguiente bloque, haciendo que para modificar la información guardada en un bloque, se deban modificar todos los siguientes bloques en concordancia. Si se cuenta con que el sistema es distribuido y todos los nodos están sincronizados mediante un proceso de consenso, la estructura tiene las siguientes propiedades:

- Cualquier dato escrito en la cadena no se puede borrar.
- Cualquier escritura es verificable por cualquiera, ya que los datos son accesibles por cualquier nodo.

En el punto de partida de este proyecto, el sistema corre sobre una red blockchain Ethereum permisionada, donde solo pueden participar actores invitados y fiables.

2.1.5. Smart Contracts

Las redes Ethereum, mediante la implementación de una máquina virtual distribuida, cuentan con la posibilidad de almacenar piezas de código con las que los usuarios pueden interactuar. Denominados smart contracts, definen una lógica transparente a transacciones efectuadas en esta red. El sistema inicial hace uso de esta capacidad. En concreto, estos smart contracts definen datos de manera estructurada, lo cual permite trazar el ciclo de vida de los dispositivos.

Se utiliza la red Ethereum únicamente como registro verificable para guardar pruebas sobre eventos. Esto quiere decir que en ningún momento se utiliza esta red como sistema de inventario que guarde información específica o detallada sobre dispositivos (más allá de sus identificadores). Para guardar estos detalles se utiliza un software de inventario de dispositivos

llamado DeviceHub [8], que se encarga de guardar información sobre los eventos de la vida de un dispositivo y de presentar la información correspondiente a un DPP (a través de su interfaz web).

2.1.6. Pruebas

Se llamará prueba a una estructura de datos definida por un smart contract, que representa un evento que ha ocurrido respecto a un dispositivo, y se relaciona con este mediante su DPP.

Una prueba registrada en la blockchain debe poder verificar que un evento sucedió en cierto momento sin registrar información específica de ese evento. Para lograr eso, se proponen dos requisitos:

- Cada prueba genera un conjunto de información llamado "documento", que se debe guardar en algún lugar fuera de la cadena.
- Se registra en la cadena un hash del documento y una marca de tiempo.

Dado que también se quiere identificar al actor para poder verificar que tenía un rol adecuado en el momento de generar cierto evento y prueba, se registra en la prueba el identificador del actor. Este identificador es implícito en una red Ethereum ya que cada actor cuenta con un par de claves que lo identifica. En este caso, se registrará la clave pública, también denominada como la "Ethereum address" del actor.

En el punto de partida, una prueba tiene la siguiente estructura:

```
{
    "chid": "f04d3b9143e8b18833fa6311c36d247e4a8e28f18c6d7ce32c172e92898c6770",
    "phid": "cc13b0529af33eb0f0c4c3f85ebb7d20979dd0b226f1cdea1049eae161eea428",
    "IssuerID": "0x87593c75406382016F5D502404dEb19a844d615A",
    "DocumentHash": "cc13b0529af33eb0f0c4c3f85ebb7d20979dd0b226f1cdea1049eae161eea428",
    "Type": "DPP_creation",
    "timestamp": 1698700179,
    "blockNumber": 31605523
}
```

Figura 2.8: Primer formato de prueba, en JSON.

2.1.7. Actores y roles

El sistema cuenta con actores con diferentes roles. Estos roles limitan las acciones posibles de cada actor, añadiendo como requisito que para verificar un evento, no solo se deba verificar

el contenido de su correspondiente prueba, sino que la prueba se ha de registrar por un actor con un rol adecuado. En este contexto, son los siguientes tres:

- Operator – Actor encargado de registrar dispositivos en el sistema y de publicar sus DPPs. En el mundo real puede ser por ejemplo un fabricante o un distribuidor.
- Witness – Se encarga de registrar eventos sobre dispositivos que ha observado, aportando siempre un documento testimonio que permite verificar el hecho. En el mundo real podría ser cualquier entidad dedicada a reparar o reciclar los dispositivos.
- Verifier – Este actor entra en juego cuando se quiere recopilar información publicada por los demás actores. Su función es la de recoger los datos y verificarlos. Se puede tratar de cualquier entidad de auditoría que vela por el buen uso del sistema o busca agregar datos con diferentes objetivos.

En forma de diagrama, realizado anteriormente por el DSG:

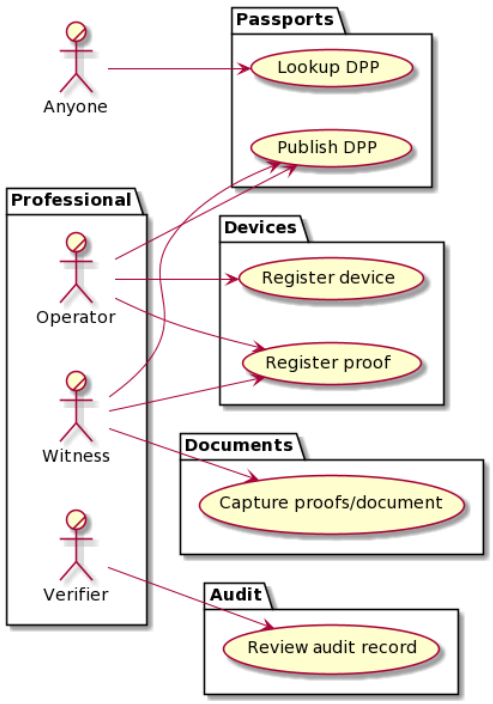

Figura 2.9: Diagrama de roles.

2.2. Descripción del sistema inicial

Todos los conceptos del apartado anterior forman parte de la arquitectura de partida:

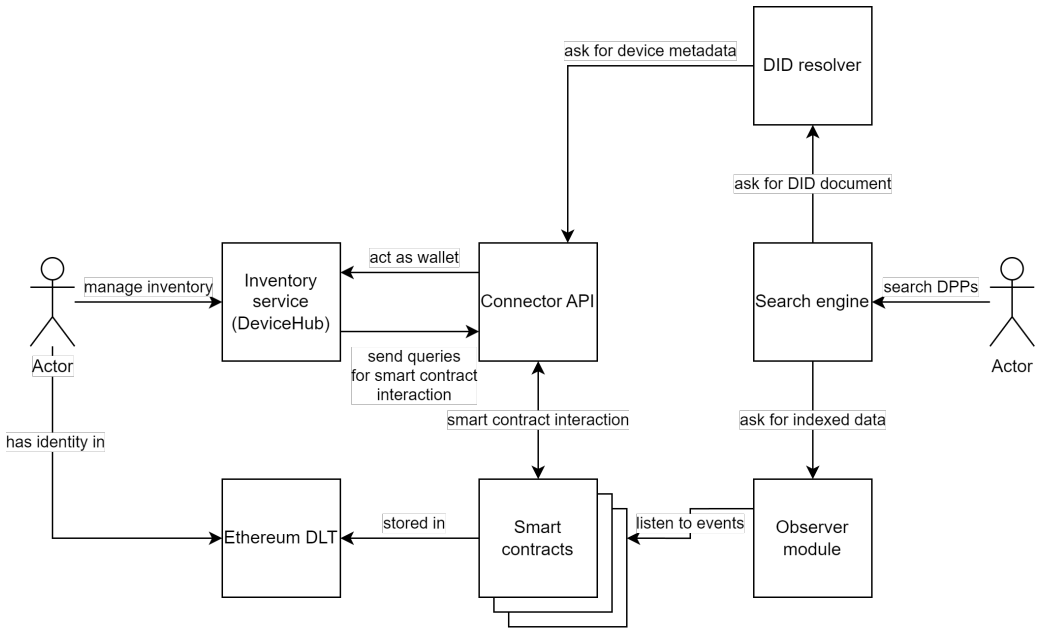

Figura 2.10: Arquitectura inicial del sistema.

Cada componente se describe y relaciona de la siguiente manera:

- Smart contracts – Dos tipos. Un smart contract del primer tipo actúa como factoría y despliega instancias del segundo tipo. Cada instancia del segundo tipo representa a un dispositivo.

  La interfaz de este contrato se detalla en el anexo A.3.

- API connector – En el centro, se encarga de abstraer las interfaces de los smart contracts mediante una API HTTP. Actúa como wallet de los usuarios, permitiendo el uso de smart contracts a terceros sin implementar nigún protocolo específico de la DLT.

- Sistema de inventario (DeviceHub) – Mencionado en la sección 2.1.5, es el punto de entrada de operators y witnesses, donde gestionan el inventario de sus dispositivos. Usa el API connector para conectar sus usuarios con la DLT.

- DID Resolver – Resuelve los DID de los dispositivos, retornando su documento. Este documento indica, por ejemplo, qué instancia de inventario registró el dispositivo o la dirección de su smart contract.

- Observer module – Observa los smart contracts para saber cuándo se registra un nuevo DPP e indexa esta información.

- Search engine – Con la ayuda del DID document y la información indexada por el observer, permite encontrar DPPs.

En resumen, toda la información (dispositivos, DPPs y eventos) parte de los sistemas de inventario, que son responsables de guardarla, operados por actores de los roles operator y witness. A través del API connector, se introduce solo lo necesario en la red Ethereum a partir de los Smart Contracts (CHIDs, identificadores de DPPs y pruebas).

Por el otro lado, actores anónimos o con el rol de verifier buscan información contenida en los DPP, como especificaciones sobre el dispositivo y su ciclo de vida (eventos). Dependiendo de si tienen un rol verificable, deberían poder ver más o menos información.

2.3. Resultado esperado

El resultado final debe ser un buscador de DPPs que presenta al usuario los eventos y pruebas escritas relacionadas con el DPP encontrado. Estas pruebas deben tener botones de verificación tanto de la prueba como del actor que la ha escrito. Al presionar estos botones se deben poner en marcha los procesos desarrollados en este proyecto.

La implementación del mecanismo de búsqueda de DPPs queda fuera del alcance de este proyecto.

Como punto de partida, se cuenta ya con el buscador y los botones "placeholder" para verificar la información.

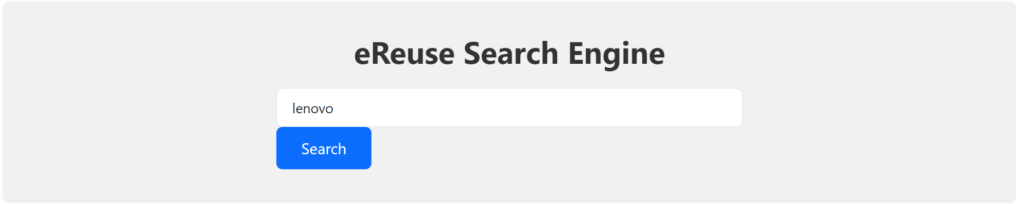

Figura 2.11: Pantalla inicial del buscador.

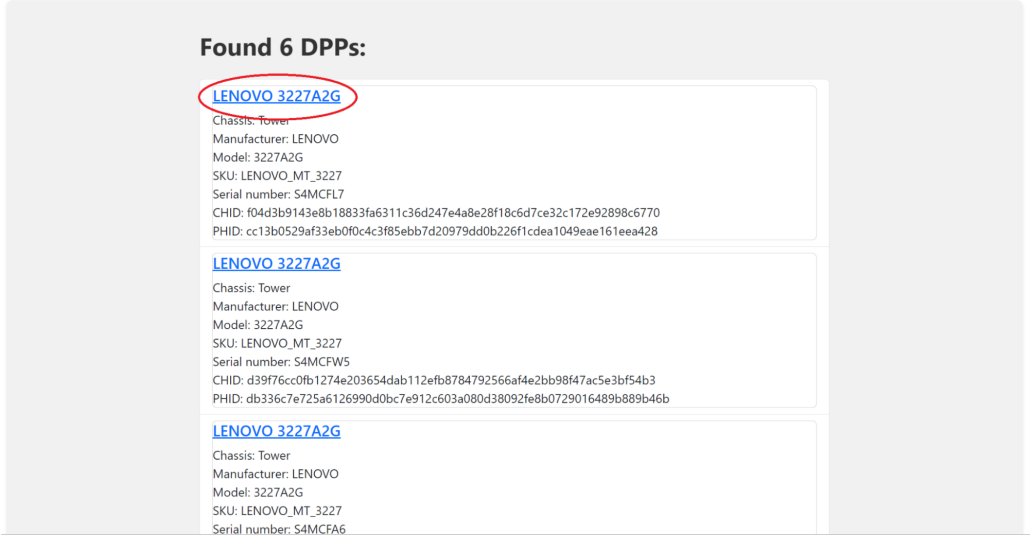

Figura 2.12: Resultados de la búsqueda.

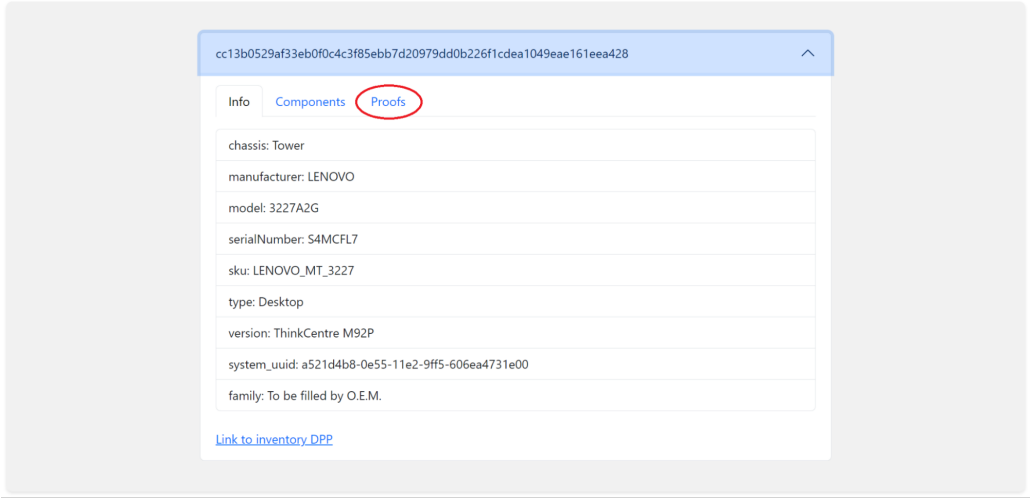

Figura 2.13: Detalles del DPP seleccionado.

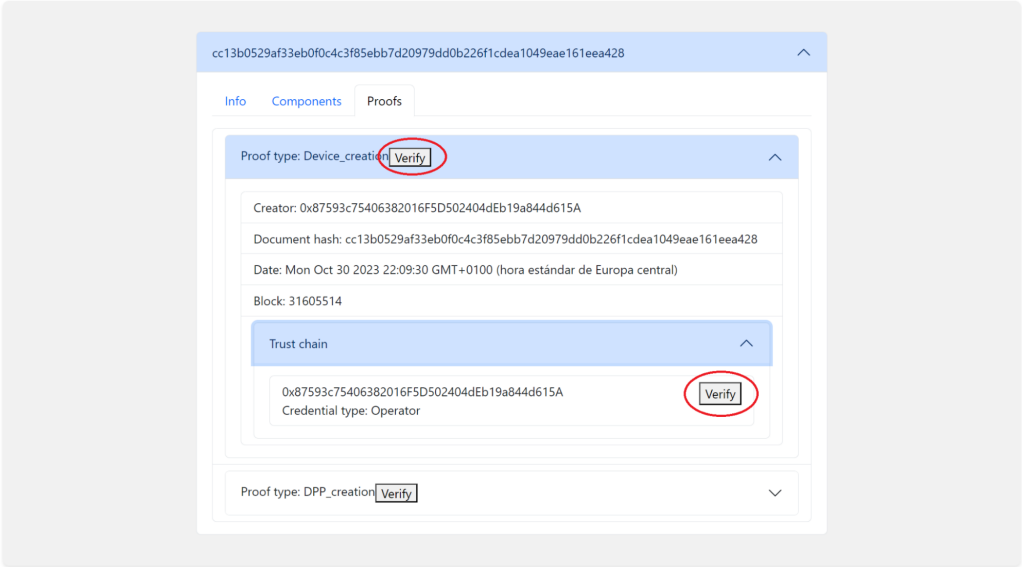

Figura 2.14: Vista de pruebas.

Capítulo 3

Diseño de métodos

Partiendo de lo detallado en el anterior capítulo, se trata de definir los procesos de verificación de eventos y procedencia a la vez que los subprocesos derivados de estos.

3.1. Contenido de una prueba y verificación de eventos

Se define el proceso de verificación del contenido de un evento como:

1. Encontrar el documento producido por el evento.
2. Producir el hash del documento.
3. Leer el contenido de la correspondiente prueba en la cadena.
4. Comparar el hash del documento con el hash guardado en la prueba.

Si los hashes coinciden, se puede asegurar que ese documento fue creado antes de la marca de tiempo guardada en la prueba.

Se ha de tener en cuenta, sin embargo, que los primeros dos puntos del proceso no son triviales. Mientras que encontrar una prueba a partir de un evento (con su correspondiente documento) puede hacerse gracias a los Smart Contracts, encontrar el documento a partir de una prueba no lo es. Se ha de guardar un puntero en la prueba que permita dirigir (resolución) al documento.

Sobre el segundo punto, para producir un hash comparable, se ha de saber el algoritmo de hash utilizado inicialmente para producir el hash de la prueba. Esto quiere decir que se

debe guardar un identificador de ese algoritmo en la prueba. Se utilizará la estandarización de nombres indicada por la Internet Assigned Numbers Authority (IANA) [9].

En resumen, para poder verificar un evento, una prueba registrada en la cadena ha de contener como mínimo:

- Hash del documento (producido por el evento).
- Identificador del algoritmo de hash utilizado.
- Puntero al documento (se detalla más adelante).
- Identificador del actor.
- Marca de tiempo.

Revisando el formato de partida de la figura 2.8, se añade el puntero (como "InventoryID") y el identificador del algoritmo de hash:

```
{
    "chid": "f04d3b9143e8b18833fa6311c36d247e4a8e28f18c6d7ce32c172e92898c6770",
    "phid": "cc13b0529af33eb0f0c4c3f85ebb7d20979dd0b226f1cdea1049eae161eea428",
    "IssuerID": "0x87593c75406382016F5D502404dEb19a844d615A",
    "InventoryID": "DH3",
    "DocumentHashAlgorithm": "sha3-256",
    "DocumentHash": "cc13b0529af33eb0f0c4c3f85ebb7d20979dd0b226f1cdea1049eae161eea428",
    "Type": "DPP_creation",
    "timestamp": 1698700179,
    "blockNumber": 31605523
}
```

Figura 3.1: Formato de prueba revisado, en JSON.

3.2. Búsqueda de un documento

Para encontrar el documento asociado a un evento, se deberá seguir el puntero guardado en una prueba. El mecanismo para seguir este puntero se puede definir de diversas maneras. En este caso se contemplan tres.

El primero podría ser guardar este puntero como un URL. Este URL llevaría a un servidor que contenga el documento. Esto tiene como problema que este tipo de enlaces apuntan a una localización específica en la web y por tanto pueden dejar de funcionar en algún momento. Dependen que el servidor siga vivo y alojado en la misma IP o nombre.

El segundo podría ser guardar el documento en un sistema descentralizado como IPFS (InterPlanetary File System) [10], que permite acceder a un contenido a partir del hash del

mismo. El puntero apuntaría entonces a ese sistema (a través de su nombre o uno o varios nodos) y con el hash del documento se podría llegar a recuperar. Como desventaja, este sistema no asegura que un documento se mantenga para siempre y necesita de más infraestructura para funcionar, dificultando además filtrar el acceso de usuarios no autorizados a documentos.

El tercero, que será el utilizado, pasa por dejar que una de las instancias de inventario DeviceHub (mencionado en la sección 2.1.5) guarde el documento. De esta manera, el puntero será un ID único que corresponde a esa instancia, y se mantendrá un sistema de resolución de estos IDs para llegar al inventario. Las ventajas respecto a las otras dos soluciones son que el mismo sistema que producirá los eventos los acabará guardando. Al igual, este sistema es el encargado de mostrar DPPs, así que tiene sentido que la información se haga disponible en ese punto. Guardando las IDs y manteniendo un sistema de resolución a URLs se evita además el punto flojo de la primera opción.

3.3. Verificación de la procedencia de una prueba

Como ya se ha mencionado, no basta con verificar el contenido de un evento. Se ha de verificar también que el actor productor de este evento esté autorizado para ello. De acuerdo a lo que ya se propuso anteriormente, se asignarán estos permisos o roles utilizando credenciales verificables.

No obstante, antes de hablar del sistema de credenciales, se identifican dos vertientes que tienen que ver con el momento de registrar una prueba en la cadena. ¿Se debe verificar al actor antes o después de escribir una prueba?

Si se identificara al actor antes, se podría impedir el registro de una prueba, haciendo que implícitamente, una prueba siempre provenga de un actor válido, lo cual requiere que los smart contracts puedan hacer la verificación del actor. Para que esto sea posible haría falta contar con uno de los dos siguientes casos:

- El sistema de credenciales se basa en la misma red Ethereum, dando a los smart contracts mecanismos para leerlo.

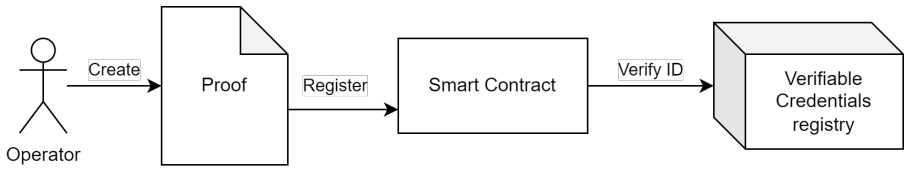

Figura 3.2: Registro de pruebas verificadas con credenciales verificables en la red Ethereum.

- El sistema de credenciales se basa en otra red, pero existe un oráculo que mantiene la red Ethereum al tanto de las credenciales de cada usuario a través de un smart contract.

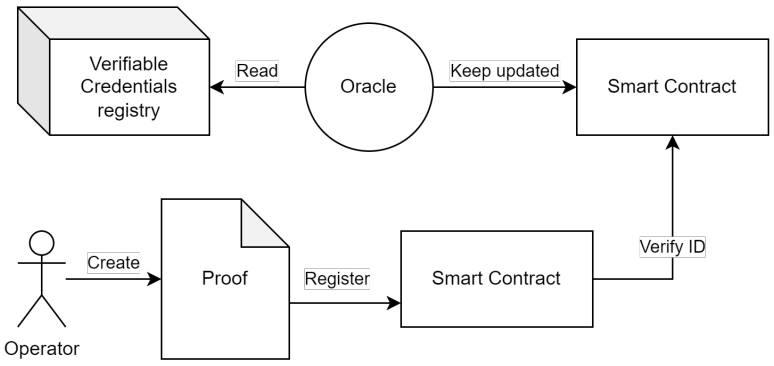

Figura 3.3: Registro de pruebas verificadas con credenciales verificables en la red Ethereum actualizadas por un oráculo.

Como desventajas, el primer punto limita la implementación de las credenciales a un solo entorno, y el segundo añade un punto de centralización en el oráculo que puede ser poco deseable, dependiendo de si viene empaquetado en la propia tecnología de credenciales verificables.

Si se identifica al actor después, al estilo de una estrategia de "lazy evaluation" [11], el proceso de verificación de un evento se complica un poco ya que se añade el proceso de verificación de credenciales a cada prueba. Además se introduce un riesgo de spam al sistema. Si se permite escribir a un actor sin una credencial adecuada, este puede ser malicioso y podría saturar la red de información inútil. Para solucionar esto, se añadiría como requisito que la red Ethereum deba ser permisionada o privada.

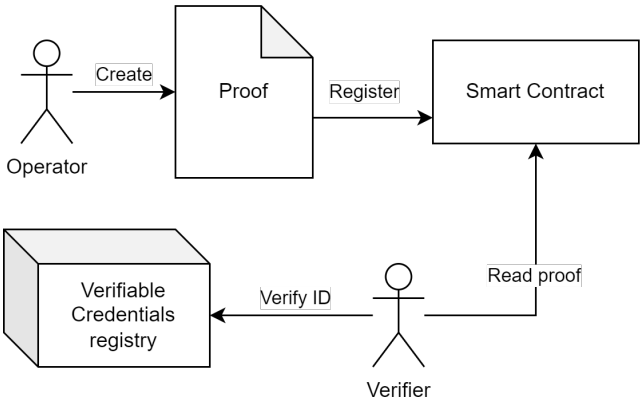

Figura 3.4: Registro de pruebas sin verificación previa.

3.4. Sistemas de credenciales verificables

3.4.1. Actores

En su definición más simple, una credencial es un conjunto de datos que dicen algo sobre un actor. En una credencial verificable estos datos vienen firmados por otro actor que puede afirmarlos, tiene autoridad para ello, y está registrado en un registro verificable. Así, un tercer actor, a través de las claves asimétricas e identidades encontradas en el registro verificable, lo puede verificar y se puede fiar. Estos actores son entonces de tres tipos:

- Holder - Actor poseedor de la credencial.
- Issuer - Actor que firma una credencial.
- Verifier - Actor que verifica que una credencial presentada por el holder la ha firmado un issuer.

Mientras que cualquier actor podría ser de uno o más de estos tipos, se definen en el mundo real específicamente como issuers los llamados "Trusted Accreditation Organisations" (TAO) y "Trusted Issuers" (TI). Las TAO y los TI juegan un papel similar, pudiendo firmar las credenciales, con la diferencia de que las TAO serían una raíz de confianza y los TI dispondrían de una acreditación, una credencial firmada por una TAO, para asegurar su capacidad o acreditación para emitir ciertas credenciales.

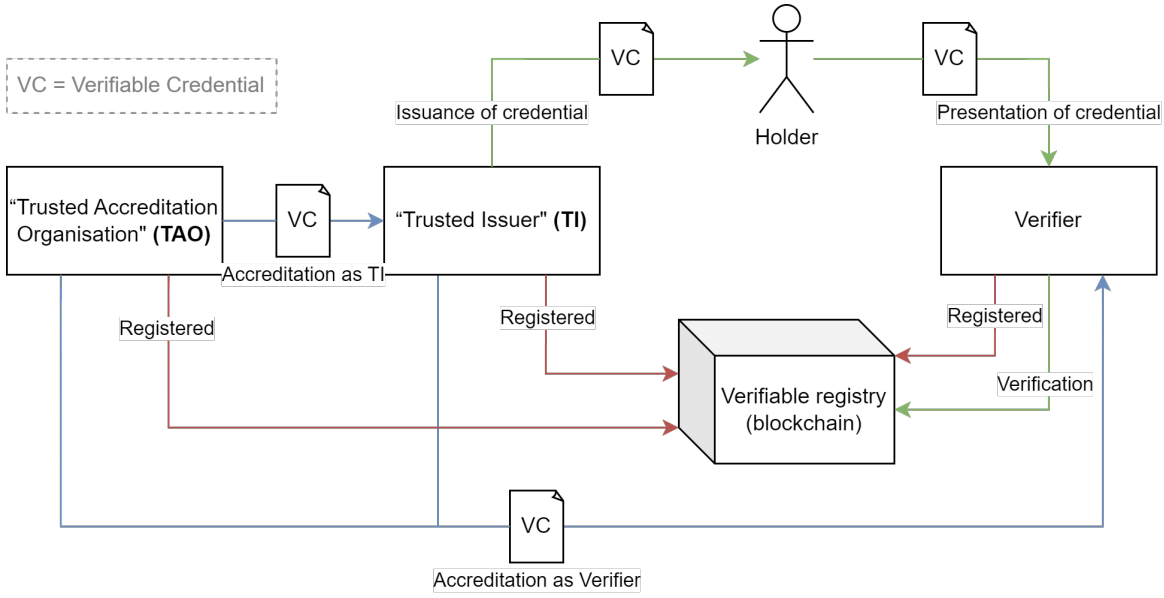

Figura 3.5: Flujo del sistema de credenciales verificables.

Así, hay que añadir a los roles de operator, witness y verifier de la figura 2.9, el rol de issuer, generando la siguiente tabla de correspondencia:

Tipo de actor en el sistema de credenciales verificables	Tipo de actor en el sistema de trazabilidad de dispositivos
Holder	Operator, Witness, Verifier, Issuer (solo TI)
Issuer (TAO y TI)	Issuer
Verifier	Verifier

Cuadro 3.1: Correspondencia entre actores del sistema de credenciales verificables y actores del sistema de trazabilidad.

En definitiva, los operators, witnesses, verifiers e issuers necesitan una credencial que defina su rol, emitida por un Issuer, para operar.

3.4.2. Soluciones

Para poder confiar en la identidad de un TAO o TI es necesario que estos guarden sus claves públicas en un registro verificable como una blockchain donde las partes que necesiten verificar puedan acceder. Este registro asegurará el acceso a esas claves en el tiempo, asegurando su confiabilidad y poniéndolas a disposición de un verifier.

A su misma vez, es necesario un framework para gestionar las credenciales y verificarlas.

Existen varios proyectos que incluyen ambos requisitos como Sovrin [12], EBSI [13] o el sistema desarrollado por la IOTA Foundation [14].

Existen también frameworks compatibles con diversos registros verificables como Veramo [15], Walt.id [16], SpruceID [17] o Hyperledger Aries [18].

El objetivo será utilizar alguno de estos proyectos que se ajuste respectivamente a los casos definidos en la sección 3.3. Uno con verificación de actores antes de escribir una prueba, y otro después.

Capítulo 4

Arquitectura e implementación

En este capítulo se detalla la implementación de los componentes relevantes a los procesos de verificación, extendiendo la arquitectura presentada en la sección 2.2.

4.1. Verificación de eventos

4.1.1. Componentes

Se detalla ahora la extensión o implementación de los componentes relevantes al proceso de verificación de eventos, extendiendo la arquitectura presentada en la sección 2.2.

Smart contracts

Los smart contracts en Ethereum se escriben en el lenguaje de programación Solidity [19]. Este lenguaje permite la creación de estructuras de datos y por lo tanto se usará para crear una estructura acorde a la prueba mostrada en la figura 3.1.

```
    struct GenericProofData {
        string chid;
        string phid;
        address issuerID;
        string inventoryID;
        string documentHashAlgorithm;
        string documentHash;
        string documentType;
        uint timestamp;
        uint blockNumber;
    }
```

Figura 4.1: Estructura de datos de una prueba en Solidity.

Por otro lado, se extienden también métodos para escribir y leer estas pruebas con los nuevos parámetros de "inventoryID" y "documentHashAlgorithm".

```
function generateGenericProof(string calldata _documentHashAlgorithm, string calldata
    _documentHash, string calldata _documentType, string calldata _inventoryID) public;

function getGenericProofs() public view returns (GenericProofData[] memory _data);
```

Figura 4.2: Cabeceras de los métodos para escribir y leer pruebas.

Servicio de resolución de nombres de inventarios

Este nuevo servicio se implementa como una API HTTP con dos funciones:

1. Devuelve una ID única a un servicio de inventario y guarda una relación entre esta ID y su URL.
2. Devuelve una URL dada una ID de un servicio inventario.

Se utiliza para averiguar la URL donde se encuentra el documento perteneciente a un evento. La ID del inventario se corresponde al campo "InventoryID" de la prueba (Figura 3.1).

API connector

Este componente, existente en el proyecto inicial, abstrae la interfaz de los smart contracts en llamadas HTTP para hacer la conexión de los demás componentes a la DLT más fácil. Actúa como "wallet", guardando el par de claves Ethereum de un usuario, encriptadas simétricamente con un token que sólo sabe el usuario. Este token se pasa en cada llamada, habilitando la desencriptación de las claves para comunicarse con la DLT. Es de código libre y por lo tanto

lo puede desplegar cualquiera. Esto es de vital importancia ya que si no sería un punto de centralización no deseado.

La abstracción de la interfaz de los smart contracts incluye los métodos para leer pruebas, que el buscador utiliza. Dado que debemos verificarlas en algún punto, se extiende este componente para implementar una llamada HTTP de verificación. Esta llamada sigue el siguiente proceso:

1. Localiza la URL del inventario, llamando al servicio de resolución con la ID que viene en la prueba.
2. Llama al inventario para recoger el documento del evento correspondiente a la prueba. En este paso, el inventario puede requerir a la API que se identifique.
3. Una vez obtenido el documento, se genera un hash con el algoritmo que indica el nombre estandarizado del algoritmo contenido en la prueba.
4. Se compara el hash obtenido con el hash guardado en la prueba.
5. Devuelve true o false dependiendo de si coincide el resultado.

Buscador

El buscador usa la llamada que se acaba de detallar al tocar el primer botón de "verify" de la figura 2.14. El flujo seguido por el buscador una vez localizado un DPP es, por lo tanto, el siguiente:

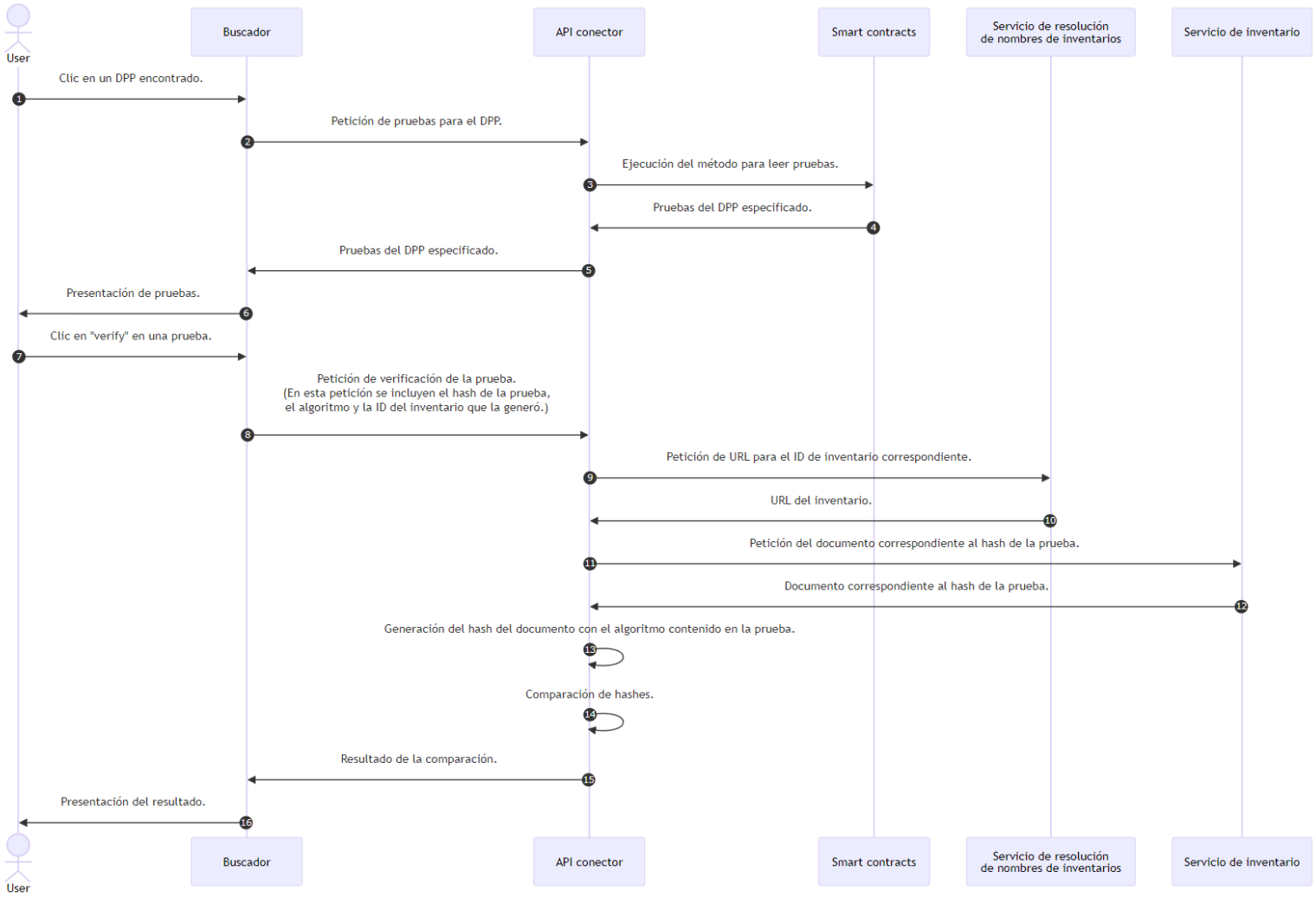

Figura 4.3: Flujo del proceso de verificación.

4.1.2. Estructura post implementación

Tras la implementación de esta parte, se puede actualizar el diagrama de arquitectura, quedando tal que:

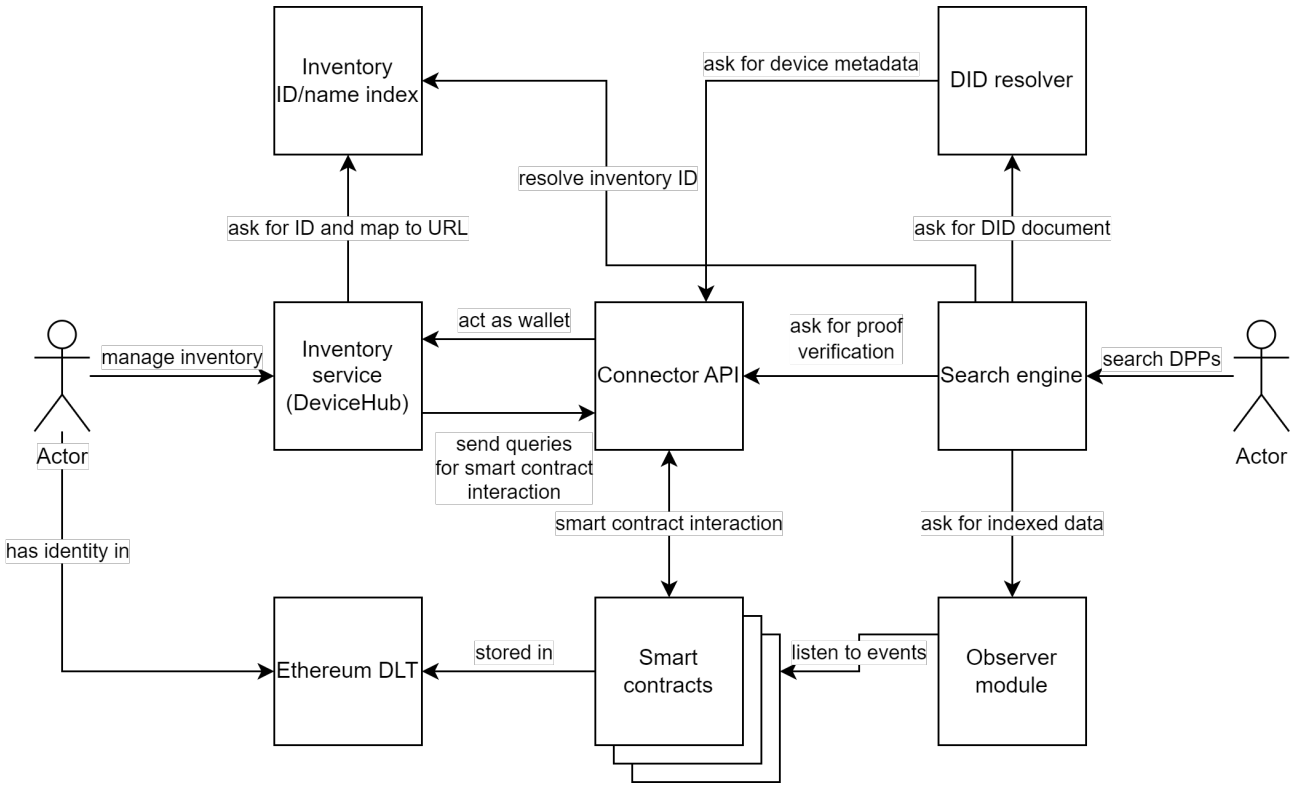

Figura 4.4: Arquitectura actualizada tras implementar la verificación de eventos.

El nuevo módulo habilita la posibilidad de encontrar los documentos correspondientes a las pruebas y se ha definido e implementado, mediante la extensión de los smart contracts, el API connector y el buscador, el proceso de verificación de eventos.

4.2. Verificación de actores

4.2.1. Componentes necesarios de un sistema de credenciales verificables

La verificación de procedencia es más compleja. Para usar un sistema de credenciales verificables en este proyecto, se detectan ciertos componentes con los que debe contar la arquitectura.

Registro verificable

Un registro verificable en forma de DLT. Este registro verificable debe acomodar algún método DID definido. La razón es que, de acuerdo a la especificación ofrecida por el W3C [4],

los actores de un sistema de credenciales verificables se identifican con un DID. Por lo tanto, a partir del DID de un actor, se debe poder llegar al documento que lo describe mediante el registro verificable. Como ejemplo, un verifier que dispone de una credencial y del DID del sujeto que la ha firmado, puede llegar a obtener la clave pública del sujeto resolviendo el DID en un documento a través del registro verificable. Así, podrá verificar la firma.

Dado que ya se dispone de una red Ethereum desplegada, se optará por utilizarla como registro verificable para el sistema de credenciales. En este contexto, se encuentra el método "did:ethr" [20], que mediante un contrato que actúa como registro [21], convierte cualquier dirección Ethereum de manera implícita en un DID. Llamando a este contrato, se puede encontrar el documento de cualquier DID de este método.

En definitiva, para contar con este elemento, simplemente se desplegará el contrato necesario a la red.

Estructura de una credencial

Se declaran primero los campos que definen el núcleo la estructura de una credencial sin firmar:

```
{
  "credentialSubject": {
    "id": "did:ethr:0x03753dfa2506c91498e038906331b75e2cf9b69c83a2569fd0e3c8a61471d67907",
    "role": "operator"
  },
  "type": [
    "VerifiableCredential"
  ],
}
```

Figura 4.5: Estructura de datos del núcleo de una credencial.

Esto da al actor representado por el DID que aparece en el campo "id", el rol de operator. Si se le añade el DID del emisor de la credencial, la fecha de emisión, el contexto definido por el W3C (en JSON-LD [22]), y se firma en un formato compatible, añadiendo un campo "proof", se obtiene una credencial verificable completa:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "issuer": {
    "id": "did:ethr:0x0272b6c8c5390188a770ef439164117f755a051ca4661b80f43ae4752f349a16a8"
  },
  "credentialSubject": {
    "id": "did:ethr:0x03753dfa2506c91498e038906331b75e2cf9b69c83a2569fd0e3c8a61471d67907",
    "role": "operator"
  },
  "issuanceDate": "2024-01-08T17:30:29.712Z",
  "type": [
    "VerifiableCredential"
  ],
  "proof": {
    "verificationMethod": "did:ethr:0x0272b6c8c5390188a770ef439164117f755a051ca4661b80f43ae475
        2f349a16a8#controller",
    "created": "2024-01-08T17:30:29.712Z",
    "proofPurpose": "assertionMethod",
    "type": "EthereumEip712Signature2021",
    "proofValue": "0x87c48652f053571909ab14f6dcf3bb12592ba7b57fba26eb19bed089db2f2e5f4be200066
        358fbc3d43661f109dcca18979327b0fc836eadc854f42e597da6581b",
    "eip712": {
      "domain": {
        "chainId": 457,
        "name": "VerifiableCredential",
        "version": "1"
      },
      "types": {
        "EIP712Domain": [
          ...
        ]
      },
      "primaryType": "VerifiableCredential"
    }
  }
}
```

Figura 4.6: Estructura de las credenciales verificables a utilizar.

Se ha omitido el campo "EIP712Domain" por brevedad. Se puede ver en el anexo A.4.

En este caso, siguiendo con el contexto Ethereum, se ha optado por utilizar una firma detallada en el "Ethereum Improvement Proposal 712" [23], que es compatible con claves Ethereum. Esto aporta la ventaja que en el momento de verificación puede no ser necesario contactar con el registro verificable, ya que el propio DID contiene la clave pública que puede haber sido utilizada para firmar, proporcionando así un mecanismo de verificación offline.

Framework compatible

Para no tener que implementar los mecanismos de firma y verificación desde cero, se ha optado por usar un framework compatible con el método DID y de firma escogidos. En este caso se trata de Veramo, mencionado en la sección 3.4.2.

El framework Veramo permite crear un objeto llamado “Agente”, al que se le puede configurar el entorno del registro verificable y uno o varios DID con los que emitir credenciales. De la misma manera, disponiendo de una credencial y conexión al registro verificable, se puede verificar de manera sencilla.

4.2.2. Verificación antes de una escritura

Tal como se detalla en la sección 3.3, para limitar las escrituras de eventos en la DLT a solo los actores con una credencial válida, el smart contract correspondiente a un dispositivo ha de saber si el actor que está intentando escribir la tiene o no. Como un smart contract solo puede obtener datos de dentro de la red Ethereum, necesita de otro contrato que se lo diga. Este otro contrato ha de estar actualizado desde el mundo exterior por un actor al que se llama “Oráculo”. En este punto, la implementación va a diferir del diseño bajo las siguientes consideraciones.

En primer lugar, en la fase de diseño se habló del caso de que si el sistema de credenciales basaba su registro verificable en la misma red que los demás smart contracts, se podría acceder a qué credenciales se habían emitido de manera implícita. Esto se ha visto que no es posible en este momento, ya que en el proceso de emisión de credenciales no interviene el registro. Solo interviene en el proceso de verificación, guardando las claves necesarias relacionadas con los DID.

Segundo, se proponía que el oráculo observaría estas emisiones de credenciales y mantendría a un contrato actualizado en todo momento. En este caso, al igual que en el anterior punto, no hay nada que observar directamente, dado que solo el emisor sabe el momento en el que se emite una credencial. Aún así, este oráculo puede tomar un rol “pasivo” aceptando notificaciones de los propios holders de que se ha emitido una credencial, verificándolas y actualizando el estado del contrato.

En resumen, hace falta implementar un nuevo tipo de contrato, que mantiene los roles de los actores actualizados, modificar los contratos de los dispositivos para que obtengan información sobre este último, y desarrollar el oráculo.

Smart contracts

Para el nuevo tipo de contrato, se desarrollan estructuras que dan soporte a registrar si existen credenciales de cada tipo para cada actor. Si se registra la existencia, se debe registrar también al actor que la ha emitido.

```
struct Credential{
    address issuer;
    bool valid;
}

struct Actor{
    bool exists;
    Credential[] issuer_credentials;
    Credential[] operator_credentials;
    Credential[] witness_credentials;
    Credential[] verifier_credentials;
}

mapping(address => bool) issuers;
mapping(address => bool) operators;
mapping(address => bool) witnesses;
mapping(address => bool) verifiers;
mapping(address => Actor) actors;

address[] trustAnchor;
```

Figura 4.7: Estructura de las credenciales verificables a utilizar.

Por supuesto, se deben registrar además los "trust anchors" del sistema, correspondientes a los TAOs definidos en la sección 3.4.1, que actúan como raíz de confianza. De esta forma, esta estructura proporciona una cadena de confianza. Empezando por la credencial de cualquier actor, siguiendo el camino de emisores se debe poder llegar a una raíz.

Por otro lado, el contrato cuenta con una interfaz de métodos para poder modificar estas variables, limitando su acceso a solo actores con los roles de TAO o TI.

Respecto a las modificaciones a los contratos correspondientes a los dispositivos, se les añade en su constructor la dirección del nuevo contrato, exponiendo así llamadas que permiten averiguar el rol de un actor. Así, se limitan las escrituras a estos contratos a actores con los roles operator o verifier.

Oráculo

El oráculo se implementa como una llamada HTTP al API connector. Esta llamada debe proporcionar la credencial verificable para que el API connector pueda actualizar el nuevo contrato. Para que esto funcione, el API connector debe utilizar la identidad de un actor autorizado para escribir en ese contrato (TAO o TI). Con lo cual solo hay dos opciones:

- La llamada la realiza un TAO o TI, haciendo que el API connector actúe en su nombre.
- Existe una instancia del API connector configurada para utilizar siempre la identidad de un TAO o TI. En este caso la llamada la puede hacer el holder de la credencial.

Por simplicidad, se contempla solo el primer caso en esta primera implementación. El flujo de emisión de credenciales es entonces el siguiente:

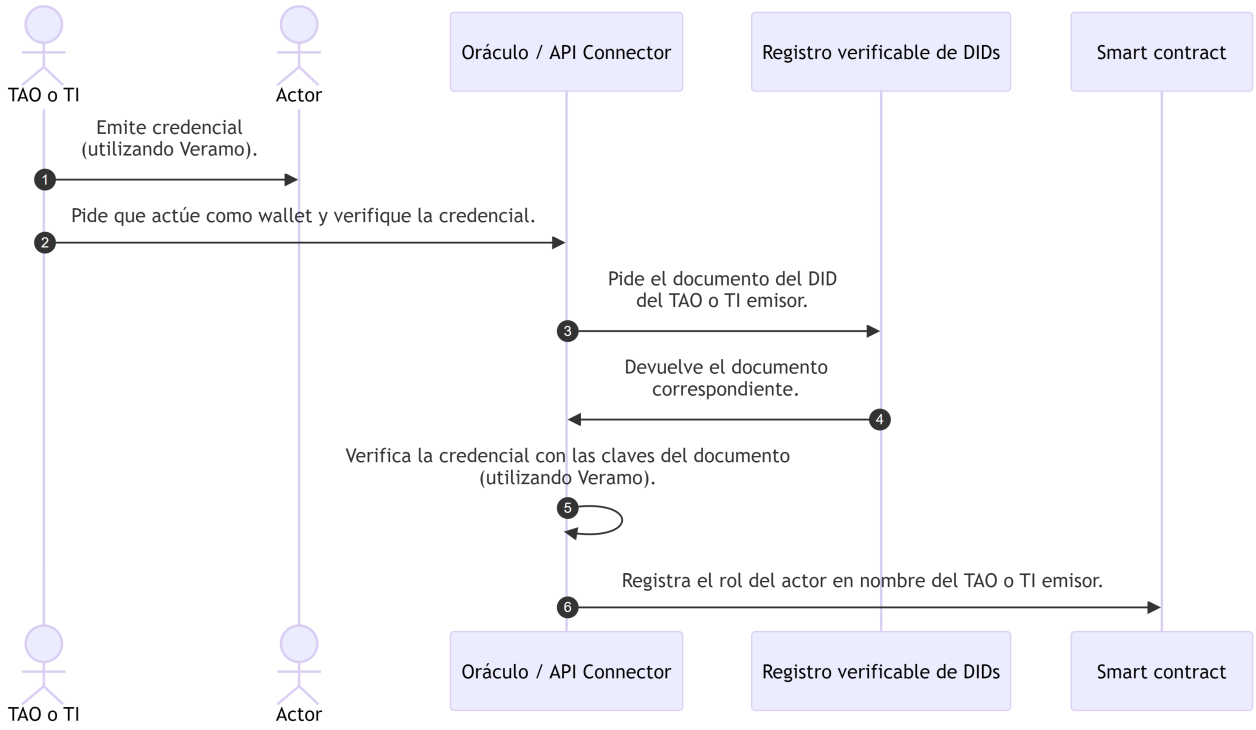

Figura 4.8: Flujo del proceso de emisión de una credencial.

Y el flujo de una escritura es el siguiente:

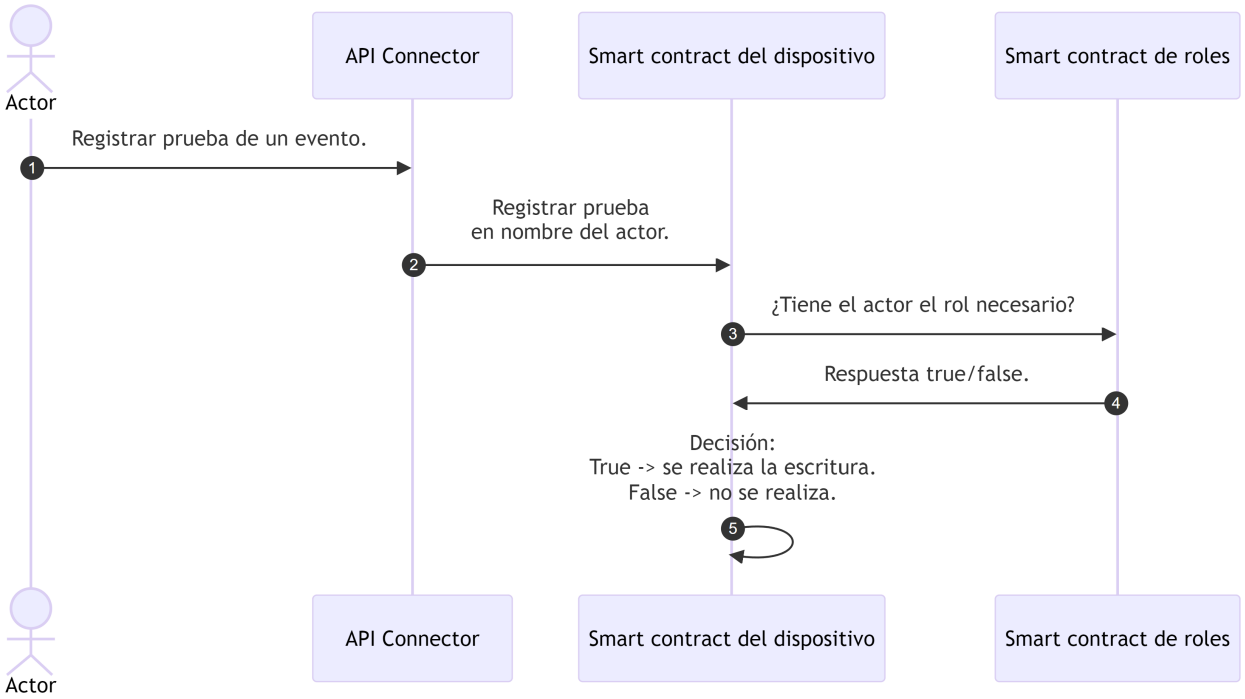

Figura 4.9: Flujo del proceso de escritura de una prueba.

4.2.3. Verificación tras la escritura

Recordando, en esta versión se permiten todas las escrituras de pruebas en los contratos de los dispositivos, y es responsabilidad del verifier decidir si una prueba es válida o no. Para esto, tiene que conseguir la credencial del actor que ha hecho la escritura y verificarla por sí mismo.

Este caso permite una implementación menos rígida y se deja a gusto del verificador cómo implementa su flujo de trabajo. Se proporcionan scripts de ejemplo usando el agente de Veramo para la verificación, pero dado que en general el único punto de acceso a la DLT en este proyecto es el API connector, se implementa una llamada en este para hacer la verificación utilizando Veramo. El flujo es el siguiente:

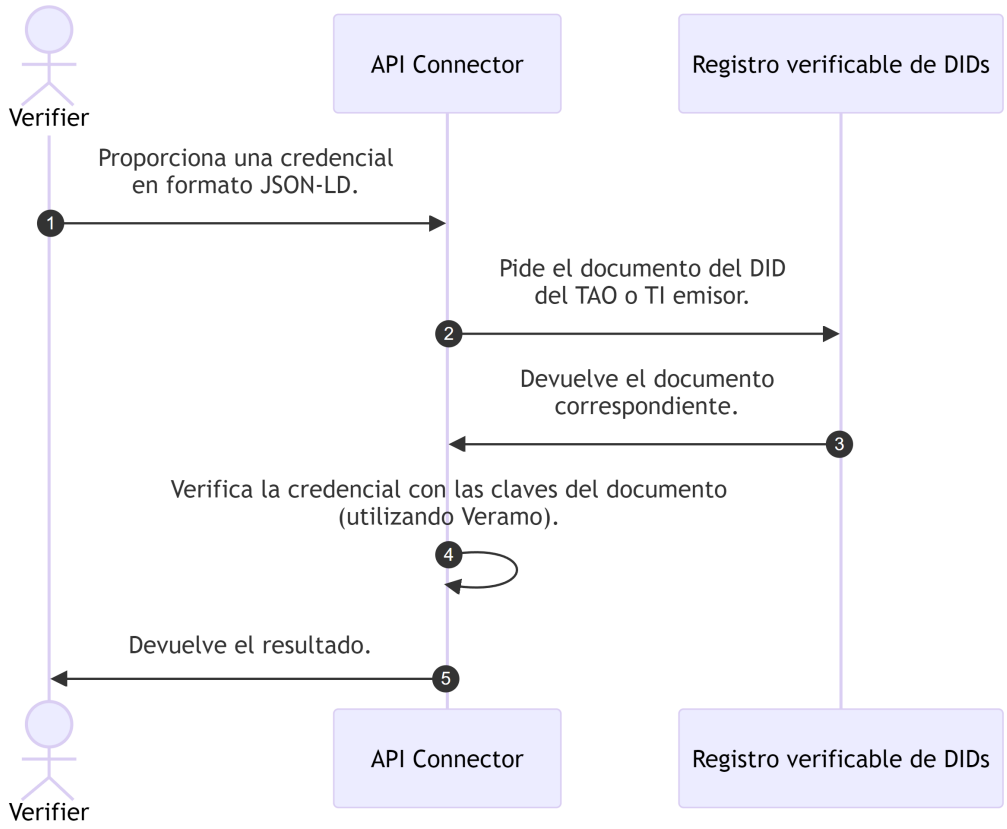

Figura 4.10: Flujo de verificación de una credencial por un verifier.

El proceso de obtención de la credencial no se detalla. En este caso se considera que cada proceso de auditoría puede obtenerla de forma diferente. Sin embargo, se proporciona un mecanismo en la interfaz de usuario para elegir la credencial como un archivo y producir la llamada al API connector.

4.2.4. Estructura final del sistema

En ambas implementaciones de la verificación de procedencia de los datos, la estructura del sistema queda de la siguiente forma:

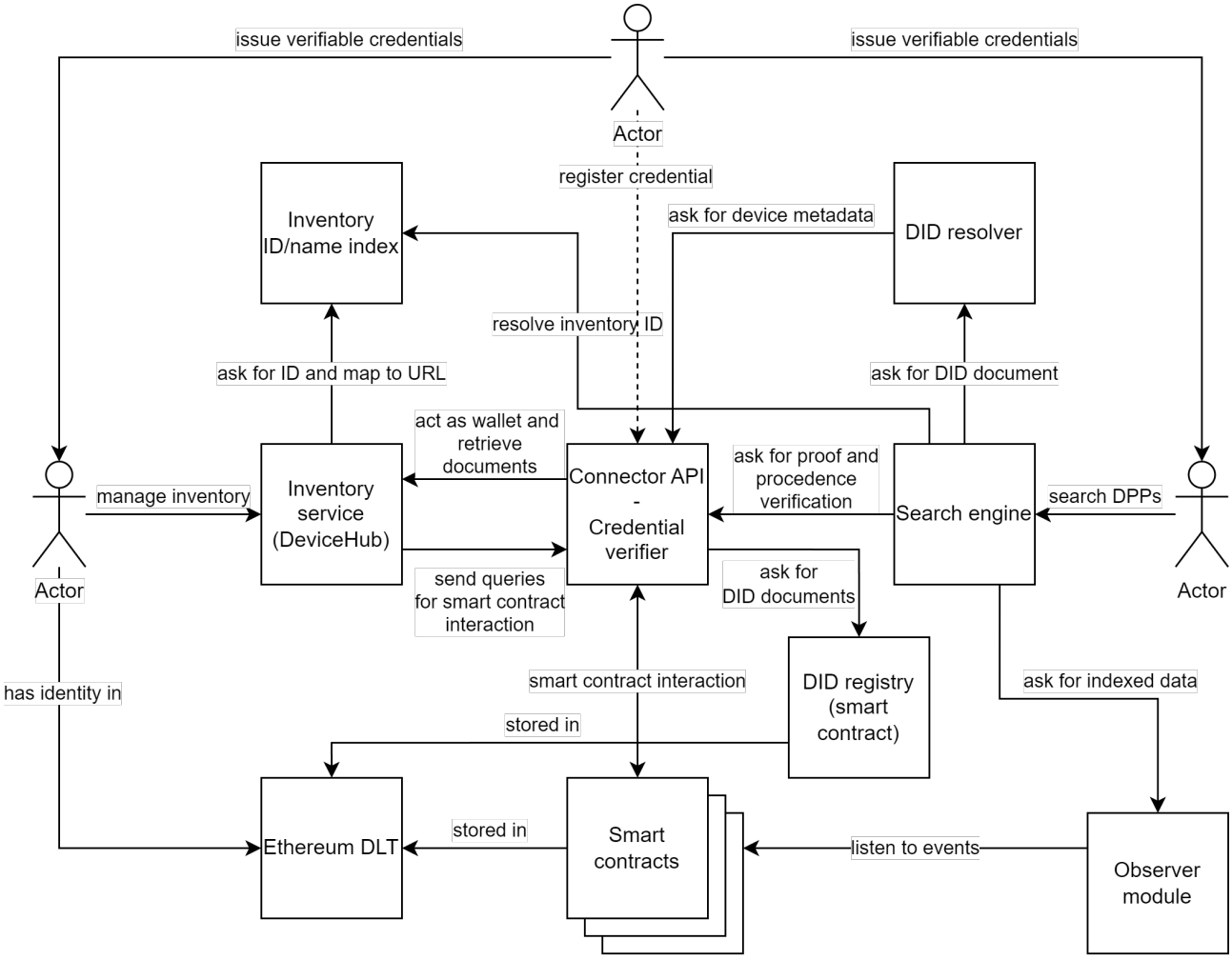

Figura 4.11: Arquitectura final del sistema.

A destacar, se han añadido el registro de DIDs de soporte a las credenciales verificables y actores que emiten credenciales verificables. Se denota con una línea de puntos el único proceso que pasa solo en la versión de verificación de actores antes de la escritura, ya que solo en ese caso se presentan al oráculo las credenciales para su registro en la cadena.

Capítulo 5

Resultados y conclusiones

5.1. Producto final

El producto final, tal como se buscaba, es la ampliación del buscador con los procesos de verificación implementados. Respecto a la figura 2.14, se pueden ver las siguientes diferencias:

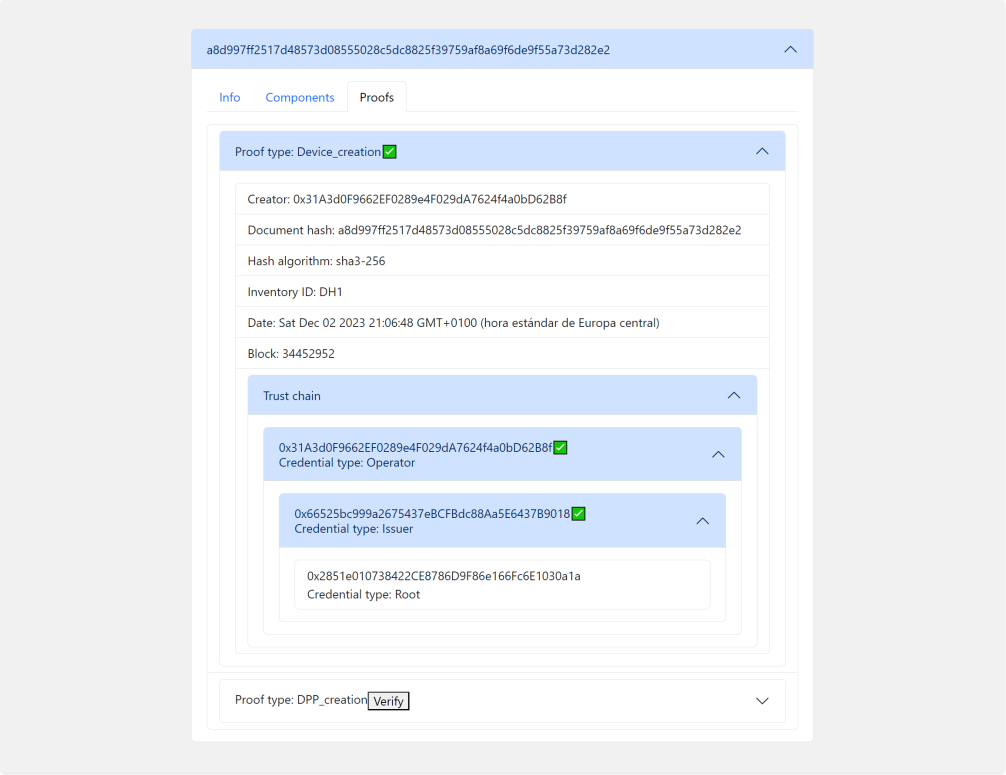

Figura 5.1: Buscador con los procesos de verificación implementados.

En la figura aparecen ticks verdes que simbolizan la finalización correcta de los procesos de verificación. Arriba, el proceso de verificación del evento y abajo el proceso de verificación de actores. Se observa cómo se puede llegar a verificar toda la cadena de confianza hasta llegar a una raíz.

5.2. Cumplimiento de objetivos y reflexiones

Visto el producto final del anterior apartado, se puede comprobar que los objetivos se han cumplido en su mayoría. Con este producto, contamos con procesos de verificación de la información descentralizados que funcionan.

Los objetivos que se consideran cumplidos en menor medida son los de valoración de diferentes frameworks. Se ha optado por la utilización de uno que funciona y se ajusta a la arquitectura de partida del proyecto, pero no se han valorado ventajas e inconvenientes frente a otros.

Tampoco se ha hablado extensamente del diseño de la estructura de credenciales. Se ha definido simplemente un campo "role", que es lo mínimo viable para la implementación del sistema. Gracias al formato JSON-LD, se podrían haber definido esquemas más complejos.

Sin embargo, este ha sido un proceso de investigación del que se partía con un conocimiento básico y donde se han podido profundizar los conocimientos sobre sistemas de credenciales verificables, tanto a nivel conceptual como a nivel de implementación y uso.

5.3. Trabajo futuro

Como potenciales mejoras futuras, la más evidente sería, en el caso de verificación de un actor previa escritura, el dar la oportunidad al actor, que es el holder de la credencial, de presentarla y añadirla al sistema cuando quiera. Tal como se ha implementado, esto no es posible y la responsabilidad cae demasiado en un punto (el issuer). Una solución podría ser que se creara un sistema de consenso con el que añadir esta información al smart contract de roles. Por ejemplo, si más de la mitad de TAOs y/o TIs del sistema verifican la credencial.

De la misma manera, si una credencial pertenece a un holder, se pueden explorar mecanismos de generación de las llamadas verifiable presentations. Estas son estructuras firmadas por el holder que permiten presentar de manera verificable los "claims" hechos por una o varias credenciales verificables.

Otra gran mejora sería explorar mecanismos de revocación. Estos pasarían por listas de credenciales verificables que revocan cierta credencial (al estilo de las CRLs en SSL) en llamadas al oráculo con las propias revocaciones por parte de los issuers.

Dejando de lado lo más técnico, también sería necesario ponerle un poco más de atención a la interfaz de usuario del buscador. Se le ha dado poca importancia y el foco ha sido crear algo funcional como demostrador. No obstante, un acabado más elegante de esa interfaz daría un aspecto más profesional a todo el proyecto.

Finalmente, decir que este es un proyecto que se ha enfocado en algo muy concreto de un caso de uso que abarca muchas cosas muy complejas. Cosas como la propia definición de los DPPs, la captura de información del hardware o de lo que realmente es un evento interesante, que están aún bastante en el aire y sobre lo que aún queda mucho trabajo por hacer en el proyecto de investigación en que se enmarca este trabajo. Aún así, se considera que probar técnicamente la viabilidad de la verificación tal como se ha tratado de hacer en este proyecto es un paso adelante que era necesario.

Bibliografía

[1] **European Commission** - Green claims. [En línea] Consultado el 10/10/2023. Disponible en: https://environment.ec.europa.eu/topics/circular-economy/green-claims_en

[2] **European Commission** - Ecodesign for Sustainable Products Regulation (2022). [En línea] Consultado el 10/10/2023. Disponible en: https://commission.europa.eu/energy-climate-change-environment/standards-tools-and-labels/products-labelling-rules-and-requirements/sustainable-products/ecodesign-sustainable-products-regulation_en

[3] **NGI Search** - About. [En línea] Consultado el 10/10/2023. Disponible en: https://www.ngisearch.eu/view/Main/

[4] **W3C** - Verifiable credentials data model. Por Manu Sporny, Dave Longley, David Chadwick. [En línea] Consultado el 10/10/2023. Disponible en: https://www.w3.org/TR/vc-data-model/

[5] **IOTA Foundation** - What is IOTA. [En línea] Consultado el 10/10/2023. Disponible en: https://www.iota.org/get-started/what-is-iota

[6] **European Commission** - Commission welcomes provisional agreement for more sustainable, repairable and circular products. [En línea] Consultado el 9/1/2024. Disponible en: https://ec.europa.eu/commission/presscorner/detail/en/ip_23_6257

[7] **W3C** - Decentralized Identifiers (DIDs) v1.0. Por Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, Christopher Allen. [En línea] Consultado el 8/1/2024. Disponible en: https://www.w3.org/TR/did-core/

[8] **GitHub** - eReuse/devicehub-teal. [En línea] Consultado el 7/11/2023. Disponible en: https://github.com/eReuse/devicehub-teal

[9] **IANA** - Named Information Hash Algorithm Registry. [En línea] Consultado el 7/11/2023. Disponible en: https://www.iana.org/assignments/named-information/named-information.xhtml

[10] **IPFS** - An open system to manage data without a central server. [En línea] Consultado el 7/11/2023. Disponible en: https://ipfs.tech/

[11] **Wikipedia** - Lazy evaluation. [En línea] Consultado el 7/11/2023. Disponible en: https://en.wikipedia.org/wiki/Lazy_evaluation

[12] **Sovrin** - Developers. [En línea] Consultado el 7/11/2023. Disponible en: https://sovrin.org/developers/

[13] **EBSI** - Verifiable Credentials Framework. [En línea] Consultado el 7/11/2023. Disponible en: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/EBSI+Verifiable+Credentials

[14] **IOTA Wiki** - Verifiable Credentials Overview. [En línea] Consultado el 7/11/2023. Disponible en: https://wiki.iota.org/identity.rs/concepts/verifiable_credentials/overview/

[15] **Veramo** - Introduction. [En línea] Consultado el 7/11/2023. Disponible en: https://veramo.io/docs/basics/introduction

[16] **walt.id Docs** - Introduction. [En línea] Consultado el 7/11/2023. Disponible en: https://docs.walt.id/v/ssikit/ssi-kit/readme

[17] **SpruceID** - DIDKit. [En línea] Consultado el 7/11/2023. Disponible en: https://www.spruceid.dev/didkit/didkit

[18] **Hyperledger** - Aries. [En línea] Consultado el 7/11/2023. Disponible en: https://www.hyperledger.org/projects/aries

[19] **Solidity** - Documentation. [En línea] Consultado el 5/12/2023. Disponible en: https://docs.soliditylang.org/en/v0.8.23/

[20] **GitHub** - ETHR DID Method Specification. [En línea] Consultado el 8/1/2024. Disponible en: https://github.com/decentralized-identity/ethr-did-resolver/blob/master/doc/did-method-spec.md

[21] **GitHub** - Ethereum DID Registry. [En línea] Consultado el 8/1/2024. Disponible en: https://github.com/uport-project/ethr-did-registry

[22] **W3C** - JSON-LD 1.1. Por Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, Niklas Lindström. [En línea] Consultado el 8/1/2024. Disponible en: https://www.w3.org/TR/json-ld/

[23] **Ethereum Improvement Proposals** - EIP-712: Typed structured data hashing and signing. [En línea] Consultado el 8/1/2024. Disponible en: https://eips.ethereum.org/EIPS/eip-712

Apéndice A

Código

A.1. Repositorio

El repositorio de código utilizado se encuentra en:
`https://gitlab.com/dsg-upc/tfm-ict-vc-snapshot`.

A.2. Despliegue del sistema

Todo el despliegue se hace en contenedores por Docker. Se puede encontrar una guía de cada componente en los ficheros "README" del repositorio.

En el directorio "demo_eel" se puede encontrar una aplicación demo para interactuar con la interfaz del API connector. Se ofrece también en forma de librería Python, disponible en:

`https://test.pypi.org/project/ereuseapitest/`

Adicionalmente, el directorio "veramo" contiene ejemplos de su uso para generar y verificar credenciales.

A.3. Smart contract de dispositivos

Se detallan aquí las partes más importantes de la interfaz del smart contract que representa a un dispositivo.

```
    struct DevData {
        string chid;
        string phid;
        uint registerDate;
        address owner;
        bool deregistered;
    }
    struct GenericProofData {
        string chid;
        string phid;
        address issuerID;
        string inventoryID;
        string documentHashAlgorithm;
        string documentHash;
        string documentType;
        uint timestamp;
        uint blockNumber;
    }
    struct Service {
        string endpoint;
        string type_;
        string description;
        string fragment;
    }
    struct DidData {
        address contractAddress;
        address controller;
        string chid;
        Service[] services;
        uint chainid;
    }
    // variables ------------------------------------------------------------
    DevData data;
    string[] phids;
    GenericProofData[] genericProofs;
    Service[] services;
```

Figura A.1: Estructuras para guardar pruebas y el documento DID.

Con estas estructuras y arrays se guardan los identificadores del dispositivo, a la vez que toda la información necesaria para generar un documento DID.

```
    modifier onlyOpWitVer;

    modifier onlyOpWit();

    modifier onlyOp();
```

Figura A.2: Modificadores del contrato, para acomodar roles.

Estos modificadores permiten filtrar a los usuarios y restringir lecturas o escrituras en las llamadas.

```
    function issuePassport(string calldata _phid, string calldata _documentHashAlgorithm,
        string calldata _documentHash, string calldata _inventoryID) public registered
        onlyOpWit;

    function generateGenericProof(string calldata _documentHashAlgorithm, string calldata
        _documentHash, string calldata _documentType, string calldata _inventoryID) public
        registered onlyOpWit;
```

Figura A.3: Funciones de escritura.

Estas funciones permiten la escritura de pruebas y nuevos identificadores de DPP.

```
    function getDPPs() public view onlyOpWitVer() returns (string[] memory _data);

    function getGenericProofs() public view onlyOpWitVer returns (GenericProofData[] memory
        _data);
```

Figura A.4: Funciones de lectura.

Estas funciones permiten leer pruebas e identificadores de DPP relacionados con el dispositivo.

```
    function addService(string calldata endpoint, string calldata type_, string calldata
        description, string calldata fragment) public onlyOp;

    function removeService(string calldata fragment) public onlyOp;

    function getDidData() public view returns (DidData memory _didData);
```

Figura A.5: Funciones de soporte al documento DID.

Estas funciones permiten modificar y leer el documento DID.

A.4. Ejemplo de credencial verificable

Credencial verificable de un operador:

```
{
  "issuer": {
    "id": "did:ethr:0x0272b6c8c5390188a770ef439164117f755a051ca4661b80f43ae4752f349a16a8"
  },
  "credentialSubject": {
    "id": "did:ethr:0x03753dfa2506c91498e038906331b75e2cf9b69c83a2569fd0e3c8a61471d67907",
    "role": "operator"
  },
  "issuanceDate": "2024-01-08T17:30:29.712Z",
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
```

```
  ],
  "type": [
    "VerifiableCredential"
  ],
  "proof": {
    "verificationMethod": "did:ethr:0x0272b6c8c5390188a770ef439164117f755a051ca4661b80f43ae4752f349a16a8#controller",
    "created": "2024-01-08T17:30:29.712Z",
    "proofPurpose": "assertionMethod",
    "type": "EthereumEip712Signature2021",
    "proofValue": "0x87c48652f053571909ab14f6dcf3bb12592ba7b57fba26eb19bed089db2f2e5f4be200066358fbc3d43661f109dcca18979327b0fc836eadc854f42e597da6581b",
    "eip712": {
      "domain": {
        "chainId": 457,
        "name": "VerifiableCredential",
        "version": "1"
      },
      "types": {
        "EIP712Domain": [
          {
            "name": "name",
            "type": "string"
          },
          {
            "name": "version",
            "type": "string"
          },
          {
            "name": "chainId",
            "type": "uint256"
          }
        ],
        "CredentialSubject": [
          {
            "name": "id",
            "type": "string"
          },
          {
            "name": "role",
            "type": "string"
          }
        ],
        "Issuer": [
          {
            "name": "id",
            "type": "string"
          }
        ],
        "Proof": [
          {
            "name": "created",
            "type": "string"
          },
          {
            "name": "proofPurpose",
            "type": "string"
          },
          {
            "name": "type",
            "type": "string"
          },
          {
            "name": "verificationMethod",
            "type": "string"
          }
        ],
        "VerifiableCredential": [
```

```
78            {
79              "name": "@context",
80              "type": "string[]"
81            },
82            {
83              "name": "credentialSubject",
84              "type": "CredentialSubject"
85            },
86            {
87              "name": "issuanceDate",
88              "type": "string"
89            },
90            {
91              "name": "issuer",
92              "type": "Issuer"
93            },
94            {
95              "name": "proof",
96              "type": "Proof"
97            },
98            {
99              "name": "type",
100             "type": "string[]"
101           }
102         ]
103       },
104       "primaryType": "VerifiableCredential"
105     }
106   }
107 }
```